.gitattributes

```
*.onnx filter=lfs diff=lfs merge=lfs -text
```

```
---
name: Bug report
about: Create a report to help us improve
title: "[BUG] "
labels: bug
assignees: ''
---
Please provide as much details as possible, this will help us to deliver a
fix as soon as possible.
Thank you!
**Describe the bug**
A clear and concise description of what the bug is.
**Log and Stack trace**
Please provide a log and a stack trace (with exception), if applicable.
**To Reproduce**
Please provide a relevant code snippets to reproduce this bug.
**Expected behavior**
A clear and concise description of what you expected to happen.
**Please complete the following information:**
- LangChain4j version: e.g. 0.23.0
- Java version: e.g. 11
- Spring Boot version (if applicable): e.g. 2.7.14
**Additional context**
Add any other context about the problem here.
```

.github\ISSUE_TEMPLATE\feature_request.md

```
---
name: Feature request
about: Suggest an idea for this project
title: "[FEATURE] "
labels: enhancement
assignees: ''
---
**Is your feature request related to a problem? Please describe.**
A clear and concise description of what the problem is. Ex. I'm always
frustrated when [...]
**Describe the solution you'd like**
A clear and concise description of what you want to happen.
**Describe alternatives you've considered**
A clear and concise description of any alternative solutions or features
you've considered.
**Additional context**
Add any other context or screenshots about the feature request here.
```

.github\workflows\main.yaml

```
name: Java CI
on: [pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'adopt'
      - name: Test
        run: mvn --batch-mode test
  # For checking some compliance things (require a recent JDK due to plugins
so in a separate step)
  compliance:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
        # Check we only rely on permissive licenses in the main parts of the
library:
      - name: License Compliance
        run: mvn -P compliance org.honton.chas:license-maven-plugin:compliance
# TODO's
#  - setup integration tests
#     - these require an openAI (and hugging face, etc) token
#     - do so that they always run for commits on main
#     - make the running be manually triggered for PRs (we don't want to burn
through credits)
```

.prettierrc

```
{
  "printWidth": 120
}
```

langchain4j\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j</artifactId>
    <packaging>jar</packaging>
    <name>langchain4j</name>
    <description>Java implementation of LangChain: Integrate your Java
application with countless AI tools and services
        smoothly
    </description>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.squareup.retrofit2</groupId>
            <artifactId>retrofit</artifactId>
        </dependency>
        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.opennlp</groupId>
            <artifactId>opennlp-tools</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.pdfbox</groupId>
            <artifactId>pdfbox</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
            <version>5.2.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>5.2.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
```

```
    <artifactId>poi-scratchpad</artifactId>
    <version>5.2.3</version>
</dependency>
<dependency>
    <groupId>org.jsoup</groupId>
    <artifactId>jsoup</artifactId>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
</dependency>
<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-open-ai</artifactId>
    <version>${project.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.tinylog</groupId>
    <artifactId>tinylog-impl</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.tinylog</groupId>
    <artifactId>slf4j-tinylog</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>s3</artifactId>
    <exclusions>
```

```
                <!-- Exclusion due to CWE-295 vulnerability -->
                <exclusion>
                    <groupId>io.netty</groupId>
                    <artifactId>netty-handler</artifactId>
                </exclusion>
                <!-- due to CVE-2023-44487 vulnerability -->
                <exclusion>
                    <groupId>io.netty</groupId>
                    <artifactId>netty-codec-http2</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>testcontainers</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>localstack</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j\src\main\java\dev\langchain4j\agent\tool\ToolExecutor.java

```
package dev.langchain4j.agent.tool;
import dev.langchain4j.internal.Json;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Parameter;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Map;
public class ToolExecutor {
    private static final Logger log =
LoggerFactory.getLogger(ToolExecutor.class);
    private final Object object;
    private final Method method;
    public ToolExecutor(Object object, Method method) {
        this.object = object;
        this.method = method;
    }
    public String execute(ToolExecutionRequest toolExecutionRequest) {
        log.debug("About to execute {}", toolExecutionRequest);
        // TODO ensure this method never throws exceptions
        Object[] arguments = prepareArguments(ToolExecutionRequestUtil.argumen
tsAsMap(toolExecutionRequest.arguments()));
        try {
            String result = execute(arguments);
            log.debug("Tool execution result: {}", result);
            return result;
        } catch (IllegalAccessException e) {
            try {
                method.setAccessible(true);
                String result = execute(arguments);
                log.debug("Tool execution result: {}", result);
                return result;
            } catch (IllegalAccessException e2) {
                throw new RuntimeException(e2);
            } catch (InvocationTargetException e2) {
                Throwable cause = e2.getCause();
                log.error("Error while executing tool", cause);
                return cause.getMessage();
            }
        } catch (InvocationTargetException e) {
            Throwable cause = e.getCause();
            log.error("Error while executing tool", cause);
            return cause.getMessage();
        }
    }
    private String execute(Object[] arguments) throws IllegalAccessException,
InvocationTargetException {
        Object result = method.invoke(object, arguments);
        if (method.getReturnType() == void.class) {
            return "Success";
        }
        return Json.toJson(result);
    }
    private Object[] prepareArguments(Map<String, Object> argumentsMap) {
        Parameter[] parameters = method.getParameters();
```

```
        Object[] arguments = new Object[parameters.length];
        for (int i = 0; i < parameters.length; i++) {
            String parameterName = parameters[i].getName();
            if (argumentsMap.containsKey(parameterName)) {
                Object argument = argumentsMap.get(parameterName);
                Class<?> parameterType = parameters[i].getType();
                // Gson always parses numbers into the Double type. If the
parameter type is not Double, a conversion attempt is made.
                if (argument instanceof Double && !(parameterType ==
Double.class || parameterType == double.class)) {
                    Double doubleValue = (Double) argument;
                    if (parameterType == Float.class || parameterType ==
float.class) {
                        if (doubleValue < -Float.MAX_VALUE || doubleValue >
Float.MAX_VALUE) {
                            throw new IllegalArgumentException("Double value
" + doubleValue + " is out of range for the float type");
                        }
                        argument = doubleValue.floatValue();
                    } else if (parameterType == BigDecimal.class) {
                        argument = BigDecimal.valueOf(doubleValue);
                    }
                    // Allow conversion to integer types only if double value
has no fractional part
                    if (hasNoFractionalPart(doubleValue)) {
                        if (parameterType == Integer.class || parameterType
== int.class) {
                            if (doubleValue < Integer.MIN_VALUE ||
doubleValue > Integer.MAX_VALUE) {
                                throw new IllegalArgumentException("Double
value " + doubleValue + " is out of range for the integer type");
                            }
                            argument = doubleValue.intValue();
                        } else if (parameterType == Long.class ||
parameterType == long.class) {
                            if (doubleValue < Long.MIN_VALUE || doubleValue >
Long.MAX_VALUE) {
                                throw new IllegalArgumentException("Double
value " + doubleValue + " is out of range for the long type");
                            }
                            argument = doubleValue.longValue();
                        } else if (parameterType == Short.class ||
parameterType == short.class) {
                            if (doubleValue < Short.MIN_VALUE || doubleValue
> Short.MAX_VALUE) {
                                throw new IllegalArgumentException("Double
value " + doubleValue + " is out of range for the short type");
                            }
                            argument = doubleValue.shortValue();
                        } else if (parameterType == Byte.class ||
parameterType == byte.class) {
                            if (doubleValue < Byte.MIN_VALUE || doubleValue >
Byte.MAX_VALUE) {
                                throw new IllegalArgumentException("Double
value " + doubleValue + " is out of range for the byte type");
                            }
                            argument = doubleValue.byteValue();
                        } else if (parameterType == BigInteger.class) {
                            argument =
BigDecimal.valueOf(doubleValue).toBigInteger();
                        }
```

```
                    }
                }
                arguments[i] = argument;
            }
        }
        return arguments;
    }
    private static boolean hasNoFractionalPart(Double doubleValue) {
        return doubleValue.equals(Math.floor(doubleValue));
    }
}
```

langchain4j\src\main\java\dev\langchain4j\chain\ConversationalChain.java

```
package dev.langchain4j.chain;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import lombok.Builder;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * A chain for interacting with a specified {@link ChatLanguageModel} while
maintaining a memory of the conversation.
 * Includes a default {@link ChatMemory} (a message window with maximum 10
messages), which can be overridden.
 */
public class ConversationalChain implements Chain<String, String> {
    private final ChatLanguageModel chatLanguageModel;
    private final ChatMemory chatMemory;
    @Builder
    private ConversationalChain(ChatLanguageModel chatLanguageModel,
ChatMemory chatMemory) {
        this.chatLanguageModel = ensureNotNull(chatLanguageModel,
"chatLanguageModel");
        this.chatMemory = chatMemory == null ?
MessageWindowChatMemory.withMaxMessages(10) : chatMemory;
    }
    @Override
    public String execute(String userMessage) {
        chatMemory.add(userMessage(ensureNotBlank(userMessage,
"userMessage")));
        AiMessage aiMessage =
chatLanguageModel.generate(chatMemory.messages()).content();
        chatMemory.add(aiMessage);
        return aiMessage.text();
    }
}
```

langchain4j\src\main\java\dev\langchain4j\chain\ConversationalRetrievalChain.java

```
package dev.langchain4j.chain;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.input.PromptTemplate;
import dev.langchain4j.retriever.Retriever;
import lombok.Builder;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
import static java.util.stream.Collectors.joining;
/**
 * A chain for interacting with a specified {@link ChatLanguageModel} based on the information provided by a specified {@link Retriever}.
 * Includes a default {@link PromptTemplate}, which can be overridden.
 * Includes a default {@link ChatMemory} (a message window with maximum 10 messages), which can be overridden.
 */
public class ConversationalRetrievalChain implements Chain<String, String> {
    private static final PromptTemplate DEFAULT_PROMPT_TEMPLATE = PromptTemplate.from(
            "Answer the following question to the best of your ability: {{question}}\n" +
                    "\n" +
                    "Base your answer on the following information:\n" +
                    "{{information}}");
    private final ChatLanguageModel chatLanguageModel;
    private final ChatMemory chatMemory;
    private final PromptTemplate promptTemplate;
    private final Retriever<TextSegment> retriever;
    @Builder
    public ConversationalRetrievalChain(ChatLanguageModel chatLanguageModel,
                                        ChatMemory chatMemory,
                                        PromptTemplate promptTemplate,
                                        Retriever<TextSegment> retriever) {
        this.chatLanguageModel = ensureNotNull(chatLanguageModel, "chatLanguageModel");
        this.chatMemory = chatMemory == null ? MessageWindowChatMemory.withMaxMessages(10) : chatMemory;
        this.promptTemplate = promptTemplate == null ? DEFAULT_PROMPT_TEMPLATE : promptTemplate;
        this.retriever = ensureNotNull(retriever, "retriever");
    }
    @Override
    public String execute(String question) {
        question = ensureNotBlank(question, "question");
        List<TextSegment> relevantSegments = retriever.findRelevant(question);
        Map<String, Object> variables = new HashMap<>();
        variables.put("question", question);
        variables.put("information", format(relevantSegments));
        UserMessage userMessage = promptTemplate.apply(variables).toUserMessage();
```

```
        chatMemory.add(userMessage);
        AiMessage aiMessage =
chatLanguageModel.generate(chatMemory.messages()).content();
        chatMemory.add(aiMessage);
        return aiMessage.text();
    }
    private static String format(List<TextSegment> relevantSegments) {
        return relevantSegments.stream()
                .map(TextSegment::text)
                .map(segment -> "..." + segment + "...")
                .collect(joining("\n\n"));
    }
}
```

langchain4j\src\main\java\dev\langchain4j\classification\EmbeddingModelTextClassifier.java

```
package dev.langchain4j.classification;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.RelevanceScore;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static dev.langchain4j.internal.ValidationUtils.ensureBetween;
import static dev.langchain4j.internal.ValidationUtils.ensureGreaterThanZero;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
import static java.util.Comparator.comparingDouble;
import static java.util.stream.Collectors.toList;
/**
 * A {@link TextClassifier} that uses an {@link EmbeddingModel} and predefined examples to perform classification.
 * Classification is done by comparing the embedding of the text being classified with the embeddings of predefined examples.
 * The classification quality improves with a greater number of examples for each label.
 * Examples can be easily generated by the LLM.
 * <p>
 * Example:
 * <pre>{@code
 * enum Sentiment {
 *     POSITIVE, NEUTRAL, NEGATIVE
 * }
 *
 *  Map<Sentiment, List<String>> examples = Map.of(
 *     POSITIVE, List.of("This is great!", "Wow, awesome!"),
 *     NEUTRAL,  List.of("Well, it's fine", "It's ok"),
 *     NEGATIVE, List.of("It is pretty bad", "Worst experience ever!")
 * );
 *
 * EmbeddingModel embeddingModel = new AllMiniLmL6V2QuantizedEmbeddingModel();
 *
 * TextClassifier<Sentiment> classifier = new EmbeddingModelTextClassifier<>(embeddingModel, examples);
 *
 * List<Sentiment> sentiments = classifier.classify("Awesome!");
 * System.out.println(sentiments); // [POSITIVE]
 * }</pre>
 *
 * @param <E> Enum that is the result of classification.
 */
public class EmbeddingModelTextClassifier<E extends Enum<E>> implements TextClassifier<E> {
    private final EmbeddingModel embeddingModel;
    private final Map<E, List<Embedding>> exampleEmbeddingsByLabel;
    private final int maxResults;
    private final double minScore;
    private final double meanToMaxScoreRatio;
    /**
     * Creates a classifier with the default values for {@link #maxResults} (1), {@link #minScore} (0)
```

```
     * and {@link #meanToMaxScoreRatio} (0.5).
     *
     * @param embeddingModel  The embedding model used for embedding both the
examples and the text to be classified.
     * @param examplesByLabel A map containing examples of texts for each
label.
     *                        The more examples, the better. Examples can be
easily generated by the LLM.
     */
    public EmbeddingModelTextClassifier(EmbeddingModel embeddingModel,
                                        Map<E, ? extends Collection<String>>
examplesByLabel) {
        this(embeddingModel, examplesByLabel, 1, 0, 0.5);
    }
    /**
     * Creates a classifier.
     *
     * @param embeddingModel      The embedding model used for embedding both
the examples and the text to be classified.
     * @param examplesByLabel     A map containing examples of texts for each
label.
     *                            The more examples, the better. Examples can
be easily generated by the LLM.
     * @param maxResults          The maximum number of labels to return for
each classification.
     * @param minScore            The minimum similarity score required for
classification, in the range [0..1].
     *                            Labels scoring lower than this value will
be discarded.
     * @param meanToMaxScoreRatio A ratio, in the range [0..1], between the
mean and max scores used for calculating
     *                            the final score.
     *                            During classification, the embeddings of
examples for each label are compared to
     *                            the embedding of the text being classified.
     *                            This results in two metrics: the mean and
max scores.
     *                            The mean score is the average similarity
score for all examples associated with a given label.
     *                            The max score is the highest similarity
score, corresponding to the example most
     *                            similar to the text being classified.
     *                            A value of 0 means that only the mean score
will be used for ranking labels.
     *                            A value of 0.5 means that both scores will
contribute equally to the final score.
     *                            A value of 1 means that only the max score
will be used for ranking labels.
     */
    public EmbeddingModelTextClassifier(EmbeddingModel embeddingModel,
                                        Map<E, ? extends Collection<String>>
examplesByLabel,
                                        int maxResults,
                                        double minScore,
                                        double meanToMaxScoreRatio) {
        this.embeddingModel = ensureNotNull(embeddingModel, "embeddingModel");
        ensureNotNull(examplesByLabel, "examplesByLabel");
        this.exampleEmbeddingsByLabel = new HashMap<>();
        examplesByLabel.forEach((label, examples) ->
                exampleEmbeddingsByLabel.put(label, examples.stream()
                        .map(example ->
```

```
embeddingModel.embed(example).content())
                        .collect(toList()))
        );
        this.maxResults = ensureGreaterThanZero(maxResults, "maxResults");
        this.minScore = ensureBetween(minScore, 0.0, 1.0, "minScore");
        this.meanToMaxScoreRatio = ensureBetween(meanToMaxScoreRatio, 0.0,
1.0, "meanToMaxScoreRatio");
    }
    @Override
    public List<E> classify(String text) {
        Embedding textEmbedding = embeddingModel.embed(text).content();
        List<LabelWithScore> labelsWithScores = new ArrayList<>();
        exampleEmbeddingsByLabel.forEach((label, exampleEmbeddings) -> {
            double meanScore = 0;
            double maxScore = 0;
            for (Embedding exampleEmbedding : exampleEmbeddings) {
                double cosineSimilarity =
CosineSimilarity.between(textEmbedding, exampleEmbedding);
                double score =
RelevanceScore.fromCosineSimilarity(cosineSimilarity);
                meanScore += score;
                maxScore = Math.max(score, maxScore);
            }
            meanScore /= exampleEmbeddings.size();
            labelsWithScores.add(new LabelWithScore(label,
aggregatedScore(meanScore, maxScore)));
        });
        return labelsWithScores.stream()
                .filter(it -> it.score >= minScore)
                // sorting in descending order to return highest score first
                .sorted(comparingDouble(labelWithScore -> 1 -
labelWithScore.score))
                .limit(maxResults)
                .map(it -> it.label)
                .collect(toList());
    }
    private double aggregatedScore(double meanScore, double maxScore) {
        return (meanToMaxScoreRatio * meanScore) + ((1 - meanToMaxScoreRatio)
* maxScore);
    }
    private class LabelWithScore {
        private final E label;
        private final double score;
        private LabelWithScore(E label, double score) {
            this.label = label;
            this.score = score;
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\code\JavaScriptCodeFixer.java

```
package dev.langchain4j.code;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
class JavaScriptCodeFixer {
    private static final Logger log =
LoggerFactory.getLogger(JavaScriptCodeFixer.class);
    static String fixIfNoLogToConsole(String code) {
        if (code.contains("\n")) {
            return fixIfNoLogToConsole(code, "\n");
        } else {
            return fixIfNoLogToConsole(code, " ");
        }
    }
    private static String fixIfNoLogToConsole(String code, String separator) {
        String[] parts = code.split(separator);
        String lastPart = parts[parts.length - 1];
        if (lastPart.startsWith("console.log")) {
            return code;
        }
        parts[parts.length - 1] = "console.log(" + lastPart.replace(";", "")
+ ");";
        String fixedCode = String.join(separator, parts);
        log.debug("The following code \"{}\" was fixed: \"{}\"", code,
fixedCode);
        return fixedCode;
    }
}
```

langchain4j\src\main\java\dev\langchain4j\code\Judge0JavaScriptEngine.java

```
package dev.langchain4j.code;
import dev.langchain4j.internal.Json;
import okhttp3.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.time.Duration;
import java.util.Base64;
import static dev.langchain4j.internal.Utils.isNullOrBlank;
class Judge0JavaScriptEngine implements CodeExecutionEngine {
    private static final Logger log =
LoggerFactory.getLogger(Judge0JavaScriptEngine.class);
    private static final MediaType MEDIA_TYPE = MediaType.parse("application/
json");
    private static final int ACCEPTED = 3;
    private final String apiKey;
    private final int languageId;
    private final OkHttpClient client;
    Judge0JavaScriptEngine(String apiKey, int languageId, Duration timeout) {
        this.apiKey = apiKey;
        this.languageId = languageId;
        this.client = new OkHttpClient.Builder()
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .callTimeout(timeout)
                .build();
    }
    @Override
    public String execute(String code) {
        String base64EncodedCode =
Base64.getEncoder().encodeToString(code.getBytes());
        Submission submission = new Submission(languageId, base64EncodedCode);
        RequestBody requestBody = RequestBody.create(Json.toJson(submission),
MEDIA_TYPE);
        Request request = new Request.Builder()
                .url("https://judge0-ce.p.rapidapi.com/submissions?
base64_encoded=true&wait=true&fields=*")
                .addHeader("X-RapidAPI-Key", apiKey)
                .post(requestBody)
                .build();
        try {
            Response response = client.newCall(request).execute();
            String responseBody = response.body().string();
            SubmissionResult result = Json.fromJson(responseBody,
SubmissionResult.class);
            if (result.status.id != ACCEPTED) {
                String error = result.status.description;
                if (!isNullOrBlank(result.compile_output)) {
                    error += "\n";
                    error += new
String(Base64.getMimeDecoder().decode(result.compile_output));
                }
                return error;
            }
            String base64EncodedStdout = result.stdout;
            if (base64EncodedStdout == null) {
                return "No result: nothing was printed out to the console";
            }
```

```
            return new
String(Base64.getMimeDecoder().decode(base64EncodedStdout.trim())).trim();
        } catch (Exception e) {
            log.warn("Error during code execution", e);
            return e.getMessage();
        }
    }
    private static class Submission {
        int language_id;
        String source_code;
        Submission(int languageId, String sourceCode) {
            this.language_id = languageId;
            this.source_code = sourceCode;
        }
    }
    private static class SubmissionResult {
        String stdout;
        Status status;
        String compile_output;
    }
    private static class Status {
        int id;
        String description;
    }
}
```

langchain4j\src\main\java\dev\langchain4j\code\Judge0JavaScriptExecutionTool.java

```
package dev.langchain4j.code;
import dev.langchain4j.agent.tool.P;
import dev.langchain4j.agent.tool.Tool;
import java.time.Duration;
import static dev.langchain4j.code.JavaScriptCodeFixer.fixIfNoLogToConsole;
import static dev.langchain4j.internal.Utils.isNullOrBlank;
/**
 * A tool that executes JS code using the Judge0 service, hosted by Rapid API.
 */
public class Judge0JavaScriptExecutionTool {
    private static final int JAVASCRIPT = 93;
    private final Judge0JavaScriptEngine engine;
    private final boolean fixCodeIfNeeded;
    /**
     * Constructs a new instance with the provided Rapid API key.
     * The code fixing feature is enabled by default.
     * Default timeout is 10 seconds.
     *
     * @param apiKey The Rapid API key. You can subscribe to the free plan
(Basic) here: https://rapidapi.com/judge0-official/api/judge0-ce/pricing
     */
    public Judge0JavaScriptExecutionTool(String apiKey) {
        this(apiKey, true, Duration.ofSeconds(10));
    }
    /**
     * Constructs a new instance with the provided Rapid API key, a flag to
control whether to fix the code, and a timeout.
     *
     * @param apiKey          Rapid API key. You can subscribe to the free
plan (Basic) here: https://rapidapi.com/judge0-official/api/judge0-ce/pricing
     * @param fixCodeIfNeeded Judge0 can return result of an execution if it
was printed to the console.
     *                        If provided JS code does not print result to
the console, attempt will be made to fix it.
     * @param timeout         Timeout for calling Judge0.
     */
    public Judge0JavaScriptExecutionTool(String apiKey, boolean
fixCodeIfNeeded, Duration timeout) {
        if (isNullOrBlank(apiKey)) {
            throw new IllegalArgumentException("Please provide a valid Rapid
API key");
        }
        this.engine = new Judge0JavaScriptEngine(apiKey, JAVASCRIPT, timeout);
        this.fixCodeIfNeeded = fixCodeIfNeeded;
    }
    @Tool("MUST be used for accurate calculations: math, sorting, filtering,
aggregating, string processing, etc")
    public String executeJavaScriptCode(
            @P("JavaScript code to execute, result MUST be printed to
console")
            String javaScriptCode
    ) {
        if (fixCodeIfNeeded) {
            javaScriptCode = fixIfNoLogToConsole(javaScriptCode);
        }
        return engine.execute(javaScriptCode);
    }
```

```
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\AbstractS3Loader.java

```
package dev.langchain4j.data.document;
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.AwsSessionCredentials;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.S3ClientBuilder;
import java.net.URI;
import java.net.URISyntaxException;
import static dev.langchain4j.internal.Utils.isNotNullOrBlank;
import static dev.langchain4j.internal.Utils.isNullOrBlank;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
public abstract class AbstractS3Loader<T> {
    protected final String bucket;
    protected final String region;
    protected final String endpointUrl;
    protected final String profile;
    protected final boolean forcePathStyle;
    protected final AwsCredentials awsCredentials;
    protected AbstractS3Loader(Builder builder) {
        this.bucket = ensureNotBlank(builder.bucket, "bucket");
        this.region = builder.region;
        this.endpointUrl = builder.endpointUrl;
        this.profile = builder.profile;
        this.forcePathStyle = builder.forcePathStyle;
        this.awsCredentials = builder.awsCredentials;
    }
    /**
     * Initiates the loading process by configuring the AWS credentials and
S3 client,
     * then loads either a document or a list of documents.
     *
     * @return A generic object of type T, which could be a Document or a
list of Documents
     * @throws RuntimeException if there are issues with AWS credentials or
S3 client configuration
     */
    public T load() {
        AwsCredentialsProvider awsCredentialsProvider =
configureCredentialsProvider();
        S3Client s3Client = configureS3Client(awsCredentialsProvider);
        return load(s3Client);
    }
    private static AwsSessionCredentials
toAwsSessionCredentials(AwsCredentials awsCredentials) {
        return AwsSessionCredentials.create(awsCredentials.accessKeyId(),
awsCredentials.secretAccessKey(), awsCredentials.sessionToken());
    }
    private static software.amazon.awssdk.auth.credentials.AwsCredentials
toAwsCredentials(AwsCredentials awsCredentials) {
        return AwsBasicCredentials.create(awsCredentials.accessKeyId(),
awsCredentials.secretAccessKey());
    }
    protected abstract T load(S3Client s3Client);
    private AwsCredentialsProvider configureCredentialsProvider() {
```

```
        AwsCredentialsProvider provider =
DefaultCredentialsProvider.builder().build();
        if (awsCredentials != null) {
            if (awsCredentials.hasAllCredentials()) {
                provider =
StaticCredentialsProvider.create(toAwsSessionCredentials(awsCredentials));
            }else if (awsCredentials.hasAccessKeyIdAndSecretKey()) {
                provider =
StaticCredentialsProvider.create(toAwsCredentials(awsCredentials));
            }
        }
        if( isNotNullOrBlank(profile) ) {
            provider = ProfileCredentialsProvider.create(profile);
        }
        return provider;
    }
    private S3Client configureS3Client(AwsCredentialsProvider provider) {
        S3ClientBuilder s3ClientBuilder = S3Client.builder()
                .region(Region.of(isNotNullOrBlank(region) ? region :
Region.US_EAST_1.id()))
                .forcePathStyle(forcePathStyle)
                .credentialsProvider(provider);
        if (!isNullOrBlank(endpointUrl)) {
            try {
                s3ClientBuilder.endpointOverride(new URI(endpointUrl));
            } catch (URISyntaxException e) {
                throw new IllegalArgumentException("Invalid URL: " +
endpointUrl, e);
            }
        }
        return s3ClientBuilder.build();
    }
    public static abstract class Builder<T extends Builder<T>> {
        private String bucket;
        private String region;
        private String endpointUrl;
        private String profile;
        private boolean forcePathStyle;
        private AwsCredentials awsCredentials;
        /**
         * Set the AWS bucket.
         *
         * @return The builder instance.
         */
        public T bucket(String bucket) {
            this.bucket = bucket;
            return self();
        }
        /**
         * Set the AWS region. Defaults to US_EAST_1
         *
         * @param region The AWS region.
         * @return The builder instance.
         */
        public T region(String region) {
            this.region = region;
            return self();
        }
        /**
         * Specifies a custom endpoint URL to override the default service
URL.
```

```
         *
         * @param endpointUrl The endpoint URL.
         * @return The builder instance.
         */
        public T endpointUrl(String endpointUrl) {
            this.endpointUrl = endpointUrl;
            return self();
        }
        /**
         * Set the profile defined in AWS credentials. If not set, it will
use the default profile.
         *
         * @param profile The profile defined in AWS credentials.
         * @return The builder instance.
         */
        public T profile(String profile) {
            this.profile = profile;
            return self();
        }
        /**
         * Set the forcePathStyle. When enabled, it will use the path-style
URL
         *
         * @param forcePathStyle The forcePathStyle.
         * @return The builder instance.
         */
        public T forcePathStyle(boolean forcePathStyle) {
            this.forcePathStyle = forcePathStyle;
            return self();
        }
        /**
         * Set the AWS credentials. If not set, it will use the default
credentials.
         *
         * @param awsCredentials The AWS credentials.
         * @return The builder instance.
         */
        public T awsCredentials(AwsCredentials awsCredentials) {
            this.awsCredentials = awsCredentials;
            return self();
        }
        public abstract AbstractS3Loader build();
        protected abstract T self();
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\AwsCredentials.java

```
package dev.langchain4j.data.document;
import static dev.langchain4j.internal.Utils.areNotNullOrBlank;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
/**
 * Represents an AWS credentials object, including access key ID, secret
access key, and optional session token.
 */
public class AwsCredentials {
    private final String accessKeyId;
    private final String secretAccessKey;
    private String sessionToken;
    public AwsCredentials(String accessKeyId, String secretAccessKey, String
sessionToken) {
        this.accessKeyId = ensureNotBlank(accessKeyId, "accessKeyId");
        this.secretAccessKey = ensureNotBlank(secretAccessKey,
"secretAccessKey");
        this.sessionToken = sessionToken;
    }
    public AwsCredentials(String accessKeyId, String secretAccessKey) {
        this(accessKeyId, secretAccessKey, null);
    }
    public String accessKeyId() {
        return accessKeyId;
    }
    public String secretAccessKey() {
        return secretAccessKey;
    }
    public String sessionToken() {
        return sessionToken;
    }
    public boolean hasAccessKeyIdAndSecretKey() {
        return areNotNullOrBlank(accessKeyId, secretAccessKey);
    }
    public boolean hasAllCredentials() {
        return areNotNullOrBlank(accessKeyId, secretAccessKey, sessionToken);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\DocumentLoaderUtils.java

```
package dev.langchain4j.data.document;
import dev.langchain4j.data.document.parser.MsOfficeDocumentParser;
import dev.langchain4j.data.document.parser.PdfDocumentParser;
import dev.langchain4j.data.document.parser.TextDocumentParser;
import java.io.InputStream;
class DocumentLoaderUtils {
    static Document load(DocumentSource source, DocumentParser parser) {
        try (InputStream inputStream = source.inputStream()) {
            Document document = parser.parse(inputStream);
            source.metadata().asMap().forEach((key, value) ->
document.metadata().add(key, value));
            return document;
        } catch (Exception e) {
            throw new RuntimeException("Failed to load document", e);
        }
    }
    static DocumentParser parserFor(DocumentType type) {
        switch (type) {
            case TXT:
            case HTML:
            case UNKNOWN:
                return new TextDocumentParser(type);
            case PDF:
                return new PdfDocumentParser();
            case DOC:
            case XLS:
            case PPT:
                return new MsOfficeDocumentParser(type);
            default:
                throw new RuntimeException(String.format("Cannot find parser
for document type '%s'", type));
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\DocumentType.java

```
package dev.langchain4j.data.document;
import static java.util.Arrays.asList;
public enum DocumentType {
    TXT(".txt"),
    PDF(".pdf"),
    HTML(".html", ".htm", ".xhtml"),
    DOC(".doc", ".docx"),
    XLS(".xls", ".xlsx"),
    PPT(".ppt", ".pptx"),
    UNKNOWN;
    private final Iterable<String> supportedExtensions;
    DocumentType(String... supportedExtensions) {
        this.supportedExtensions = asList(supportedExtensions);
    }
    public static DocumentType of(String fileName) {
        for (DocumentType documentType : values()) {
            for (String supportedExtension :
documentType.supportedExtensions) {
                if (fileName.toLowerCase().endsWith(supportedExtension)) {
                    return documentType;
                }
            }
        }
        return UNKNOWN;
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\FileSystemDocumentLoader.java

```
package dev.langchain4j.data.document;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Stream;
import static dev.langchain4j.data.document.DocumentLoaderUtils.parserFor;
import static dev.langchain4j.data.document.source.FileSystemSource.from;
import static dev.langchain4j.internal.Exceptions.illegalArgument;
import static java.nio.file.Files.isDirectory;
import static java.nio.file.Files.isRegularFile;
public class FileSystemDocumentLoader {
    private static final Logger log =
LoggerFactory.getLogger(FileSystemDocumentLoader.class);
    /**
     * Loads a document from the specified file, detecting document type
automatically.
     * See {@link DocumentType} for the list of supported document types.
     * If the document type is UNKNOWN, it is treated as TXT.
     *
     * @param filePath path to the file
     * @return document
     * @throws IllegalArgumentException if specified path is not a file
     */
    public static Document loadDocument(Path filePath) {
        return loadDocument(filePath, DocumentType.of(filePath.toString()));
    }
    /**
     * Loads a document from the specified file, detecting document type
automatically.
     * See {@link DocumentType} for the list of supported document types.
     * If the document type is UNKNOWN, it is treated as TXT.
     *
     * @param filePath path to the file
     * @return document
     * @throws IllegalArgumentException if specified path is not a file
     */
    public static Document loadDocument(String filePath) {
        return loadDocument(Paths.get(filePath));
    }
    /**
     * Loads a document from the specified file.
     *
     * @param filePath     path to the file
     * @param documentType type of the document
     * @return document
     * @throws IllegalArgumentException if specified path is not a file
     */
    public static Document loadDocument(Path filePath, DocumentType
documentType) {
        if (!isRegularFile(filePath)) {
            throw illegalArgument("%s is not a file", filePath);
        }
```

```
        return DocumentLoaderUtils.load(from(filePath),
parserFor(documentType));
    }
    /**
     * Loads a document from the specified file.
     *
     * @param filePath     path to the file
     * @param documentType type of the document
     * @return document
     * @throws IllegalArgumentException if specified path is not a file
     */
    public static Document loadDocument(String filePath, DocumentType
documentType) {
        return loadDocument(Paths.get(filePath), documentType);
    }
    /**
     * Loads documents from the specified directory. Does not use recursion.
     * Detects document types automatically.
     * See {@link DocumentType} for the list of supported document types.
     * If the document type is UNKNOWN, it is treated as TXT.
     *
     * @param directoryPath path to the directory with files
     * @return list of documents
     * @throws IllegalArgumentException if specified path is not a directory
     */
    public static List<Document> loadDocuments(Path directoryPath) {
        if (!isDirectory(directoryPath)) {
            throw illegalArgument("%s is not a directory", directoryPath);
        }
        List<Document> documents = new ArrayList<>();
        try (Stream<Path> paths = Files.list(directoryPath)) {
            paths.filter(Files::isRegularFile)
                    .forEach(filePath -> {
                        try {
                            Document document = loadDocument(filePath);
                            documents.add(document);
                        } catch (Exception e) {
                            log.warn("Failed to load document from " +
filePath, e);
                        }
                    });
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        return documents;
    }
    /**
     * Loads documents from the specified directory. Does not use recursion.
     * Detects document types automatically.
     * See {@link DocumentType} for the list of supported document types.
     * If the document type is UNKNOWN, it is treated as TXT.
     *
     * @param directoryPath path to the directory with files
     * @return list of documents
     * @throws IllegalArgumentException if specified path is not a directory
     */
    public static List<Document> loadDocuments(String directoryPath) {
        return loadDocuments(Paths.get(directoryPath));
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\parser\MsOfficeDocumentParser.java

```
package dev.langchain4j.data.document.parser;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentParser;
import dev.langchain4j.data.document.DocumentType;
import dev.langchain4j.data.document.Metadata;
import org.apache.poi.extractor.ExtractorFactory;
import org.apache.poi.extractor.POITextExtractor;
import java.io.IOException;
import java.io.InputStream;
import static dev.langchain4j.data.document.Document.DOCUMENT_TYPE;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Extracts text from a Microsoft Office document.
 * This parser supports various file formats, including ppt, pptx, doc, docx, xls, and xlsx.
 * For detailed information on supported formats, please refer to the <a href="https://poi.apache.org/">official Apache POI website</a>.
 */
public class MsOfficeDocumentParser implements DocumentParser {
    private final DocumentType documentType;
    public MsOfficeDocumentParser(DocumentType documentType) {
        this.documentType = ensureNotNull(documentType, "documentType");
    }
    @Override
    public Document parse(InputStream inputStream) {
        try (POITextExtractor extractor = ExtractorFactory.createExtractor(inputStream)) {
            return new Document(extractor.getText(), Metadata.from(DOCUMENT_TYPE, documentType));
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\parser\PdfDocumentParser.java

```
package dev.langchain4j.data.document.parser;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentParser;
import dev.langchain4j.data.document.Metadata;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.text.PDFTextStripper;
import java.io.IOException;
import java.io.InputStream;
import static dev.langchain4j.data.document.Document.DOCUMENT_TYPE;
import static dev.langchain4j.data.document.DocumentType.PDF;
public class PdfDocumentParser implements DocumentParser {
    @Override
    public Document parse(InputStream inputStream) {
        try {
            PDDocument pdfDocument = PDDocument.load(inputStream);
            PDFTextStripper stripper = new PDFTextStripper();
            String content = stripper.getText(pdfDocument);
            pdfDocument.close();
            return Document.from(content, Metadata.from(DOCUMENT_TYPE, PDF));
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\parser\TextDocumentParser.java

```
package dev.langchain4j.data.document.parser;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentParser;
import dev.langchain4j.data.document.DocumentType;
import dev.langchain4j.data.document.Metadata;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.nio.charset.Charset;
import static dev.langchain4j.data.document.Document.DOCUMENT_TYPE;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
import static java.nio.charset.StandardCharsets.UTF_8;
public class TextDocumentParser implements DocumentParser {
    private final DocumentType documentType;
    private final Charset charset;
    public TextDocumentParser(DocumentType documentType) {
        this(documentType, UTF_8);
    }
    public TextDocumentParser(DocumentType documentType, Charset charset) {
        this.documentType = ensureNotNull(documentType, "documentType");
        this.charset = ensureNotNull(charset, "charset");
    }
    @Override
    public Document parse(InputStream inputStream) {
        try {
            ByteArrayOutputStream buffer = new ByteArrayOutputStream();
            int nRead;
            byte[] data = new byte[1024];
            while ((nRead = inputStream.read(data, 0, data.length)) != -1) {
                buffer.write(data, 0, nRead);
            }
            buffer.flush();
            String text = new String(buffer.toByteArray(), charset);
            return Document.from(text, Metadata.from(DOCUMENT_TYPE,
documentType.toString()));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\S3DirectoryLoader.java

```
package dev.langchain4j.data.document;
import dev.langchain4j.data.document.source.S3Source;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.*;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import static dev.langchain4j.data.document.DocumentLoaderUtils.parserFor;
/**
 * S3 Directory Loader Implementation
 */
public class S3DirectoryLoader extends AbstractS3Loader<List<Document>> {
    private static final Logger log =
LoggerFactory.getLogger(S3DirectoryLoader.class);
    private final String prefix;
    private S3DirectoryLoader(Builder builder) {
        super(builder);
        this.prefix = builder.prefix;
    }
    /**
     * Loads a list of documents from an S3 bucket, ignoring unsupported
document types.
     * If a prefix is specified, only objects with that prefix will be loaded.
     *
     * @param s3Client The S3 client used for the operation
     * @return A list of Document objects containing the content and metadata
of the S3 objects
     * @throws RuntimeException if an S3 exception occurs during the operation
     */
    @Override
    protected List<Document> load(S3Client s3Client) {
        List<Document> documents = new ArrayList<>();
        ListObjectsV2Request listObjectsV2Request =
ListObjectsV2Request.builder()
                .bucket(bucket)
                .prefix(prefix)
                .build();
        ListObjectsV2Response listObjectsV2Response =
s3Client.listObjectsV2(listObjectsV2Request);
        List<S3Object> filteredS3Objects =
listObjectsV2Response.contents().stream()
                .filter(s3Object -> !s3Object.key().endsWith("/") &&
s3Object.size() > 0)
                .collect(Collectors.toList());
        for (S3Object s3Object : filteredS3Objects) {
            String key = s3Object.key();
            GetObjectRequest getObjectRequest = GetObjectRequest.builder()
                    .bucket(bucket)
                    .key(key)
                    .build();
            ResponseInputStream<GetObjectResponse> inputStream =
s3Client.getObject(getObjectRequest);
            try {
                documents.add(DocumentLoaderUtils.load(new S3Source(bucket,
```

```
key, inputStream), parserFor(DocumentType.of(key))));
            } catch (Exception e) {
                log.warn("Failed to load document from S3", e);
            }
        }
        return documents;
    }
    public static Builder builder() {
        return new Builder();
    }
    public static final class Builder extends
AbstractS3Loader.Builder<Builder> {
        private String prefix = "";
        /**
         * Set the prefix.
         *
         * @param prefix Prefix.
         */
        public Builder prefix(String prefix) {
            this.prefix = prefix;
            return this;
        }
        @Override
        public S3DirectoryLoader build() {
            return new S3DirectoryLoader(this);
        }
        @Override
        protected Builder self() {
            return this;
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\S3FileLoader.java

```
package dev.langchain4j.data.document;
import dev.langchain4j.data.document.source.S3Source;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import static dev.langchain4j.data.document.DocumentLoaderUtils.parserFor;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
/**
 * S3 File Loader Implementation
 */
public class S3FileLoader extends AbstractS3Loader<Document> {
    private final String key;
    private S3FileLoader(Builder builder) {
        super(builder);
        this.key = ensureNotBlank(builder.key, "key");
    }
    /**
     * Loads a document from an S3 bucket based on the specified key.
     *
     * @param s3Client The S3 client used for the operation
     * @return A Document object containing the content and metadata of the
S3 object
     * @throws RuntimeException if an S3 exception occurs during the operation
     */
    @Override
    protected Document load(S3Client s3Client) {
        try {
            GetObjectRequest objectRequest =
GetObjectRequest.builder().bucket(bucket).key(key).build();
            ResponseInputStream<GetObjectResponse> inputStream =
s3Client.getObject(objectRequest);
            return DocumentLoaderUtils.load(new S3Source(bucket, key,
inputStream), parserFor(DocumentType.of(key)));
        } catch (S3Exception e) {
            throw new RuntimeException("Failed to load document from s3", e);
        }
    }
    public static Builder builder() {
        return new Builder();
    }
    public static final class Builder extends
AbstractS3Loader.Builder<Builder> {
        private String key;
        /**
         * Set the object key.
         *
         * @param key Key.
         */
        public Builder key(String key) {
            this.key = key;
            return this;
        }
        @Override
        public S3FileLoader build() {
            return new S3FileLoader(this);
        }
```

```
        @Override
        protected Builder self() {
            return this;
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\source\FileSystemSource.java

```
package dev.langchain4j.data.document.source;
import dev.langchain4j.data.document.DocumentSource;
import dev.langchain4j.data.document.Metadata;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.net.URI;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import static dev.langchain4j.data.document.Document.ABSOLUTE_DIRECTORY_PATH;
import static dev.langchain4j.data.document.Document.FILE_NAME;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
public class FileSystemSource implements DocumentSource {
    public final Path path;
    public FileSystemSource(Path path) {
        this.path = ensureNotNull(path, "path");
    }
    @Override
    public InputStream inputStream() throws IOException {
        return Files.newInputStream(path);
    }
    @Override
    public Metadata metadata() {
        return new Metadata()
                .add(FILE_NAME, path.getFileName())
                .add(ABSOLUTE_DIRECTORY_PATH,
path.getParent().toAbsolutePath());
    }
    public static FileSystemSource from(Path filePath) {
        return new FileSystemSource(filePath);
    }
    public static FileSystemSource from(String filePath) {
        return new FileSystemSource(Paths.get(filePath));
    }
    public static FileSystemSource from(URI fileUri) {
        return new FileSystemSource(Paths.get(fileUri));
    }
    public static FileSystemSource from(File file) {
        return new FileSystemSource(file.toPath());
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\source\S3Source.java

```
package dev.langchain4j.data.document.source;
import dev.langchain4j.data.document.DocumentSource;
import dev.langchain4j.data.document.Metadata;
import java.io.IOException;
import java.io.InputStream;
public class S3Source implements DocumentSource {
    private static final String SOURCE = "source";
    private InputStream inputStream;
    private final String bucket;
    private final String key;
    public S3Source(String bucket, String key, InputStream inputStream) {
        this.inputStream = inputStream;
        this.bucket = bucket;
        this.key = key;
    }
    @Override
    public InputStream inputStream() throws IOException {
        return inputStream;
    }
    @Override
    public Metadata metadata() {
        return new Metadata()
                .add(SOURCE, String.format("s3://%s/%s", bucket, key));
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\source\UrlSource.java

```
package dev.langchain4j.data.document.source;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSource;
import dev.langchain4j.data.document.Metadata;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URI;
import java.net.URL;
import java.net.URLConnection;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
public class UrlSource implements DocumentSource {
    private final URL url;
    public UrlSource(URL url) {
        this.url = ensureNotNull(url, "url");
    }
    @Override
    public InputStream inputStream() throws IOException {
        URLConnection connection = url.openConnection();
        return connection.getInputStream();
    }
    @Override
    public Metadata metadata() {
        return Metadata.from(Document.URL, url);
    }
    public static UrlSource from(String url) {
        try {
            return new UrlSource(new URL(url));
        } catch (MalformedURLException e) {
            throw new RuntimeException(e);
        }
    }
    public static UrlSource from(URL url) {
        return new UrlSource(url);
    }
    public static UrlSource from(URI uri) {
        try {
            return new UrlSource(uri.toURL());
        } catch (MalformedURLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\DocumentByCharacterSplitter.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
/**
 * Splits the provided {@link Document} into characters and attempts to fit as many characters as possible
 * into a single {@link TextSegment}, adhering to the limit set by {@code maxSegmentSize}.
 * <p>
 * The {@code maxSegmentSize} can be defined in terms of characters (default) or tokens.
 * For token-based limit, a {@link Tokenizer} must be provided.
 * <p>
 * If multiple characters fit within {@code maxSegmentSize}, they are joined together without delimiters.
 * <p>
 * Each {@link TextSegment} inherits all metadata from the {@link Document} and includes an "index" metadata key
 * representing its position within the document (starting from 0).
 */
public class DocumentByCharacterSplitter extends HierarchicalDocumentSplitter
{
    public DocumentByCharacterSplitter(int maxSegmentSizeInChars,
                                       int maxOverlapSizeInChars) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, null);
    }
    public DocumentByCharacterSplitter(int maxSegmentSizeInChars,
                                       int maxOverlapSizeInChars,
                                       DocumentSplitter subSplitter) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, subSplitter);
    }
    public DocumentByCharacterSplitter(int maxSegmentSizeInTokens,
                                       int maxOverlapSizeInTokens,
                                       Tokenizer tokenizer) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, null);
    }
    public DocumentByCharacterSplitter(int maxSegmentSizeInTokens,
                                       int maxOverlapSizeInTokens,
                                       Tokenizer tokenizer,
                                       DocumentSplitter subSplitter) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, subSplitter);
    }
    @Override
    public String[] split(String text) {
        return text.split("");
    }
    @Override
    public String joinDelimiter() {
        return "";
    }
    @Override
    protected DocumentSplitter defaultSubSplitter() {
```

```
            return null;
        }
    }
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\DocumentByLineSplitter.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
/**
 * Splits the provided {@link Document} into lines and attempts to fit as many lines as possible
 * into a single {@link TextSegment}, adhering to the limit set by {@code maxSegmentSize}.
 * <p>
 * The {@code maxSegmentSize} can be defined in terms of characters (default) or tokens.
 * For token-based limit, a {@link Tokenizer} must be provided.
 * <p>
 * Line boundaries are detected by a minimum of one newline character ("\n").
 * Any additional whitespaces before or after are ignored.
 * So, the following examples are all valid line separators: "\n", "\n\n", " \n", "\n " and so on.
 * <p>
 * If multiple lines fit within {@code maxSegmentSize}, they are joined together using a newline ("\n").
 * <p>
 * If a single line is too long and exceeds {@code maxSegmentSize},
 * the {@code subSplitter} ({@link DocumentBySentenceSplitter} by default) is used to split it into smaller parts and
 * place them into multiple segments.
 * Such segments contain only the parts of the split long line.
 * <p>
 * Each {@link TextSegment} inherits all metadata from the {@link Document} and includes an "index" metadata key
 * representing its position within the document (starting from 0).
 */
public class DocumentByLineSplitter extends HierarchicalDocumentSplitter {
    public DocumentByLineSplitter(int maxSegmentSizeInChars,
                                  int maxOverlapSizeInChars) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, null);
    }
    public DocumentByLineSplitter(int maxSegmentSizeInChars,
                                  int maxOverlapSizeInChars,
                                  DocumentSplitter subSplitter) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, subSplitter);
    }
    public DocumentByLineSplitter(int maxSegmentSizeInTokens,
                                  int maxOverlapSizeInTokens,
                                  Tokenizer tokenizer) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, null);
    }
    public DocumentByLineSplitter(int maxSegmentSizeInTokens,
                                  int maxOverlapSizeInTokens,
                                  Tokenizer tokenizer,
                                  DocumentSplitter subSplitter) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, subSplitter);
    }
```

```
    @Override
    public String[] split(String text) {
        return text.split("\\s*\\R\\s*"); // additional whitespaces are
ignored
    }
    @Override
    public String joinDelimiter() {
        return "\n";
    }
    @Override
    protected DocumentSplitter defaultSubSplitter() {
        return new DocumentBySentenceSplitter(maxSegmentSize, maxOverlapSize,
tokenizer);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\DocumentByParagraphSplitter.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
/**
 * Splits the provided {@link Document} into paragraphs and attempts to fit as many paragraphs as possible
 * into a single {@link TextSegment}, adhering to the limit set by {@code maxSegmentSize}.
 * <p>
 * The {@code maxSegmentSize} can be defined in terms of characters (default) or tokens.
 * For token-based limit, a {@link Tokenizer} must be provided.
 * <p>
 * Paragraph boundaries are detected by a minimum of two newline characters ("\n\n").
 * Any additional whitespaces before, between, or after are ignored.
 * So, the following examples are all valid paragraph separators: "\n\n", "\n\n\n", "\n \n", " \n \n ", and so on.
 * <p>
 * If multiple paragraphs fit within {@code maxSegmentSize}, they are joined together using a double newline ("\n\n").
 * <p>
 * If a single paragraph is too long and exceeds {@code maxSegmentSize},
 * the {@code subSplitter} ({@link DocumentBySentenceSplitter} by default) is used to split it into smaller parts and
 * place them into multiple segments.
 * Such segments contain only the parts of the split long paragraph.
 * <p>
 * Each {@link TextSegment} inherits all metadata from the {@link Document} and includes an "index" metadata key
 * representing its position within the document (starting from 0).
 */
public class DocumentByParagraphSplitter extends HierarchicalDocumentSplitter {
    public DocumentByParagraphSplitter(int maxSegmentSizeInChars,
                                       int maxOverlapSizeInChars) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, null);
    }
    public DocumentByParagraphSplitter(int maxSegmentSizeInChars,
                                       int maxOverlapSizeInChars,
                                       DocumentSplitter subSplitter) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, subSplitter);
    }
    public DocumentByParagraphSplitter(int maxSegmentSizeInTokens,
                                       int maxOverlapSizeInTokens,
                                       Tokenizer tokenizer) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, null);
    }
    public DocumentByParagraphSplitter(int maxSegmentSizeInTokens,
                                       int maxOverlapSizeInTokens,
                                       Tokenizer tokenizer,
                                       DocumentSplitter subSplitter) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer,
```

```
subSplitter);
    }
    @Override
    public String[] split(String text) {
        return text.split("\\s*\\R\\s*\\R\\s*"); // additional whitespaces
are ignored
    }
    @Override
    public String joinDelimiter() {
        return "\n\n";
    }
    @Override
    protected DocumentSplitter defaultSubSplitter() {
        return new DocumentBySentenceSplitter(maxSegmentSize, maxOverlapSize,
tokenizer);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\DocumentByRegexSplitter.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Splits the provided {@link Document} into parts using the provided {@code regex} and attempts to fit as many parts
 * as possible into a single {@link TextSegment}, adhering to the limit set by {@code maxSegmentSize}.
 * <p>
 * The {@code maxSegmentSize} can be defined in terms of characters (default) or tokens.
 * For token-based limit, a {@link Tokenizer} must be provided.
 * <p>
 * If multiple parts fit within {@code maxSegmentSize}, they are joined together using the provided {@code joinDelimiter}.
 * <p>
 * If a single part is too long and exceeds {@code maxSegmentSize}, the {@code subSplitter} (which should be provided)
 * is used to split it into sub-parts and place them into multiple segments.
 * Such segments contain only the sub-parts of the split long part.
 * <p>
 * Each {@link TextSegment} inherits all metadata from the {@link Document} and includes an "index" metadata key
 * representing its position within the document (starting from 0).
 */
public class DocumentByRegexSplitter extends HierarchicalDocumentSplitter {
    private final String regex;
    private final String joinDelimiter;
    public DocumentByRegexSplitter(String regex,
                                   String joinDelimiter,
                                   int maxSegmentSizeInChars,
                                   int maxOverlapSizeInChars) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, null);
        this.regex = ensureNotNull(regex, "regex");
        this.joinDelimiter = ensureNotNull(joinDelimiter, "joinDelimiter");
    }
    public DocumentByRegexSplitter(String regex,
                                   String joinDelimiter,
                                   int maxSegmentSizeInChars,
                                   int maxOverlapSizeInChars,
                                   DocumentSplitter subSplitter) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, subSplitter);
        this.regex = ensureNotNull(regex, "regex");
        this.joinDelimiter = ensureNotNull(joinDelimiter, "joinDelimiter");
    }
    public DocumentByRegexSplitter(String regex,
                                   String joinDelimiter,
                                   int maxSegmentSizeInTokens,
                                   int maxOverlapSizeInTokens,
                                   Tokenizer tokenizer) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, null);
        this.regex = ensureNotNull(regex, "regex");
```

```
        this.joinDelimiter = ensureNotNull(joinDelimiter, "joinDelimiter");
    }
    public DocumentByRegexSplitter(String regex,
                                   String joinDelimiter,
                                   int maxSegmentSizeInTokens,
                                   int maxOverlapSizeInTokens,
                                   Tokenizer tokenizer,
                                   DocumentSplitter subSplitter) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer,
subSplitter);
        this.regex = ensureNotNull(regex, "regex");
        this.joinDelimiter = ensureNotNull(joinDelimiter, "joinDelimiter");
    }
    @Override
    public String[] split(String text) {
        return text.split(regex);
    }
    @Override
    public String joinDelimiter() {
        return joinDelimiter;
    }
    @Override
    protected DocumentSplitter defaultSubSplitter() {
        return null;
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\DocumentBySentenceSplitter.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;
import java.io.InputStream;
/**
 * Splits the provided {@link Document} into sentences and attempts to fit as many sentences as possible
 * into a single {@link TextSegment}, adhering to the limit set by {@code maxSegmentSize}.
 * <p>
 * The {@code maxSegmentSize} can be defined in terms of characters (default) or tokens.
 * For token-based limit, a {@link Tokenizer} must be provided.
 * <p>
 * Sentence boundaries are detected using the Apache OpenNLP library with the English sentence model.
 * <p>
 * If multiple sentences fit within {@code maxSegmentSize}, they are joined together using a space (" ").
 * <p>
 * If a single sentence is too long and exceeds {@code maxSegmentSize},
 * the {@code subSplitter} ({@link DocumentByWordSplitter} by default) is used to split it into smaller parts and
 * place them into multiple segments.
 * Such segments contain only the parts of the split long sentence.
 * <p>
 * Each {@link TextSegment} inherits all metadata from the {@link Document} and includes an "index" metadata key
 * representing its position within the document (starting from 0).
 */
public class DocumentBySentenceSplitter extends HierarchicalDocumentSplitter {
    private final SentenceDetectorME sentenceDetector;
    public DocumentBySentenceSplitter(int maxSegmentSizeInChars,
                                      int maxOverlapSizeInChars) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, null);
        this.sentenceDetector = createSentenceDetector();
    }
    public DocumentBySentenceSplitter(int maxSegmentSizeInChars,
                                      int maxOverlapSizeInChars,
                                      DocumentSplitter subSplitter) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, subSplitter);
        this.sentenceDetector = createSentenceDetector();
    }
    public DocumentBySentenceSplitter(int maxSegmentSizeInTokens,
                                      int maxOverlapSizeInTokens,
                                      Tokenizer tokenizer) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, null);
        this.sentenceDetector = createSentenceDetector();
    }
    public DocumentBySentenceSplitter(int maxSegmentSizeInTokens,
                                      int maxOverlapSizeInTokens,
```

```
                                        Tokenizer tokenizer,
                                        DocumentSplitter subSplitter) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer,
subSplitter);
        this.sentenceDetector = createSentenceDetector();
    }
    private SentenceDetectorME createSentenceDetector() {
        String sentenceModelFilePath = "/opennlp/opennlp-en-ud-ewt-
sentence-1.0-1.9.3.bin";
        try (InputStream is =
getClass().getResourceAsStream(sentenceModelFilePath)) {
            return new SentenceDetectorME(new SentenceModel(is));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
    @Override
    public String[] split(String text) {
        return sentenceDetector.sentDetect(text);
    }
    @Override
    public String joinDelimiter() {
        return " ";
    }
    @Override
    protected DocumentSplitter defaultSubSplitter() {
        return new DocumentByWordSplitter(maxSegmentSize, maxOverlapSize,
tokenizer);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\DocumentByWordSplitter.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
/**
 * Splits the provided {@link Document} into words and attempts to fit as many words as possible
 * into a single {@link TextSegment}, adhering to the limit set by {@code maxSegmentSize}.
 * <p>
 * The {@code maxSegmentSize} can be defined in terms of characters (default) or tokens.
 * For token-based limit, a {@link Tokenizer} must be provided.
 * <p>
 * Word boundaries are detected by a minimum of one space (" ").
 * Any additional whitespaces before or after are ignored.
 * So, the following examples are all valid word separators: " ", "  ", "\n", and so on.
 * <p>
 * If multiple words fit within {@code maxSegmentSize}, they are joined together using a space (" ").
 * <p>
 * Although this should not happen, if a single word is too long and exceeds {@code maxSegmentSize},
 * the {@code subSplitter} ({@link DocumentByCharacterSplitter} by default) is used to split it into smaller parts and
 * place them into multiple segments.
 * Such segments contain only the parts of the split long word.
 * <p>
 * Each {@link TextSegment} inherits all metadata from the {@link Document} and includes an "index" metadata key
 * representing its position within the document (starting from 0).
 */
public class DocumentByWordSplitter extends HierarchicalDocumentSplitter {
    public DocumentByWordSplitter(int maxSegmentSizeInChars,
                                  int maxOverlapSizeInChars) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, null);
    }
    public DocumentByWordSplitter(int maxSegmentSizeInChars,
                                  int maxOverlapSizeInChars,
                                  DocumentSplitter subSplitter) {
        super(maxSegmentSizeInChars, maxOverlapSizeInChars, null, subSplitter);
    }
    public DocumentByWordSplitter(int maxSegmentSizeInTokens,
                                  int maxOverlapSizeInTokens,
                                  Tokenizer tokenizer) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, null);
    }
    public DocumentByWordSplitter(int maxSegmentSizeInTokens,
                                  int maxOverlapSizeInTokens,
                                  Tokenizer tokenizer,
                                  DocumentSplitter subSplitter) {
        super(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, subSplitter);
```

```
    }
    @Override
    public String[] split(String text) {
        return text.split("\\s+"); // additional whitespaces are ignored
    }
    @Override
    public String joinDelimiter() {
        return " ";
    }
    @Override
    protected DocumentSplitter defaultSubSplitter() {
        return new DocumentByCharacterSplitter(maxSegmentSize,
maxOverlapSize, tokenizer);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\DocumentSplitters.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.model.Tokenizer;
public class DocumentSplitters {
    /**
     * This is a recommended {@link DocumentSplitter} for generic text.
     * It tries to split the document into paragraphs first and fits
     * as many paragraphs into a single {@link
dev.langchain4j.data.segment.TextSegment} as possible.
     * If some paragraphs are too long, they are recursively split into
lines, then sentences,
     * then words, and then characters until they fit into a segment.
     *
     * @param maxSegmentSizeInTokens The maximum size of the segment, defined
in tokens.
     * @param maxOverlapSizeInTokens The maximum size of the overlap, defined
in tokens.
     *                               Only full sentences are considered for
the overlap.
     * @param tokenizer              The tokenizer that is used to count
tokens in the text.
     * @return recursive document splitter
     */
    public static DocumentSplitter recursive(int maxSegmentSizeInTokens,
                                             int maxOverlapSizeInTokens,
                                             Tokenizer tokenizer) {
        return new DocumentByParagraphSplitter(maxSegmentSizeInTokens,
maxOverlapSizeInTokens, tokenizer,
                new DocumentByLineSplitter(maxSegmentSizeInTokens,
maxOverlapSizeInTokens, tokenizer,
                        new
DocumentBySentenceSplitter(maxSegmentSizeInTokens, maxOverlapSizeInTokens,
tokenizer,
                                new
DocumentByWordSplitter(maxSegmentSizeInTokens, maxOverlapSizeInTokens,
tokenizer)
                        )
                )
        );
    }
    /**
     * This is a recommended {@link DocumentSplitter} for generic text.
     * It tries to split the document into paragraphs first and fits
     * as many paragraphs into a single {@link
dev.langchain4j.data.segment.TextSegment} as possible.
     * If some paragraphs are too long, they are recursively split into
lines, then sentences,
     * then words, and then characters until they fit into a segment.
     *
     * @param maxSegmentSizeInChars  The maximum size of the segment, defined
in characters.
     * @param maxOverlapSizeInTokens The maximum size of the overlap, defined
in characters.
     *                               Only full sentences are considered for
the overlap.
     * @return recursive document splitter
     */
```

```
    public static DocumentSplitter recursive(int maxSegmentSizeInChars, int
maxOverlapSizeInTokens) {
        return recursive(maxSegmentSizeInChars, maxOverlapSizeInTokens, null);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\HierarchicalDocumentSplitter.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.atomic.AtomicInteger;
import static dev.langchain4j.internal.Utils.firstChars;
import static dev.langchain4j.internal.ValidationUtils.*;
import static java.lang.String.format;
public abstract class HierarchicalDocumentSplitter implements
DocumentSplitter {
    private static final String INDEX = "index";
    protected final int maxSegmentSize;
    protected final int maxOverlapSize;
    protected final Tokenizer tokenizer;
    protected final DocumentSplitter subSplitter;
    protected HierarchicalDocumentSplitter(int maxSegmentSizeInChars, int
maxOverlapSizeInChars) {
        this(maxSegmentSizeInChars, maxOverlapSizeInChars, null, null);
    }
    protected HierarchicalDocumentSplitter(int maxSegmentSizeInChars,
                                           int maxOverlapSizeInChars,
                                           HierarchicalDocumentSplitter
subSplitter) {
        this(maxSegmentSizeInChars, maxOverlapSizeInChars, null, subSplitter);
    }
    protected HierarchicalDocumentSplitter(int maxSegmentSizeInTokens,
                                           int maxOverlapSizeInTokens,
                                           Tokenizer tokenizer) {
        this(maxSegmentSizeInTokens, maxOverlapSizeInTokens, tokenizer, null);
    }
    protected HierarchicalDocumentSplitter(int maxSegmentSizeInTokens,
                                           int maxOverlapSizeInTokens,
                                           Tokenizer tokenizer,
                                           DocumentSplitter subSplitter) {
        this.maxSegmentSize = ensureGreaterThanZero(maxSegmentSizeInTokens,
"maxSegmentSize");
        this.maxOverlapSize = ensureBetween(maxOverlapSizeInTokens, 0,
maxSegmentSize, "maxOverlapSize");
        this.tokenizer = tokenizer;
        this.subSplitter = subSplitter == null ? defaultSubSplitter() :
subSplitter;
    }
    protected abstract String[] split(String text);
    protected abstract String joinDelimiter();
    protected abstract DocumentSplitter defaultSubSplitter();
    @Override
    public List<TextSegment> split(Document document) {
        ensureNotNull(document, "document");
        List<TextSegment> segments = new ArrayList<>();
        SegmentBuilder segmentBuilder = new SegmentBuilder(maxSegmentSize,
this::sizeOf, joinDelimiter());
        AtomicInteger index = new AtomicInteger(0);
        String[] parts = split(document.text());
```

```
        String overlap = null;
        for (String part : parts) {
            if (segmentBuilder.hasSpaceFor(part)) {
                segmentBuilder.append(part);
            } else {
                if (segmentBuilder.isNotEmpty() && !
segmentBuilder.build().equals(overlap)) {
                    String segmentText = segmentBuilder.build();
                    segments.add(createSegment(segmentText, document,
index.getAndIncrement()));
                    segmentBuilder.reset();
                    overlap = overlapFrom(segmentText);
                    segmentBuilder.append(overlap);
                }
                if (segmentBuilder.hasSpaceFor(part)) {
                    segmentBuilder.append(part);
                } else {
                    if (subSplitter == null) {
                        throw new RuntimeException(format(
                                "The text \"%s...\" (%s %s long) doesn't fit
into the maximum segment size (%s %s), " +
                                        "and there is no subSplitter defined
to split it further.",
                                firstChars(part, 30),
                                sizeOf(part), tokenizer == null ?
"characters" : "tokens",
                                maxSegmentSize, tokenizer == null ?
"characters" : "tokens"
                        ));
                    }
                    segmentBuilder.append(part);
                    for (TextSegment segment :
subSplitter.split(Document.from(segmentBuilder.build()))) {
                        segments.add(createSegment(segment.text(), document,
index.getAndIncrement()));
                    }
                    segmentBuilder.reset();
                    TextSegment lastSegment = segments.get(segments.size() -
1);
                    overlap = overlapFrom(lastSegment.text());
                    segmentBuilder.append(overlap);
                }
            }
        }
        if (segmentBuilder.isNotEmpty() && !
segmentBuilder.build().equals(overlap)) {
            segments.add(createSegment(segmentBuilder.build(), document,
index.getAndIncrement()));
        }
        return segments;
    }
    private String overlapFrom(String segmentText) {
        if (maxOverlapSize == 0) {
            return "";
        }
        SegmentBuilder overlapBuilder = new SegmentBuilder(maxOverlapSize,
this::sizeOf, joinDelimiter());
        String[] sentences = new DocumentBySentenceSplitter(1, 0, null,
null).split(segmentText);
        for (int i = sentences.length - 1; i >= 0; i--) {
            String part = sentences[i];
```

```
            if (overlapBuilder.hasSpaceFor(part)) {
                overlapBuilder.prepend(part);
            } else {
                return overlapBuilder.build();
            }
        }
        return "";
    }
    private int sizeOf(String text) {
        if (tokenizer != null) {
            return tokenizer.estimateTokenCountInText(text);
        } else {
            return text.length();
        }
    }
    private static TextSegment createSegment(String text, Document document,
int index) {
        Metadata metadata = document.metadata().copy().add(INDEX, index);
        return TextSegment.from(text, metadata);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\splitter\SegmentBuild
er.java

```
package dev.langchain4j.data.document.splitter;
import java.util.function.Function;
import static dev.langchain4j.internal.ValidationUtils.ensureGreaterThanZero;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
class SegmentBuilder {
    private StringBuilder segmentBuilder;
    private final int maxSegmentSize;
    private final Function<String, Integer> sizeFunction;
    private final String joinSeparator;
    SegmentBuilder(int maxSegmentSize, Function<String, Integer>
sizeFunction, String joinSeparator) {
        this.segmentBuilder = new StringBuilder();
        this.maxSegmentSize = ensureGreaterThanZero(maxSegmentSize,
"maxSegmentSize");
        this.sizeFunction = ensureNotNull(sizeFunction, "sizeFunction");
        this.joinSeparator = ensureNotNull(joinSeparator, "joinSeparator");
    }
    boolean hasSpaceFor(String text) {
        if (isNotEmpty()) {
            return sizeOf(segmentBuilder.toString()) + sizeOf(joinSeparator)
+ sizeOf(text) <= maxSegmentSize;
        } else {
            return sizeOf(text) <= maxSegmentSize;
        }
    }
    private int sizeOf(String text) {
        return sizeFunction.apply(text);
    }
    void append(String text) {
        if (isNotEmpty()) {
            segmentBuilder.append(joinSeparator);
        }
        segmentBuilder.append(text);
    }
    void prepend(String text) {
        if (isNotEmpty()) {
            segmentBuilder.insert(0, text + joinSeparator);
        } else {
            segmentBuilder.insert(0, text);
        }
    }
    boolean isNotEmpty() {
        return segmentBuilder.length() > 0;
    }
    String build() {
        return segmentBuilder.toString().trim();
    }
    void reset() {
        segmentBuilder = new StringBuilder();
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\transformer\HtmlTextExtractor.java

```
package dev.langchain4j.data.document.transformer;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentTransformer;
import dev.langchain4j.data.document.Metadata;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;
import org.jsoup.nodes.Node;
import org.jsoup.nodes.TextNode;
import org.jsoup.select.NodeVisitor;
import java.util.Map;
import static java.lang.String.format;
import static java.util.stream.Collectors.joining;
import static org.jsoup.internal.StringUtil.in;
import static org.jsoup.select.NodeTraversor.traverse;
/**
 * Extracts text from a given HTML document.
 * A CSS selector can be specified to extract text only from desired element(s).
 * Also, multiple CSS selectors can be specified to extract metadata from desired elements.
 */
public class HtmlTextExtractor implements DocumentTransformer {
    private final String cssSelector;
    private final Map<String, String> metadataCssSelectors;
    private final boolean includeLinks;
    /**
     * Constructs an instance of HtmlToTextTransformer that extracts all text from a given Document containing HTML.
     */
    public HtmlTextExtractor() {
        this(null, null, false);
    }
    /**
     * Constructs an instance of HtmlToTextTransformer that extracts text from HTML elements matching the provided CSS selector.
     *
     * @param cssSelector          A CSS selector.
     *                             For example, "#page-content" will extract text from the HTML element with the id "page-content".
     * @param metadataCssSelectors A mapping from metadata keys to CSS selectors.
     *                             For example, Mep.of("title", "#page-title") will extract all text from the HTML element
     *                             with id "title" and store it in {@link Metadata} under the key "title".
     * @param includeLinks         Specifies whether links should be included in the extracted text.
     */
    public HtmlTextExtractor(String cssSelector, Map<String, String> metadataCssSelectors, boolean includeLinks) {
        this.cssSelector = cssSelector;
        this.metadataCssSelectors = metadataCssSelectors;
        this.includeLinks = includeLinks;
    }
    @Override
    public Document transform(Document document) {
        String html = document.text();
```

```
        org.jsoup.nodes.Document jsoupDocument = Jsoup.parse(html);
        String text;
        if (cssSelector != null) {
            text = extractText(jsoupDocument, cssSelector, includeLinks);
        } else {
            text = extractText(jsoupDocument, includeLinks);
        }
        Metadata metadata = document.metadata().copy();
        if (metadataCssSelectors != null) {
            metadataCssSelectors.forEach((metadataKey, cssSelector) ->
                    metadata.add(metadataKey,
jsoupDocument.select(cssSelector).text()));
        }
        return Document.from(text, metadata);
    }
    private static String extractText(org.jsoup.nodes.Document jsoupDocument,
String cssSelector, boolean includeLinks) {
        return jsoupDocument.select(cssSelector).stream()
                .map(element -> extractText(element, includeLinks))
                .collect(joining("\n\n"));
    }
    private static String extractText(Element element, boolean includeLinks) {
        NodeVisitor visitor = new TextExtractingVisitor(includeLinks);
        traverse(visitor, element);
        return visitor.toString().trim();
    }
    // taken from https://github.com/jhy/jsoup/blob/master/src/main/java/org/
jsoup/examples/HtmlToPlainText.java
    private static class TextExtractingVisitor implements NodeVisitor {
        private final StringBuilder textBuilder = new StringBuilder();
        private final boolean includeLinks;
        private TextExtractingVisitor(boolean includeLinks) {
            this.includeLinks = includeLinks;
        }
        @Override
        public void head(Node node, int depth) { // hit when the node is
first seen
            String name = node.nodeName();
            if (node instanceof TextNode)
                textBuilder.append(((TextNode) node).text());
            else if (name.equals("li"))
                textBuilder.append("\n * ");
            else if (name.equals("dt"))
                textBuilder.append("  ");
            else if (in(name, "p", "h1", "h2", "h3", "h4", "h5", "h6", "tr"))
                textBuilder.append("\n");
        }
        @Override
        public void tail(Node node, int depth) { // hit when all the node's
children (if any) have been visited
            String name = node.nodeName();
            if (in(name, "br", "dd", "dt", "p", "h1", "h2", "h3", "h4", "h5",
"h6"))
                textBuilder.append("\n");
            else if (includeLinks && name.equals("a"))
                textBuilder.append(format(" <%s>", node.absUrl("href")));
        }
        @Override
        public String toString() {
            return textBuilder.toString();
        }
```

```
    }
}
```

langchain4j\src\main\java\dev\langchain4j\data\document\UrlDocumentLoader.java

```
package dev.langchain4j.data.document;
import dev.langchain4j.data.document.source.UrlSource;
import java.net.MalformedURLException;
import java.net.URL;
import static dev.langchain4j.data.document.DocumentLoaderUtils.parserFor;
public class UrlDocumentLoader {
    /**
     * Loads a document from the specified URL, detecting the document type
automatically.
     * See {@link DocumentType} for the list of supported document types.
     * If the document type is UNKNOWN, it is treated as TXT.
     *
     * @param url URL of the file
     * @return document
     */
    public static Document load(URL url) {
        return load(url, DocumentType.of(url.toString()));
    }
    /**
     * Loads a document from the specified URL, detecting the document type
automatically.
     * See {@link DocumentType} for the list of supported document types.
     * If the document type is UNKNOWN, it is treated as TXT.
     *
     * @param url URL of the file
     * @return document
     * @throws RuntimeException if specified URL is malformed
     */
    public static Document load(String url) {
        try {
            return load(new URL(url));
        } catch (MalformedURLException e) {
            throw new RuntimeException(e);
        }
    }
    /**
     * Loads a document from the specified URL.
     *
     * @param url          URL of the file
     * @param documentType type of the document
     * @return document
     */
    public static Document load(URL url, DocumentType documentType) {
        return DocumentLoaderUtils.load(UrlSource.from(url),
parserFor(documentType));
    }
    /**
     * Loads a document from the specified URL.
     *
     * @param url          URL of the file
     * @param documentType type of the document
     * @return document
     * @throws RuntimeException if specified URL is malformed
     */
    public static Document load(String url, DocumentType documentType) {
        try {
            return load(new URL(url), documentType);
```

```
        } catch (MalformedURLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\exception\IllegalConfigurationException.java

```
package dev.langchain4j.exception;
import static java.lang.String.format;
public class IllegalConfigurationException extends RuntimeException {
    public IllegalConfigurationException(String message) {
        super(message);
    }
    public static IllegalConfigurationException illegalConfiguration(String message) {
        return new IllegalConfigurationException(message);
    }
    public static IllegalConfigurationException illegalConfiguration(String format, Object... args) {
        return new IllegalConfigurationException(format(format, args));
    }
}
```

langchain4j\src\main\java\dev\langchain4j\memory\chat\ChatMemoryProvider.java

```
package dev.langchain4j.memory.chat;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.service.MemoryId;
/**
 * Provides instances of {@link ChatMemory}.
 * Intended to be used with {@link dev.langchain4j.service.AiServices}.
 */
@FunctionalInterface
public interface ChatMemoryProvider {
    /**
     * Provides an instance of {@link ChatMemory}.
     * This method is called each time an AI Service method (having a
parameter annotated with {@link MemoryId})
     * is called with a previously unseen memory ID.
     * Once the {@link ChatMemory} instance is returned, it's retained in
memory and managed by {@link dev.langchain4j.service.AiServices}.
     *
     * @param memoryId The ID of the chat memory.
     * @return A {@link ChatMemory} instance.
     * @see MemoryId
     */
    ChatMemory get(Object memoryId);
}
```

langchain4j\src\main\java\dev\langchain4j\memory\chat\MessageWindowChatMemory.java

```
package dev.langchain4j.memory.chat;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.store.memory.chat.ChatMemoryStore;
import dev.langchain4j.store.memory.chat.InMemoryChatMemoryStore;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import static dev.langchain4j.internal.ValidationUtils.ensureGreaterThanZero;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * This chat memory operates as a sliding window of {@link #maxMessages} messages.
 * It retains as many of the most recent messages as can fit into the window.
 * If there isn't enough space for a new message, the oldest one is discarded.
 * <p>
 * Once added, a {@link SystemMessage} is always retained.
 * Only one {@link SystemMessage} can be held at a time.
 * If a new {@link SystemMessage} with the same content is added, it is ignored.
 * If a new {@link SystemMessage} with different content is added, it replaces the previous one.
 * <p>
 * The state of chat memory is stored in {@link ChatMemoryStore}.
 */
public class MessageWindowChatMemory implements ChatMemory {
    private static final Logger log = LoggerFactory.getLogger(MessageWindowChatMemory.class);
    private final Object id;
    private final Integer maxMessages;
    private final ChatMemoryStore store;
    private MessageWindowChatMemory(Builder builder) {
        this.id = ensureNotNull(builder.id, "id");
        this.maxMessages = ensureGreaterThanZero(builder.maxMessages, "maxMessages");
        this.store = ensureNotNull(builder.store, "store");
    }
    @Override
    public Object id() {
        return id;
    }
    @Override
    public void add(ChatMessage message) {
        List<ChatMessage> messages = messages();
        if (message instanceof SystemMessage) {
            Optional<SystemMessage> systemMessage = findSystemMessage(messages);
            if (systemMessage.isPresent()) {
                if (systemMessage.get().equals(message)) {
                    return; // do not add the same system message
                } else {
                    messages.remove(systemMessage.get()); // need to replace existing system message
                }
```

```
            }
        }
        messages.add(message);
        ensureCapacity(messages, maxMessages);
        store.updateMessages(id, messages);
    }
    private static Optional<SystemMessage>
findSystemMessage(List<ChatMessage> messages) {
        return messages.stream()
                .filter(message -> message instanceof SystemMessage)
                .map(message -> (SystemMessage) message)
                .findAny();
    }
    @Override
    public List<ChatMessage> messages() {
        List<ChatMessage> messages = new ArrayList<>(store.getMessages(id));
        ensureCapacity(messages, maxMessages);
        return messages;
    }
    private static void ensureCapacity(List<ChatMessage> messages, int
maxMessages) {
        while (messages.size() > maxMessages) {
            int messageToRemove = 0;
            if (messages.get(0) instanceof SystemMessage) {
                messageToRemove = 1;
            }
            ChatMessage removedMessage = messages.remove(messageToRemove);
            log.trace("Removing the following message to comply with the
capacity requirements: {}", removedMessage);
        }
    }
    @Override
    public void clear() {
        store.deleteMessages(id);
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private Object id = "default";
        private Integer maxMessages;
        private ChatMemoryStore store = new InMemoryChatMemoryStore();
        /**
         * @param id The ID of the {@link ChatMemory}.
         *           If not provided, a "default" will be used.
         * @return builder
         */
        public Builder id(Object id) {
            this.id = id;
            return this;
        }
        /**
         * @param maxMessages The maximum number of messages to retain.
         * @return builder
         */
        public Builder maxMessages(Integer maxMessages) {
            this.maxMessages = maxMessages;
            return this;
        }
        /**
         * @param store The chat memory store responsible for storing the
```

```
chat memory state.
         *              If not provided, an {@link InMemoryChatMemoryStore}
will be used.
         * @return builder
         */
        public Builder chatMemoryStore(ChatMemoryStore store) {
            this.store = store;
            return this;
        }
        public MessageWindowChatMemory build() {
            return new MessageWindowChatMemory(this);
        }
    }
    public static MessageWindowChatMemory withMaxMessages(int maxMessages) {
        return builder().maxMessages(maxMessages).build();
    }
}
```

langchain4j\src\main\java\dev\langchain4j\memory\chat\TokenWindowChatMemory.java

```
package dev.langchain4j.memory.chat;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.store.memory.chat.ChatMemoryStore;
import dev.langchain4j.store.memory.chat.InMemoryChatMemoryStore;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import static dev.langchain4j.internal.ValidationUtils.ensureGreaterThanZero;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * This chat memory operates as a sliding window of {@link #maxTokens} tokens.
 * It retains as many of the most recent messages as can fit into the window.
 * If there isn't enough space for a new message, the oldest one (or
multiple) is discarded.
 * Messages are indivisible. If a message doesn't fit, it's discarded
completely.
 * <p>
 * Once added, a {@link SystemMessage} is always retained.
 * Only one {@link SystemMessage} can be held at a time.
 * If a new {@link SystemMessage} with the same content is added, it is
ignored.
 * If a new {@link SystemMessage} with different content is added, it
replaces the previous one.
 * <p>
 * The state of chat memory is stored in {@link ChatMemoryStore}.
 */
public class TokenWindowChatMemory implements ChatMemory {
    private static final Logger log =
LoggerFactory.getLogger(TokenWindowChatMemory.class);
    private final Object id;
    private final Integer maxTokens;
    private final Tokenizer tokenizer;
    private final ChatMemoryStore store;
    private TokenWindowChatMemory(Builder builder) {
        this.id = ensureNotNull(builder.id, "id");
        this.maxTokens = ensureGreaterThanZero(builder.maxTokens,
"maxTokens");
        this.tokenizer = ensureNotNull(builder.tokenizer, "tokenizer");
        this.store = ensureNotNull(builder.store, "store");
    }
    @Override
    public Object id() {
        return id;
    }
    @Override
    public void add(ChatMessage message) {
        List<ChatMessage> messages = messages();
        if (message instanceof SystemMessage) {
            Optional<SystemMessage> maybeSystemMessage =
findSystemMessage(messages);
            if (maybeSystemMessage.isPresent()) {
                if (maybeSystemMessage.get().equals(message)) {
```

```
                    return; // do not add the same system message
                } else {
                    messages.remove(maybeSystemMessage.get()); // need to
replace existing system message
                }
            }
        }
        messages.add(message);
        ensureCapacity(messages, maxTokens, tokenizer);
        store.updateMessages(id, messages);
    }
    private static Optional<SystemMessage>
findSystemMessage(List<ChatMessage> messages) {
        return messages.stream()
                .filter(message -> message instanceof SystemMessage)
                .map(message -> (SystemMessage) message)
                .findAny();
    }
    @Override
    public List<ChatMessage> messages() {
        List<ChatMessage> messages = new ArrayList<>(store.getMessages(id));
        ensureCapacity(messages, maxTokens, tokenizer);
        return messages;
    }
    private static void ensureCapacity(List<ChatMessage> messages, int
maxTokens, Tokenizer tokenizer) {
        int currentTokenCount =
tokenizer.estimateTokenCountInMessages(messages);
        while (currentTokenCount > maxTokens) {
            int messageToRemove = 0;
            if (messages.get(0) instanceof SystemMessage) {
                messageToRemove = 1;
            }
            ChatMessage removedMessage = messages.remove(messageToRemove);
            int tokenCountOfRemovedMessage =
tokenizer.estimateTokenCountInMessage(removedMessage);
            log.trace("Removing the following message ({} tokens) to comply
with the capacity requirements: {}",
                    tokenCountOfRemovedMessage, removedMessage);
            currentTokenCount -= tokenCountOfRemovedMessage;
        }
    }
    @Override
    public void clear() {
        store.deleteMessages(id);
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private Object id = "default";
        private Integer maxTokens;
        private Tokenizer tokenizer;
        private ChatMemoryStore store = new InMemoryChatMemoryStore();
        /**
         * @param id The ID of the {@link ChatMemory}.
         *           If not provided, a "default" will be used.
         * @return builder
         */
        public Builder id(Object id) {
            this.id = id;
```

```
            return this;
        }
        /**
         * @param maxTokens The maximum number of tokens to retain.
         *                  Chat memory will retain as many of the most
recent messages as can fit into {@code maxTokens}.
         *                  Messages are indivisible. If a message doesn't
fit, it's discarded completely.
         * @param tokenizer A {@link Tokenizer} responsible for counting
tokens in the messages.
         * @return builder
         */
        public Builder maxTokens(Integer maxTokens, Tokenizer tokenizer) {
            this.maxTokens = maxTokens;
            this.tokenizer = tokenizer;
            return this;
        }
        /**
         * @param store The chat memory store responsible for storing the
chat memory state.
         *              If not provided, an {@link InMemoryChatMemoryStore}
will be used.
         * @return builder
         */
        public Builder chatMemoryStore(ChatMemoryStore store) {
            this.store = store;
            return this;
        }
        public TokenWindowChatMemory build() {
            return new TokenWindowChatMemory(this);
        }
    }
    public static TokenWindowChatMemory withMaxTokens(int maxTokens,
Tokenizer tokenizer) {
        return builder().maxTokens(maxTokens, tokenizer).build();
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\BigDecimalOutputParser.java

```
package dev.langchain4j.model.output;
import java.math.BigDecimal;
public class BigDecimalOutputParser implements OutputParser<BigDecimal> {
    @Override
    public BigDecimal parse(String string) {
        return new BigDecimal(string);
    }
    @Override
    public String formatInstructions() {
        return "floating point number";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\BigIntegerOutputParser.java

```
package dev.langchain4j.model.output;
import java.math.BigInteger;
public class BigIntegerOutputParser implements OutputParser<BigInteger> {
    @Override
    public BigInteger parse(String string) {
        return new BigInteger(string);
    }
    @Override
    public String formatInstructions() {
        return "integer number";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\BooleanOutputParser.java

```
package dev.langchain4j.model.output;
public class BooleanOutputParser implements OutputParser<Boolean> {
    @Override
    public Boolean parse(String string) {
        return Boolean.parseBoolean(string);
    }
    @Override
    public String formatInstructions() {
        return "one of [true, false]";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\ByteOutputParser.java

```
package dev.langchain4j.model.output;
public class ByteOutputParser implements OutputParser<Byte> {
    @Override
    public Byte parse(String string) {
        return Byte.parseByte(string);
    }
    @Override
    public String formatInstructions() {
        return "integer number in range [-128, 127]";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\DateOutputParser.java

```
package dev.langchain4j.model.output;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateOutputParser implements OutputParser<Date> {
    private static final String DATE_PATTERN = "yyyy-MM-dd";
    private static final SimpleDateFormat SIMPLE_DATE_FORMAT = new
SimpleDateFormat(DATE_PATTERN);
    @Override
    public Date parse(String string) {
        try {
            return SIMPLE_DATE_FORMAT.parse(string);
        } catch (ParseException e) {
            throw new RuntimeException(e);
        }
    }
    @Override
    public String formatInstructions() {
        return DATE_PATTERN;
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\DoubleOutputParser.java

```
package dev.langchain4j.model.output;
public class DoubleOutputParser implements OutputParser<Double> {
    @Override
    public Double parse(String string) {
        return Double.parseDouble(string);
    }
    @Override
    public String formatInstructions() {
        return "floating point number";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\EnumOutputParser.java

```
package dev.langchain4j.model.output;
import dev.langchain4j.internal.Json;
import java.util.Arrays;
public class EnumOutputParser implements OutputParser<Enum> {
    private final Class<? extends Enum> enumClass;
    public EnumOutputParser(Class<? extends Enum> enumClass) {
        this.enumClass = enumClass;
    }
    @Override
    public Enum parse(String string) {
        return Json.fromJson(string, enumClass);
    }
    @Override
    public String formatInstructions() {
        return "one of " + Arrays.toString(enumClass.getEnumConstants());
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\FloatOutputParser.java

```
package dev.langchain4j.model.output;
public class FloatOutputParser implements OutputParser<Float> {
    @Override
    public Float parse(String string) {
        return Float.parseFloat(string);
    }
    @Override
    public String formatInstructions() {
        return "floating point number";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\IntOutputParser.java

```
package dev.langchain4j.model.output;
public class IntOutputParser implements OutputParser<Integer> {
    @Override
    public Integer parse(String string) {
        return Integer.parseInt(string);
    }
    @Override
    public String formatInstructions() {
        return "integer number";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\LocalDateOutputParser.java

```
package dev.langchain4j.model.output;
import java.time.LocalDate;
import static java.time.format.DateTimeFormatter.ISO_LOCAL_DATE;
public class LocalDateOutputParser implements OutputParser<LocalDate> {
    @Override
    public LocalDate parse(String string) {
        return LocalDate.parse(string, ISO_LOCAL_DATE);
    }
    @Override
    public String formatInstructions() {
        return "2023-12-31";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\LocalDateTimeOutputParser.java

```
package dev.langchain4j.model.output;
import java.time.LocalDateTime;
import static java.time.format.DateTimeFormatter.ISO_LOCAL_DATE_TIME;
public class LocalDateTimeOutputParser implements OutputParser<LocalDateTime>
{
    @Override
    public LocalDateTime parse(String string) {
        return LocalDateTime.parse(string, ISO_LOCAL_DATE_TIME);
    }
    @Override
    public String formatInstructions() {
        return "2023-12-31T23:59:59";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\LocalTimeOutputParser.java

```
package dev.langchain4j.model.output;
import java.time.LocalTime;
import static java.time.format.DateTimeFormatter.ISO_LOCAL_TIME;
public class LocalTimeOutputParser implements OutputParser<LocalTime> {
    @Override
    public LocalTime parse(String string) {
        return LocalTime.parse(string, ISO_LOCAL_TIME);
    }
    @Override
    public String formatInstructions() {
        return "23:59:59";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\LongOutputParser.java

```
package dev.langchain4j.model.output;
public class LongOutputParser implements OutputParser<Long> {
    @Override
    public Long parse(String string) {
        return Long.parseLong(string);
    }
    @Override
    public String formatInstructions() {
        return "integer number";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\model\output\ShortOutputParser.java

```
package dev.langchain4j.model.output;
public class ShortOutputParser implements OutputParser<Short> {
    @Override
    public Short parse(String string) {
        return Short.parseShort(string);
    }
    @Override
    public String formatInstructions() {
        return "integer number in range [-32768, 32767]";
    }
}
```

langchain4j\src\main\java\dev\langchain4j\retriever\EmbeddingStoreRetriever.java

```
package dev.langchain4j.retriever;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import java.util.List;
import static java.util.stream.Collectors.toList;
public class EmbeddingStoreRetriever implements Retriever<TextSegment> {
    private final EmbeddingStore<TextSegment> embeddingStore;
    private final EmbeddingModel embeddingModel;
    private final int maxResults;
    private final Double minScore;
    public EmbeddingStoreRetriever(EmbeddingStore<TextSegment> embeddingStore,
                                   EmbeddingModel embeddingModel,
                                   int maxResults,
                                   Double minScore) {
        this.embeddingStore = embeddingStore;
        this.embeddingModel = embeddingModel;
        this.maxResults = maxResults;
        this.minScore = minScore;
    }
    @Override
    public List<TextSegment> findRelevant(String text) {
        Embedding embeddedText = embeddingModel.embed(text).content();
        List<EmbeddingMatch<TextSegment>> relevant;
        if (minScore == null) {
            relevant = embeddingStore.findRelevant(embeddedText, maxResults);
        } else {
            relevant = embeddingStore.findRelevant(embeddedText, maxResults,
minScore);
        }
        return relevant.stream()
                .map(EmbeddingMatch::embedded)
                .collect(toList());
    }
    public static EmbeddingStoreRetriever from(EmbeddingStore<TextSegment>
embeddingStore, EmbeddingModel embeddingModel) {
        return new EmbeddingStoreRetriever(embeddingStore, embeddingModel, 2,
null);
    }
    public static EmbeddingStoreRetriever from(EmbeddingStore<TextSegment>
embeddingStore, EmbeddingModel embeddingModel, int maxResults) {
        return new EmbeddingStoreRetriever(embeddingStore, embeddingModel,
maxResults, null);
    }
    public static EmbeddingStoreRetriever from(EmbeddingStore<TextSegment>
embeddingStore, EmbeddingModel embeddingModel, int maxResults, double
minScore) {
        return new EmbeddingStoreRetriever(embeddingStore, embeddingModel,
maxResults, minScore);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\service\AiServiceContext.java

```
package dev.langchain4j.service;
import dev.langchain4j.agent.tool.ToolExecutor;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.ChatMemoryProvider;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.moderation.ModerationModel;
import dev.langchain4j.retriever.Retriever;
import java.util.List;
import java.util.Map;
class AiServiceContext {
    Class<?> aiServiceClass;
    ChatLanguageModel chatModel;
    StreamingChatLanguageModel streamingChatModel;
    Map</* id */ Object, ChatMemory> chatMemories;
    ChatMemoryProvider chatMemoryProvider;
    ModerationModel moderationModel;
    List<ToolSpecification> toolSpecifications;
    Map<String, ToolExecutor> toolExecutors;
    Retriever<TextSegment> retriever;
    boolean hasChatMemory() {
        return chatMemories != null;
    }
    ChatMemory chatMemory(Object memoryId) {
        return chatMemories.computeIfAbsent(memoryId, ignored ->
chatMemoryProvider.get(memoryId));
    }
}
```

langchain4j\src\main\java\dev\langchain4j\service\AiServices.java

```
package dev.langchain4j.service;
import dev.langchain4j.agent.tool.Tool;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import dev.langchain4j.agent.tool.ToolExecutor;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.ToolExecutionResultMessage;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.ChatMemoryProvider;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.PromptTemplate;
import dev.langchain4j.model.input.structured.StructuredPrompt;
import dev.langchain4j.model.input.structured.StructuredPromptProcessor;
import dev.langchain4j.model.moderation.Moderation;
import dev.langchain4j.model.moderation.ModerationModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.retriever.Retriever;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.lang.reflect.*;
import java.util.*;
import java.util.concurrent.*;
import static
dev.langchain4j.agent.tool.ToolSpecifications.toolSpecificationFrom;
import static dev.langchain4j.data.message.ToolExecutionResultMessage.toolExec
utionResultMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static
dev.langchain4j.exception.IllegalConfigurationException.illegalConfiguration;
import static dev.langchain4j.internal.Exceptions.illegalArgument;
import static
dev.langchain4j.service.ServiceOutputParser.outputFormatInstructions;
import static java.util.Collections.singletonMap;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;
/**
 * AI Services provide a simpler and more flexible alternative to chains.
 * You can define your own API (a Java interface with one or more methods),
 * and AiServices will provide an implementation for it (we call this "AI
Service").
 * <p>
 * Currently, AI Services support:
 * <pre>
 * - Prompt templates for user and system messages using {@link UserMessage}
and {@link SystemMessage}
 * - Structured prompts as method arguments (see {@link StructuredPrompt})
 * - Shared or per-user (see {@link MemoryId}) chat memory
 * - Retrievers
 * - Tools (see {@link Tool})
 * - Various return types (output parsers), see below
 * - Streaming (use {@link TokenStream} as a return type)
 * - Auto-moderation using {@link Moderate}
 * </pre>
 * <p>
```

```
 * Here is the simplest example of an AI Service:
 *
 * <pre>
 * interface Assistant {
 *
 *     String chat(String userMessage);
 * }
 *
 * Assistant assistant = AiServices.create(Assistant.class, model);
 *
 * String answer = assistant.chat("hello");
 * System.out.println(answer); // Hello, how can I help you today?
 * </pre>
 *
 * <pre>
 * The return type of methods in your AI Service can be any of the following:
 * - a {@code String} or an {@link AiMessage}, if you want to get the answer
from the LLM as-is
 * - a {@code List<String>} or {@code Set<String>}, if you want to receive
the answer as a collection of items or bullet points
 * - any {@code Enum} or a {@code boolean}, if you want to use the LLM for
classification
 * - a primitive or boxed Java type: {@code int}, {@code Double}, etc., if
you want to use the LLM for data extraction
 * - many default Java types: {@code Date}, {@code LocalDateTime}, {@code
BigDecimal}, etc., if you want to use the LLM for data extraction
 * - any custom POJO, if you want to use the LLM for data extraction
 * </pre>
 * <p>
 * Let's see how we can classify the sentiment of a text:
 * <pre>
 * enum Sentiment {
 *     POSITIVE, NEUTRAL, NEGATIVE
 * }
 *
 * interface SentimentAnalyzer {
 *
 *     {@code @UserMessage}("Analyze sentiment of {{it}}")
 *     Sentiment analyzeSentimentOf(String text);
 * }
 *
 * SentimentAnalyzer assistant = AiServices.create(SentimentAnalyzer.class,
model);
 *
 * Sentiment sentiment = analyzeSentimentOf.chat("I love you");
 * System.out.println(sentiment); // POSITIVE
 * </pre>
 * <p>
 * As demonstrated, you can put {@link UserMessage} and {@link SystemMessage}
annotations above a method to define
 * templates for user and system messages, respectively.
 * In this example, the special {@code {{it}}} prompt template variable is
used because there's only one method parameter.
 * However, you can use more parameters as demonstrated in the following
example:
 * <pre>
 * interface Translator {
 *
 *     {@code @SystemMessage}("You are a professional translator into
{{language}}")
 *     {@code @UserMessage}("Translate the following text: {{text}}")
```

```
 *     String translate(@V("text") String text, @V("language") String
language);
 * }
 * </pre>
 * <p>
 * See more examples <a href="https://github.com/langchain4j/langchain4j-
examples/tree/main/other-examples/src/main/java">here</a>.
 *
 * @param <T> The interface for which AiServices will provide an
implementation.
 */
public class AiServices<T> {
    private final Logger log = LoggerFactory.getLogger(AiServices.class);
    private static final String DEFAULT = "default";
    private final AiServiceContext context = new AiServiceContext();
    private AiServices(Class<T> aiServiceClass) {
        context.aiServiceClass = aiServiceClass;
    }
    /**
     * Creates an AI Service (an implementation of the provided interface),
that is backed by the provided chat model.
     * This convenience method can be used to create simple AI Services.
     * For more complex cases, please use {@link #builder}.
     *
     * @param aiService         The class of the interface to be implemented.
     * @param chatLanguageModel The chat model to be used under the hood.
     * @return An instance of the provided interface, implementing all its
defined methods.
     */
    public static <T> T create(Class<T> aiService, ChatLanguageModel
chatLanguageModel) {
        return builder(aiService)
                .chatLanguageModel(chatLanguageModel)
                .build();
    }
    /**
     * Creates an AI Service (an implementation of the provided interface),
that is backed by the provided streaming chat model.
     * This convenience method can be used to create simple AI Services.
     * For more complex cases, please use {@link #builder}.
     *
     * @param aiService                 The class of the interface to be
implemented.
     * @param streamingChatLanguageModel The streaming chat model to be used
under the hood.
     *                                  The return type of all methods
should be {@link TokenStream}.
     * @return An instance of the provided interface, implementing all its
defined methods.
     */
    public static <T> T create(Class<T> aiService, StreamingChatLanguageModel
streamingChatLanguageModel) {
        return builder(aiService)
                .streamingChatLanguageModel(streamingChatLanguageModel)
                .build();
    }
    /**
     * Begins the construction of an AI Service.
     *
     * @param aiService The class of the interface to be implemented.
     * @return builder
```

```
     */
    public static <T> AiServices<T> builder(Class<T> aiService) {
        return new AiServices<>(aiService);
    }
    /**
     * Configures chat model that will be used under the hood of the AI
Service.
     * <p>
     * Either {@link ChatLanguageModel} or {@link StreamingChatLanguageModel}
should be configured,
     * but not both at the same time.
     *
     * @param chatLanguageModel Chat model that will be used under the hood
of the AI Service.
     * @return builder
     */
    public AiServices<T> chatLanguageModel(ChatLanguageModel
chatLanguageModel) {
        context.chatModel = chatLanguageModel;
        return this;
    }
    /**
     * Configures streaming chat model that will be used under the hood of
the AI Service.
     * The methods of the AI Service must return a {@link TokenStream} type.
     * <p>
     * Either {@link ChatLanguageModel} or {@link StreamingChatLanguageModel}
should be configured,
     * but not both at the same time.
     *
     * @param streamingChatLanguageModel Streaming chat model that will be
used under the hood of the AI Service.
     * @return builder
     */
    public AiServices<T>
streamingChatLanguageModel(StreamingChatLanguageModel
streamingChatLanguageModel) {
        context.streamingChatModel = streamingChatLanguageModel;
        return this;
    }
    /**
     * Configures the chat memory that will be used to preserve conversation
history between method calls.
     * <p>
     * Unless a {@link ChatMemory} or {@link ChatMemoryProvider} is
configured, all method calls will be independent of each other.
     * In other words, the LLM will not remember the conversation from the
previous method calls.
     * <p>
     * The same {@link ChatMemory} instance will be used for every method
call.
     * <p>
     * If you want to have a separate {@link ChatMemory} for each user/
conversation, configure {@link #chatMemoryProvider} instead.
     * <p>
     * Either a {@link ChatMemory} or a {@link ChatMemoryProvider} can be
configured, but not both simultaneously.
     *
     * @param chatMemory An instance of chat memory to be used by the AI
Service.
     * @return builder
```

```
     */
    public AiServices<T> chatMemory(ChatMemory chatMemory) {
        context.chatMemories = new ConcurrentHashMap<>();
        context.chatMemories.put(DEFAULT, chatMemory);
        return this;
    }
    /**
     * Configures the chat memory provider, which provides a dedicated
instance of {@link ChatMemory} for each user/conversation.
     * To distinguish between users/conversations, one of the method's
arguments should be a memory ID (of any data type)
     * annotated with {@link MemoryId}.
     * For each new (previously unseen) memoryId, an instance of {@link
ChatMemory} will be automatically obtained
     * by invoking {@link ChatMemoryProvider#get(Object id)}.
     * Example:
     * <pre>
     * interface Assistant {
     *
     *     String chat(@MemoryId int memoryId, @UserMessage String message);
     * }
     * </pre>
     * If you prefer to use the same (shared) {@link ChatMemory} for all
users/conversations, configure a {@link #chatMemory} instead.
     * <p>
     * Either a {@link ChatMemory} or a {@link ChatMemoryProvider} can be
configured, but not both simultaneously.
     *
     * @param chatMemoryProvider The provider of a {@link ChatMemory} for
each new user/conversation.
     * @return builder
     */
    public AiServices<T> chatMemoryProvider(ChatMemoryProvider
chatMemoryProvider) {
        context.chatMemories = new ConcurrentHashMap<>();
        context.chatMemoryProvider = chatMemoryProvider;
        return this;
    }
    /**
     * Configures a moderation model to be used for automatic content
moderation.
     * If a method in the AI Service is annotated with {@link Moderate}, the
moderation model will be invoked
     * to check the user content for any inappropriate or harmful material.
     *
     * @param moderationModel The moderation model to be used for content
moderation.
     * @return builder
     * @see Moderate
     */
    public AiServices<T> moderationModel(ModerationModel moderationModel) {
        context.moderationModel = moderationModel;
        return this;
    }
    /**
     * Configures the tools that the LLM can use.
     * A {@link ChatMemory} that can hold at least 3 messages is required for
the tools to work properly.
     *
     * @param objectsWithTools One or more objects whose methods are
annotated with {@link Tool}.
```

```
     *                           All these tools (methods annotated with {@link
Tool}) will be accessible to the LLM.
     *                           Note that inherited methods are ignored.
     * @return builder
     * @see Tool
     */
    public AiServices<T> tools(Object... objectsWithTools) {
        return tools(Arrays.asList(objectsWithTools));
    }
    /**
     * Configures the tools that the LLM can use.
     * A {@link ChatMemory} that can hold at least 3 messages is required for
the tools to work properly.
     *
     * @param objectsWithTools A list of objects whose methods are annotated
with {@link Tool}.
     *                           All these tools (methods annotated with {@link
Tool}) are accessible to the LLM.
     *                           Note that inherited methods are ignored.
     * @return builder
     * @see Tool
     */
    public AiServices<T> tools(List<Object> objectsWithTools) {
        context.toolSpecifications = new ArrayList<>();
        context.toolExecutors = new HashMap<>();
        for (Object objectWithTool : objectsWithTools) {
            for (Method method :
objectWithTool.getClass().getDeclaredMethods()) {
                if (method.isAnnotationPresent(Tool.class)) {
                    ToolSpecification toolSpecification =
toolSpecificationFrom(method);
                    context.toolSpecifications.add(toolSpecification);
                    context.toolExecutors.put(toolSpecification.name(), new
ToolExecutor(objectWithTool, method));
                }
            }
        }
        return this;
    }
    // TODO separate retriever per user
    // TODO way to configure custom prompt with original message and context
    // TODO callback to transform/filter retrieved segments
    /**
     * Configures a retriever that will be invoked on every method call to
fetch relevant information
     * related to the current user message from an underlying source (e.g.,
embedding store).
     * This relevant information is automatically injected into the message
sent to the LLM.
     *
     * @param retriever The retriever to be used by the AI Service.
     * @return builder
     */
    public AiServices<T> retriever(Retriever<TextSegment> retriever) {
        context.retriever = retriever;
        return this;
    }
    /**
     * Constructs and returns the AI Service.
     *
     * @return An instance of the AI Service implementing the specified
```

```
interface.
     */
    public T build() {
        if (context.chatModel == null && context.streamingChatModel == null) {
            throw illegalConfiguration("Please specify either
chatLanguageModel or streamingChatLanguageModel");
        }
        for (Method method : context.aiServiceClass.getMethods()) {
            if (method.isAnnotationPresent(Moderate.class) &&
context.moderationModel == null) {
                throw illegalConfiguration("The @Moderate annotation is
present, but the moderationModel is not set up. " +
                        "Please ensure a valid moderationModel is configured
before using the @Moderate annotation.");
            }
        }
        if (context.toolSpecifications != null && !context.hasChatMemory()) {
            throw illegalConfiguration(
                    "Please set up chatMemory or chatMemoryProvider in order
to use tools. "
                            + "A ChatMemory that can hold at least 3 messages
is required for the tools to work properly. "
                            + "While the LLM can technically execute a tool
without chat memory, if it only receives the " +
                            "result of the tool's execution without the
initial message from the user, it won't interpret " +
                            "the result properly."
            );
        }
        Object proxyInstance = Proxy.newProxyInstance(
                context.aiServiceClass.getClassLoader(),
                new Class<?>[]{context.aiServiceClass},
                new InvocationHandler() {
                    private final ExecutorService executor =
Executors.newCachedThreadPool();
                    @Override
                    public Object invoke(Object proxy, Method method,
Object[] args) throws Exception {
                        if (method.getDeclaringClass() == Object.class) {
                            // methods like equals(), hashCode() and
toString() should not be handled by this proxy
                            return method.invoke(this, args);
                        }
                        validateParameters(method);
                        Optional<ChatMessage> systemMessage =
prepareSystemMessage(method, args);
                        ChatMessage userMessage = prepareUserMessage(method,
args);
                        if (context.retriever != null) { // TODO extract
method/class
                            List<TextSegment> relevant =
context.retriever.findRelevant(userMessage.text());
                            if (relevant == null || relevant.isEmpty()) {
                                log.debug("No relevant information was
found");
                            } else {
                                String relevantConcatenated =
relevant.stream()
                                        .map(TextSegment::text)
                                        .collect(joining("\n\n"));
                                log.debug("Retrieved relevant information:\n"
```

```
+ relevantConcatenated + "\n");
                                userMessage = userMessage(userMessage.text()
                                        + "\n\nHere is some information that
might be useful for answering:\n\n"
                                        + relevantConcatenated);
                            }
                        }
                        Object memoryId = memoryId(method,
args).orElse(DEFAULT);
                        if (context.hasChatMemory()) {
                            ChatMemory chatMemory =
context.chatMemory(memoryId);
                            systemMessage.ifPresent(chatMemory::add);
                            chatMemory.add(userMessage);
                        }
                        List<ChatMessage> messages;
                        if (context.hasChatMemory()) {
                            messages =
context.chatMemory(memoryId).messages();
                        } else {
                            messages = new ArrayList<>();
                            systemMessage.ifPresent(messages::add);
                            messages.add(userMessage);
                        }
                        Future<Moderation> moderationFuture =
triggerModerationIfNeeded(method, messages);
                        if (method.getReturnType() == TokenStream.class) {
                            return new AiServiceTokenStream(messages,
context, memoryId); // TODO moderation
                        }
                        Response<AiMessage> response =
context.toolSpecifications != null ?
                                context.chatModel.generate(messages,
context.toolSpecifications) :
                                context.chatModel.generate(messages);
                        verifyModerationIfNeeded(moderationFuture);
                        ToolExecutionRequest toolExecutionRequest;
                        while (true) { // TODO limit number of cycles
                            if (context.hasChatMemory()) {

context.chatMemory(memoryId).add(response.content());
                            }
                            toolExecutionRequest =
response.content().toolExecutionRequest();
                            if (toolExecutionRequest == null) {
                                break;
                            }
                            ToolExecutor toolExecutor =
context.toolExecutors.get(toolExecutionRequest.name());
                            String toolExecutionResult =
toolExecutor.execute(toolExecutionRequest);
                            ToolExecutionResultMessage
toolExecutionResultMessage
                                    =
toolExecutionResultMessage(toolExecutionRequest.name(), toolExecutionResult);
                            ChatMemory chatMemory =
context.chatMemory(memoryId);
                            chatMemory.add(toolExecutionResultMessage);
                            response =
context.chatModel.generate(chatMemory.messages(), context.toolSpecifications);
                        }
```

```
                        return ServiceOutputParser.parse(response,
method.getReturnType());
                    }
                    private Future<Moderation>
triggerModerationIfNeeded(Method method, List<ChatMessage> messages) {
                        if (method.isAnnotationPresent(Moderate.class)) {
                            return executor.submit(() -> {
                                List<ChatMessage> messagesToModerate =
removeToolMessages(messages);
                                return
context.moderationModel.moderate(messagesToModerate).content();
                            });
                        }
                        return null;
                    }
                    private List<ChatMessage>
removeToolMessages(List<ChatMessage> messages) {
                        return messages.stream()
                                .filter(it -> !(it instanceof
ToolExecutionResultMessage))
                                .filter(it -> !(it instanceof AiMessage &&
((AiMessage) it).toolExecutionRequest() != null))
                                .collect(toList());
                    }
                    private void verifyModerationIfNeeded(Future<Moderation>
moderationFuture) {
                        if (moderationFuture != null) {
                            try {
                                Moderation moderation =
moderationFuture.get();
                                if (moderation.flagged()) {
                                    throw new
ModerationException(String.format("Text \"%s\" violates content policy",
moderation.flaggedText()));
                                }
                            } catch (InterruptedException |
ExecutionException e) {
                                throw new RuntimeException(e);
                            }
                        }
                    }
                });
        return (T) proxyInstance;
    }
    private Optional<Object> memoryId(Method method, Object[] args) {
        Parameter[] parameters = method.getParameters();
        for (int i = 0; i < parameters.length; i++) {
            if (parameters[i].isAnnotationPresent(MemoryId.class)) {
                Object memoryId = args[i];
                if (memoryId == null) {
                    throw illegalArgument("The value of parameter %s
annotated with @MemoryId in method %s must not be null",
                            parameters[i].getName(), method.getName());
                }
                return Optional.of(memoryId);
            }
        }
        return Optional.empty();
    }
    private Optional<ChatMessage> prepareSystemMessage(Method method,
Object[] args) {
```

```
        Parameter[] parameters = method.getParameters();
        Map<String, Object> variables = getPromptTemplateVariables(args,
parameters);
        SystemMessage annotation = method.getAnnotation(SystemMessage.class);
        if (annotation != null) {
            String systemMessageTemplate =
String.join(annotation.delimiter(), annotation.value());
            if (systemMessageTemplate.isEmpty()) {
                throw illegalConfiguration("@SystemMessage's template cannot
be empty");
            }
            Prompt prompt =
PromptTemplate.from(systemMessageTemplate).apply(variables);
            return Optional.of(prompt.toSystemMessage());
        }
        return Optional.empty();
    }
    private static ChatMessage prepareUserMessage(Method method, Object[]
args) {
        Parameter[] parameters = method.getParameters();
        Map<String, Object> variables = getPromptTemplateVariables(args,
parameters);
        String outputFormatInstructions =
outputFormatInstructions(method.getReturnType());
        String userName = getUserName(parameters, args);
        UserMessage annotation = method.getAnnotation(UserMessage.class);
        if (annotation != null) {
            String userMessageTemplate = String.join(annotation.delimiter(),
annotation.value()) + outputFormatInstructions;
            if (userMessageTemplate.contains("{{it}}")) {
                if (parameters.length != 1) {
                    throw illegalConfiguration("Error: The {{it}} placeholder
is present but the method does not have exactly one parameter. " +
                            "Please ensure that methods using the {{it}}
placeholder have exactly one parameter.");
                }
                variables = singletonMap("it", toString(args[0]));
            }
            Prompt prompt =
PromptTemplate.from(userMessageTemplate).apply(variables);
            return userMessage(userName, prompt.text());
        }
        for (int i = 0; i < parameters.length; i++) {
            if (parameters[i].isAnnotationPresent(UserMessage.class)) {
                return userMessage(userName, toString(args[i]) +
outputFormatInstructions);
            }
        }
        if (args == null || args.length == 0) {
            throw illegalConfiguration("Method should have at least one
argument");
        }
        if (args.length == 1) {
            return userMessage(userName, toString(args[0]) +
outputFormatInstructions);
        }
        throw illegalConfiguration("For methods with multiple parameters,
each parameter must be annotated with @V, @UserMessage, @UserName or
@MemoryId");
    }
    private static String getUserName(Parameter[] parameters, Object[] args) {
```

```
        for (int i = 0; i < parameters.length; i++) {
            if (parameters[i].isAnnotationPresent(UserName.class)) {
                return args[i].toString();
            }
        }
        return null;
    }
    private static void validateParameters(Method method) {
        Parameter[] parameters = method.getParameters();
        if (parameters == null || parameters.length < 2) {
            return;
        }
        for (Parameter parameter : parameters) {
            V v = parameter.getAnnotation(V.class);
            UserMessage userMessage =
parameter.getAnnotation(UserMessage.class);
            MemoryId memoryId = parameter.getAnnotation(MemoryId.class);
            UserName userName = parameter.getAnnotation(UserName.class);
            if (v == null && userMessage == null && memoryId == null &&
userName == null) {
                throw illegalConfiguration(
                        "Parameter '%s' of method '%s' should be annotated
with @V or @UserMessage or @UserName or @MemoryId",
                        parameter.getName(), method.getName()
                );
            }
        }
    }
    private static Map<String, Object> getPromptTemplateVariables(Object[]
args, Parameter[] parameters) {
        Map<String, Object> variables = new HashMap<>();
        for (int i = 0; i < parameters.length; i++) {
            V varAnnotation = parameters[i].getAnnotation(V.class);
            if (varAnnotation != null) {
                String variableName = varAnnotation.value();
                Object variableValue = args[i];
                variables.put(variableName, variableValue);
            }
        }
        return variables;
    }
    private static Object toString(Object arg) {
        if (arg.getClass().isArray()) {
            return arrayToString(arg);
        } else if
(arg.getClass().isAnnotationPresent(StructuredPrompt.class)) {
            return StructuredPromptProcessor.toPrompt(arg).text();
        } else {
            return arg.toString();
        }
    }
    private static String arrayToString(Object arg) {
        StringBuilder sb = new StringBuilder("[");
        int length = Array.getLength(arg);
        for (int i = 0; i < length; i++) {
            sb.append(toString(Array.get(arg, i)));
            if (i < length - 1) {
                sb.append(", ");
            }
        }
        sb.append("]");
```

```
        return sb.toString();
    }
}
```

langchain4j\src\main\java\dev\langchain4j\service\AiServiceStreamingResponseHandler.java

```
package dev.langchain4j.service;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import dev.langchain4j.agent.tool.ToolExecutor;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ToolExecutionResultMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.function.Consumer;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Handles response from a language model for AI Service that is streamed token-by-token.
 * Handles both regular (text) responses and responses with the request to execute a tool.
 */
class AiServiceStreamingResponseHandler implements StreamingResponseHandler<AiMessage> {
    private final Logger log = LoggerFactory.getLogger(AiServiceStreamingResponseHandler.class);
    private final AiServiceContext context;
    private final Object memoryId;
    private final Consumer<String> tokenHandler;
    private final Consumer<Response<AiMessage>> completionHandler;
    private final Consumer<Throwable> errorHandler;
    private final TokenUsage tokenUsage;
    AiServiceStreamingResponseHandler(AiServiceContext context,
                                      Object memoryId,
                                      Consumer<String> tokenHandler,
                                      Consumer<Response<AiMessage>> completionHandler,
                                      Consumer<Throwable> errorHandler,
                                      TokenUsage tokenUsage) {
        this.context = ensureNotNull(context, "context");
        this.memoryId = ensureNotNull(memoryId, "memoryId");
        this.tokenHandler = ensureNotNull(tokenHandler, "tokenHandler");
        this.completionHandler = completionHandler;
        this.errorHandler = errorHandler;
        this.tokenUsage = ensureNotNull(tokenUsage, "tokenUsage");
    }
    @Override
    public void onNext(String token) {
        tokenHandler.accept(token);
    }
    @Override
    public void onComplete(Response<AiMessage> response) {
        if (context.hasChatMemory()) {
            context.chatMemory(memoryId).add(response.content());
        }
        ToolExecutionRequest toolExecutionRequest = response.content().toolExecutionRequest();
        if (toolExecutionRequest != null) {
            ToolExecutor toolExecutor = context.toolExecutors.get(toolExecutionRequest.name());
            String toolExecutionResult =
```

```
toolExecutor.execute(toolExecutionRequest);
            ToolExecutionResultMessage toolExecutionResultMessage =
ToolExecutionResultMessage.from(
                    toolExecutionRequest.name(),
                    toolExecutionResult
            );
            context.chatMemory(memoryId).add(toolExecutionResultMessage);
            context.streamingChatModel.generate(
                    context.chatMemory(memoryId).messages(),
                    context.toolSpecifications,
                    new AiServiceStreamingResponseHandler(
                            context,
                            memoryId,
                            tokenHandler,
                            completionHandler,
                            errorHandler,
                            tokenUsage.add(response.tokenUsage())
                    )
            );
        } else {
            if (completionHandler != null) {
                completionHandler.accept(Response.from(
                        response.content(),
                        tokenUsage.add(response.tokenUsage()),
                        response.finishReason())
                );
            }
        }
    }
    @Override
    public void onError(Throwable error) {
        if (errorHandler != null) {
            try {
                errorHandler.accept(error);
            } catch (Exception e) {
                log.error("While handling the following error...", error);
                log.error("...the following error happened", e);
            }
        } else {
            log.warn("Ignored error", error);
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\service\AiServiceTokenStream.java

```
package dev.langchain4j.service;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import java.util.List;
import java.util.function.Consumer;
import static dev.langchain4j.internal.ValidationUtils.ensureNotEmpty;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
class AiServiceTokenStream implements TokenStream {
    private final List<ChatMessage> messagesToSend;
    private final AiServiceContext context;
    private final Object memoryId;
    AiServiceTokenStream(List<ChatMessage> messagesToSend, AiServiceContext
context, Object memoryId) {
        this.messagesToSend = ensureNotEmpty(messagesToSend,
"messagesToSend");
        this.context = ensureNotNull(context, "context");
        this.memoryId = ensureNotNull(memoryId, "memoryId");
        ensureNotNull(context.streamingChatModel, "streamingChatModel");
    }
    @Override
    public OnCompleteOrOnError onNext(Consumer<String> tokenHandler) {
        return new OnCompleteOrOnError() {
            @Override
            public OnError onComplete(Consumer<Response<AiMessage>>
completionHandler) {
                return new OnError() {
                    @Override
                    public OnStart onError(Consumer<Throwable> errorHandler) {
                        return new AiServiceOnStart(tokenHandler,
completionHandler, errorHandler);
                    }
                    @Override
                    public OnStart ignoreErrors() {
                        return new AiServiceOnStart(tokenHandler,
completionHandler, null);
                    }
                };
            }
            @Override
            public OnStart onError(Consumer<Throwable> errorHandler) {
                return new AiServiceOnStart(tokenHandler, null, errorHandler);
            }
            @Override
            public OnStart ignoreErrors() {
                return new AiServiceOnStart(tokenHandler, null, null);
            }
        };
    }
    private class AiServiceOnStart implements OnStart {
        private final Consumer<String> tokenHandler;
        private final Consumer<Response<AiMessage>> completionHandler;
        private final Consumer<Throwable> errorHandler;
        private AiServiceOnStart(Consumer<String> tokenHandler,
                                 Consumer<Response<AiMessage>>
completionHandler,
                                 Consumer<Throwable> errorHandler) {
```

```
            this.tokenHandler = ensureNotNull(tokenHandler, "tokenHandler");
            this.completionHandler = completionHandler;
            this.errorHandler = errorHandler;
        }
        @Override
        public void start() {
            context.streamingChatModel.generate(
                    messagesToSend,
                    context.toolSpecifications,
                    new AiServiceStreamingResponseHandler(
                            context,
                            memoryId,
                            tokenHandler,
                            completionHandler,
                            errorHandler,
                            new TokenUsage()
                    )
            );
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\service\MemoryId.java

```
package dev.langchain4j.service;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
/**
 * The value of a method parameter annotated with @MemoryId will be used to
find the memory belonging to that user/conversation.
 * A parameter annotated with @MemoryId can be of any type, provided it has
properly implemented equals() and hashCode() methods.
 */
@Retention(RUNTIME)
@Target(PARAMETER)
public @interface MemoryId {
}
```

langchain4j\src\main\java\dev\langchain4j\service\Moderate.java

```
package dev.langchain4j.service;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
/**
 * When a method in the AI Service is annotated with @Moderate, each
invocation of this method will call not only the LLM,
 * but also the moderation model (which must be provided during the
construction of the AI Service) in parallel.
 * This ensures that no malicious content is supplied by the user.
 * Before the method returns an answer from the LLM, it will wait until the
moderation model returns a result.
 * If the moderation model flags the content, a ModerationException will be
thrown.
 * There is also an option to moderate user input *before* sending it to the
LLM. If you require this functionality,
 * please open an issue.
 */
@Target(METHOD)
@Retention(RUNTIME)
public @interface Moderate {
}
```

langchain4j\src\main\java\dev\langchain4j\service\ModerationException.java

```
package dev.langchain4j.service;
/**
 * Thrown when content moderation fails, i.e., when content is flagged by the
moderation model.
 *
 * @see Moderate
 * @see dev.langchain4j.model.moderation.ModerationModel
 */
public class ModerationException extends RuntimeException {
    public ModerationException(String message) {
        super(message);
    }
}
```

langchain4j\src\main\java\dev\langchain4j\service\OnCompleteOrOnError.java

```
package dev.langchain4j.service;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.output.Response;
import java.util.function.Consumer;
public interface OnCompleteOrOnError {
    /**
     * The provided consumer will be invoked when a language model finishes
streaming a response.
     *
     * @param completionHandler lambda that will be invoked when language
model finishes streaming
     * @return the next step of the step-builder
     */
    OnError onComplete(Consumer<Response<AiMessage>> completionHandler);
    /**
     * The provided consumer will be invoked when an error occurs during
streaming.
     *
     * @param errorHandler lambda that will be invoked when an error occurs
     * @return the next step of the step-builder
     */
    OnStart onError(Consumer<Throwable> errorHandler);
    /**
     * All errors during streaming will be ignored (but will be logged with a
WARN log level).
     *
     * @return the next step of the step-builder
     */
    OnStart ignoreErrors();
}
```

langchain4j\src\main\java\dev\langchain4j\service\OnError.java

```
package dev.langchain4j.service;
import java.util.function.Consumer;
public interface OnError {
    /**
     * The provided Consumer will be invoked when an error occurs during
streaming.
     *
     * @param errorHandler lambda that will be invoked when an error occurs
     * @return the next step of the step-builder
     */
    OnStart onError(Consumer<Throwable> errorHandler);
    /**
     * All errors during streaming will be ignored (but will be logged with a
WARN log level).
     *
     * @return the next step of the step-builder
     */
    OnStart ignoreErrors();
}
```

langchain4j\src\main\java\dev\langchain4j\service\OnStart.java

```
package dev.langchain4j.service;
public interface OnStart {
    /**
     * Invoke this method to send a request to LLM and start response
streaming.
     */
    void start();
}
```

langchain4j\src\main\java\dev\langchain4j\service\ServiceOutputParser.java

```
package dev.langchain4j.service;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.internal.Json;
import dev.langchain4j.model.output.*;
import dev.langchain4j.model.output.structured.Description;
import java.lang.reflect.Field;
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.*;
import static
dev.langchain4j.exception.IllegalConfigurationException.illegalConfiguration;
import static java.lang.String.format;
import static java.util.Arrays.asList;
class ServiceOutputParser {
    private static final Map<Class<?>, OutputParser<?>> OUTPUT_PARSERS = new
HashMap<>();
    static {
        OUTPUT_PARSERS.put(boolean.class, new BooleanOutputParser());
        OUTPUT_PARSERS.put(Boolean.class, new BooleanOutputParser());
        OUTPUT_PARSERS.put(byte.class, new ByteOutputParser());
        OUTPUT_PARSERS.put(Byte.class, new ByteOutputParser());
        OUTPUT_PARSERS.put(short.class, new ShortOutputParser());
        OUTPUT_PARSERS.put(Short.class, new ShortOutputParser());
        OUTPUT_PARSERS.put(int.class, new IntOutputParser());
        OUTPUT_PARSERS.put(Integer.class, new IntOutputParser());
        OUTPUT_PARSERS.put(long.class, new LongOutputParser());
        OUTPUT_PARSERS.put(Long.class, new LongOutputParser());
        OUTPUT_PARSERS.put(BigInteger.class, new BigIntegerOutputParser());
        OUTPUT_PARSERS.put(float.class, new FloatOutputParser());
        OUTPUT_PARSERS.put(Float.class, new FloatOutputParser());
        OUTPUT_PARSERS.put(double.class, new DoubleOutputParser());
        OUTPUT_PARSERS.put(Double.class, new DoubleOutputParser());
        OUTPUT_PARSERS.put(BigDecimal.class, new BigDecimalOutputParser());
        OUTPUT_PARSERS.put(Date.class, new DateOutputParser());
        OUTPUT_PARSERS.put(LocalDate.class, new LocalDateOutputParser());
        OUTPUT_PARSERS.put(LocalTime.class, new LocalTimeOutputParser());
        OUTPUT_PARSERS.put(LocalDateTime.class, new
LocalDateTimeOutputParser());
    }
    static Object parse(Response<AiMessage> response, Class<?> returnType) {
        if (returnType == Response.class) {
            return response;
        }
        AiMessage aiMessage = response.content();
        if (returnType == AiMessage.class) {
            return aiMessage;
        }
        String text = aiMessage.text();
        if (returnType == String.class) {
            return text;
        }
        OutputParser<?> outputParser = OUTPUT_PARSERS.get(returnType);
        if (outputParser != null) {
```

```
            return outputParser.parse(text);
        }
        if (returnType == List.class) {
            return asList(text.split("\n"));
        }
        if (returnType == Set.class) {
            return new HashSet<>(asList(text.split("\n")));
        }
        return Json.fromJson(text, returnType);
    }
    static String outputFormatInstructions(Class<?> returnType) {
        if (returnType == String.class
                || returnType == AiMessage.class
                || returnType == TokenStream.class
                || returnType == Response.class) {
            return "";
        }
        if (returnType == void.class) {
            throw illegalConfiguration("Return type of method '%s' cannot be
void");
        }
        if (returnType.isEnum()) {
            String formatInstructions = new
EnumOutputParser(returnType.asSubclass(Enum.class)).formatInstructions();
            return "\nYou must answer strictly in the following format: " +
formatInstructions;
        }
        OutputParser<?> outputParser = OUTPUT_PARSERS.get(returnType);
        if (outputParser != null) {
            String formatInstructions = outputParser.formatInstructions();
            return "\nYou must answer strictly in the following format: " +
formatInstructions;
        }
        if (returnType == List.class || returnType == Set.class) {
            return "\nYou must put every item on a separate line.";
        }
        return "\nYou must answer strictly in the following JSON format: " +
jsonStructure(returnType);
    }
    private static String jsonStructure(Class<?> structured) {
        StringBuilder jsonSchema = new StringBuilder();
        jsonSchema.append("{\n");
        for (Field field : structured.getDeclaredFields()) {
            jsonSchema.append(format("\"%s\": (%s),\n", field.getName(),
descriptionFor(field)));
        }
        jsonSchema.append("}");
        return jsonSchema.toString();
    }
    private static String descriptionFor(Field field) {
        Description fieldDescription = field.getAnnotation(Description.class);
        if (fieldDescription == null) {
            return "type: " + typeOf(field);
        }
        return String.join(" ", fieldDescription.value()) + "; type: " +
typeOf(field);
    }
    private static String typeOf(Field field) {
        Type type = field.getGenericType();
        if (type instanceof ParameterizedType) {
            ParameterizedType parameterizedType = (ParameterizedType) type;
```

```
            Type[] typeArguments = parameterizedType.getActualTypeArguments();
            if (parameterizedType.getRawType().equals(List.class)
                    || parameterizedType.getRawType().equals(Set.class)) {
                return format("array of %s",
simpleTypeName(typeArguments[0]));
            }
        } else if (field.getType().isArray()) {
            return format("array of %s",
simpleTypeName(field.getType().getComponentType()));
        } else if (((Class<?>) type).isEnum()) {
            return "enum, must be one of " + Arrays.toString(((Class<?>)
type).getEnumConstants());
        }
        return simpleTypeName(type);
    }
    private static String simpleTypeName(Type type) {
        switch (type.getTypeName()) {
            case "java.lang.String":
                return "string";
            case "java.lang.Integer":
            case "int":
                return "integer";
            case "java.lang.Boolean":
            case "boolean":
                return "boolean";
            case "java.lang.Float":
            case "float":
                return "float";
            case "java.lang.Double":
            case "double":
                return "double";
            case "java.util.Date":
            case "java.time.LocalDate":
                return "date string (2023-12-31)";
            case "java.time.LocalTime":
                return "time string (23:59:59)";
            case "java.time.LocalDateTime":
                return "date-time string (2023-12-31T23:59:59)";
            default:
                return type.getTypeName();
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\service\SystemMessage.java

```
package dev.langchain4j.service;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
@Target(METHOD)
@Retention(RUNTIME)
public @interface SystemMessage {
    /**
     * Prompt template can be defined in one line or multiple lines.
     * If the template is defined in multiple lines, the lines will be joined
with a delimiter defined below.
     */
    String[] value();
    String delimiter() default "\n";
}
```

langchain4j\src\main\java\dev\langchain4j\service\TokenStream.java

```
package dev.langchain4j.service;
import java.util.function.Consumer;
/**
 * Represents a token stream from language model to which you can subscribe
and receive updates
 * when a new token is available, when language model finishes streaming, or
when an error occurs during streaming.
 * It is intended to be used as a return type in AI Service.
 */
public interface TokenStream {
    /**
     * The provided consumer will be invoked every time a new token from a
language model is available.
     *
     * @param tokenHandler lambda that consumes tokens of the response
     * @return the next step of a step-builder
     */
    OnCompleteOrOnError onNext(Consumer<String> tokenHandler);
}
```

langchain4j\src\main\java\dev\langchain4j\service\UserMessage.java

```
package dev.langchain4j.service;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
@Retention(RUNTIME)
@Target({METHOD, PARAMETER})
public @interface UserMessage {
    /**
     * Prompt template can be defined in one line or multiple lines.
     * If the template is defined in multiple lines, the lines will be joined
with a delimiter defined below.
     */
    String[] value() default "";
    String delimiter() default "\n";
}
```

langchain4j\src\main\java\dev\langchain4j\service\UserName.java

```
package dev.langchain4j.service;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
/**
 * The value of a method parameter annotated with @UserName will be injected
into the field 'name' of a UserMessage.
 */
@Retention(RUNTIME)
@Target(PARAMETER)
public @interface UserName {
}
```

langchain4j\src\main\java\dev\langchain4j\service\V.java

```
package dev.langchain4j.service;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
/**
 * The values of method parameters annotated with @V, together with prompt
templates defined by @UserMessage
 * and @SystemMessage, are used to produce a message that will be sent to the
LLM.
 * Variables (placeholders), like {{xxx}} in prompt templates, are filled
with the corresponding values
 * of parameters annotated with @V("xxx").
 * <p>
 * Example:
 * <pre>
 * {@code @UserMessage("Hello, my name is {{name}}. I am {{age}} years old.")}
 * String chat(@V("name") String name, @V("age") int age);
 * </pre>
 * <p>
 * This annotation is necessary only when the "-parameters" option is *not*
enabled during Java compilation.
 * If the "-parameters" option is enabled, parameter names can directly serve
as identifiers, eliminating
 * the need to define a value of @V annotation.
 * Example:
 * <pre>
 * {@code @UserMessage("Hello, my name is {{name}}. I am {{age}} years old.")}
 * String chat(@V String name, @V int age);
 * </pre>
 * <p>
 * When using Spring Boot, defining the value of this annotation is not
required.
 *
 * @see UserMessage
 * @see SystemMessage
 * @see dev.langchain4j.model.input.PromptTemplate
 */
@Target(PARAMETER)
@Retention(RUNTIME)
public @interface V {
    /**
     * Name of a variable (placeholder) in a prompt template.
     */
    String value();
}
```

langchain4j\src\main\java\dev\langchain4j\store\embedding\EmbeddingStoreIngestor.java

```
package dev.langchain4j.store.embedding;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.document.DocumentTransformer;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.data.segment.TextSegmentTransformer;
import dev.langchain4j.model.embedding.EmbeddingModel;
import java.util.List;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
/**
 * EmbeddingStoreIngestor is responsible for the ingestion of documents into an embedding store.
 * It manages the entire pipeline process, from splitting the documents into text segments,
 * generating embeddings for these segments using a provided embedding model, to finally
 * storing these embeddings into an embedding store.
 * Optionally, it can also transform documents before splitting them, which can be useful if you want
 * to clean your data, format it differently, etc.
 * Additionally, it can optionally transform segments after they have been split.
 */
public class EmbeddingStoreIngestor {
    private final DocumentTransformer documentTransformer;
    private final DocumentSplitter documentSplitter;
    private final TextSegmentTransformer textSegmentTransformer;
    private final EmbeddingModel embeddingModel;
    private final EmbeddingStore<TextSegment> embeddingStore;
    public EmbeddingStoreIngestor(DocumentTransformer documentTransformer,
                                  DocumentSplitter documentSplitter,
                                  TextSegmentTransformer textSegmentTransformer,
                                  EmbeddingModel embeddingModel,
                                  EmbeddingStore<TextSegment> embeddingStore) {
        this.documentTransformer = documentTransformer;
        this.documentSplitter = ensureNotNull(documentSplitter, "documentSplitter");
        this.textSegmentTransformer = textSegmentTransformer;
        this.embeddingModel = ensureNotNull(embeddingModel, "embeddingModel");
        this.embeddingStore = ensureNotNull(embeddingStore, "embeddingStore");
    }
    public void ingest(Document document) {
        ingest(singletonList(document));
    }
    public void ingest(Document... documents) {
        ingest(asList(documents));
    }
    public void ingest(List<Document> documents) {
        if (documentTransformer != null) {
            documents = documentTransformer.transformAll(documents);
        }
        List<TextSegment> segments = documentSplitter.splitAll(documents);
```

```
        if (textSegmentTransformer != null) {
            segments = textSegmentTransformer.transformAll(segments);
        }
        List<Embedding> embeddings =
embeddingModel.embedAll(segments).content();
        embeddingStore.addAll(embeddings, segments);
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private DocumentTransformer documentTransformer;
        private DocumentSplitter documentSplitter;
        private TextSegmentTransformer textSegmentTransformer;
        private EmbeddingModel embeddingModel;
        private EmbeddingStore<TextSegment> embeddingStore;
        public Builder documentTransformer(DocumentTransformer
documentTransformer) {
            this.documentTransformer = documentTransformer;
            return this;
        }
        public Builder documentSplitter(DocumentSplitter documentSplitter) {
            this.documentSplitter = documentSplitter;
            return this;
        }
        public Builder textSegmentTransformer(TextSegmentTransformer
textSegmentTransformer) {
            this.textSegmentTransformer = textSegmentTransformer;
            return this;
        }
        public Builder embeddingModel(EmbeddingModel embeddingModel) {
            this.embeddingModel = embeddingModel;
            return this;
        }
        public Builder embeddingStore(EmbeddingStore<TextSegment>
embeddingStore) {
            this.embeddingStore = embeddingStore;
            return this;
        }
        public EmbeddingStoreIngestor build() {
            return new EmbeddingStoreIngestor(
                    documentTransformer,
                    documentSplitter,
                    textSegmentTransformer,
                    embeddingModel,
                    embeddingStore
            );
        }
    }
}
```

langchain4j\src\main\java\dev\langchain4j\store\embedding\inmemory\InMemoryEmbeddingStore.java

```
package dev.langchain4j.store.embedding.inmemory;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import java.io.IOException;
import java.lang.reflect.Type;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Objects;
import java.util.PriorityQueue;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.nio.file.StandardOpenOption.CREATE;
import static java.nio.file.StandardOpenOption.TRUNCATE_EXISTING;
import static java.util.Comparator.comparingDouble;
/**
 * An {@link EmbeddingStore} that stores embeddings in memory.
 * <p>
 * Uses a brute force approach by iterating over all embeddings to find the
best matches.
 * <p>
 * This store can be persisted using the {@link #serializeToJson()} and
{@link #serializeToFile(Path)} methods.
 * <p>
 * It can also be recreated from JSON or a file using the {@link
#fromJson(String)} and {@link #fromFile(Path)} methods.
 *
 * @param <Embedded> The class of the object that has been embedded.
 *                   Typically, it is {@link
dev.langchain4j.data.segment.TextSegment}.
 */
public class InMemoryEmbeddingStore<Embedded> implements
EmbeddingStore<Embedded> {
    private static class Entry<Embedded> {
        String id;
        Embedding embedding;
        Embedded embedded;
        Entry(String id, Embedding embedding, Embedded embedded) {
            this.id = id;
            this.embedding = embedding;
            this.embedded = embedded;
        }
        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            Entry<?> that = (Entry<?>) o;
            return Objects.equals(this.id, that.id)
```

```
                    && Objects.equals(this.embedding, that.embedding)
                    && Objects.equals(this.embedded, that.embedded);
        }
        @Override
        public int hashCode() {
            return Objects.hash(id, embedding, embedded);
        }
    }
    private final List<Entry<Embedded>> entries = new ArrayList<>();
    @Override
    public String add(Embedding embedding) {
        String id = randomUUID();
        add(id, embedding);
        return id;
    }
    @Override
    public void add(String id, Embedding embedding) {
        add(id, embedding, null);
    }
    @Override
    public String add(Embedding embedding, Embedded embedded) {
        String id = randomUUID();
        add(id, embedding, embedded);
        return id;
    }
    private void add(String id, Embedding embedding, Embedded embedded) {
        entries.add(new Entry<>(id, embedding, embedded));
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings) {
        List<String> ids = new ArrayList<>();
        for (Embedding embedding : embeddings) {
            ids.add(add(embedding));
        }
        return ids;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings, List<Embedded>
embedded) {
        if (embeddings.size() != embedded.size()) {
            throw new IllegalArgumentException("The list of embeddings and
embedded must have the same size");
        }
        List<String> ids = new ArrayList<>();
        for (int i = 0; i < embeddings.size(); i++) {
            ids.add(add(embeddings.get(i), embedded.get(i)));
        }
        return ids;
    }
    @Override
    public List<EmbeddingMatch<Embedded>> findRelevant(Embedding
referenceEmbedding, int maxResults, double minScore) {
        Comparator<EmbeddingMatch<Embedded>> comparator =
comparingDouble(EmbeddingMatch::score);
        PriorityQueue<EmbeddingMatch<Embedded>> matches = new
PriorityQueue<>(comparator);
        for (Entry<Embedded> entry : entries) {
            double cosineSimilarity =
CosineSimilarity.between(entry.embedding, referenceEmbedding);
            double score =
RelevanceScore.fromCosineSimilarity(cosineSimilarity);
```

```
            if (score >= minScore) {
                matches.add(new EmbeddingMatch<>(score, entry.id,
entry.embedding, entry.embedded));
                if (matches.size() > maxResults) {
                    matches.poll();
                }
            }
        }
        List<EmbeddingMatch<Embedded>> result = new ArrayList<>(matches);
        result.sort(comparator);
        Collections.reverse(result);
        return result;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        InMemoryEmbeddingStore<?> that = (InMemoryEmbeddingStore<?>) o;
        return Objects.equals(this.entries, that.entries);
    }
    @Override
    public int hashCode() {
        return Objects.hash(entries);
    }
    public String serializeToJson() {
        return new Gson().toJson(this);
    }
    public void serializeToFile(Path filePath) {
        try {
            String json = serializeToJson();
            Files.write(filePath, json.getBytes(), CREATE, TRUNCATE_EXISTING);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    public void serializeToFile(String filePath) {
        serializeToFile(Paths.get(filePath));
    }
    public static InMemoryEmbeddingStore<TextSegment> fromJson(String json) {
        Type type = new TypeToken<InMemoryEmbeddingStore<TextSegment>>() {
        }.getType();
        return new Gson().fromJson(json, type);
    }
    public static InMemoryEmbeddingStore<TextSegment> fromFile(Path filePath)
{
        try {
            String json = new String(Files.readAllBytes(filePath));
            return fromJson(json);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    public static InMemoryEmbeddingStore<TextSegment> fromFile(String
filePath) {
        return fromFile(Paths.get(filePath));
    }
}
```

langchain4j\src\main\resources\opennlp\opennlp-en-ud-ewt-sentence-1.0-1.9.3.bin

BASE64:

UEsDBBQACAgIAIJOiFEAAAAAAAAAAAAAAATAAAAbWFuaWZlc3QucHJvcGVydGllc11Ou2rDMBTd9R
UumS1ky3LtgOiQuFOSFiI6Zbmxrx80vjKSXMjfVyRL6XbO4bw2bGNWTPbYJqJKRL1V+VaopDmbJBe5
YD20wbq7tgsS3RYerL157pFChwHbwM8RIrW4f1Dr3p8BhtbvRnCRofOaX17emHEwETpzX1A3PzHHDk
DDCgNqJPYRF06Hz/Qr+idLOuM1l+wINPXowx9ZPIsmGtJHywh+1AXWpYTiir2CXmWyq6SQUvZVCdeq
kort7LxYivb0BDPq/7ePDVs9GvuN1FCng1uRmWmOyzAvOivFayHrLFe1EuwXUEsHCOBgHTzmAAAANA
EAAFBLAwQUAAgICACCTohRAAAAAAAAAAAAAAAACgAAAHNlbnQubW9kZWxUegVYVM/3/
lICCnZ3K4qIIph4aAEpQVTsBRZYWXZxg7CxA8EO7EYsQgUVGbpEeuluEAUJC9T/zFz8fH9/n4e7u/
dOnDnxnvecK0tulZkdi8WSgXbWv3+yLBkRS4aPv8mxFBYt1ZqsyVLUWbBgsubkBSz5xUvxlSV/i9WP
r2sucOOzlHx0jSVCgSeHfHMSSrgi+k3sxuW7iliKXrp8jhdHyFLy0vUVSGYJOSx5H10JfeDE9eLyWP
18dPXt9G1YCj66Am8+qz+eKvAQCIUCb5YCX9eCI2Ype+mKhWxnvB6ZxWOL8XJ4d2u6rI+uSCzk4FH4
m4eAL3YTsVR9dNlOTgIhmcHzZfXz0hVy2DzyyeOIRCwlvq6VwFHg7MtSwTIIPDx5HDEHjyM7C9lcPt
kHL29KjtSPyMpxJr/XcYS+LHm+rr0dS5Gvu9ZNgM+JZRazfVjKeEO8sxB/HeCly/
HBAnmQFbH8U9dziNAuXKGIiijkeAqEYlZ/L11jV19PMZdNTywS8CRiroBPVnJyY/NdOSJyamsnscAR
HxKvY8oW0ROyhe74rFgsZ6IsRXJ6iYsL1QvHFR9yAF5AIOGLhVy8BBlGTq787yY9pKdAJOI68jjkUC
YcDo8Io+/BEXKdsDB4q/VkK3wMPkfo6kvWcBEInIn2vDl8MdGekURIdKTgRQ2GpeJwxW5YTCyxncST
nJ3M8mA7c8gtV45YTIZjIZy5IicJtgDWnx2X78Qh91wkQjpZiewjdMJSk9PiG/hIYqpkvNZCTU1tsp
aIuBI+ED63mOuBx2J7OAj6Rmj2fS7o+1zY96nV97mI7GtFnZE4n0DCc2Ys4ko0T/bHSuM7E/
fh8rEsHmxqEjzWkUN9gkroQTSg74r9RJ1qXcJ3ciNfLNlC/KU/fsbDYktc3chu9nxnshvWM7U/
XsFNwGhHzOa7k2l2AonYjeihzwJEz05sHo8awk1A/MlbQFZxkfCIwday3TnMM18y3Vsg5FGJ9V1csd
9wRWLGhqZcEVmUyxeJOWxnMtKEy+MsI6e0E+MAElFHEAqwG3hQe2DxudRczgKOiD+LyiriOlMTueFl
Bdh36FcPEYfnQp8KsHIGENfms/lOXOJm+MAiCRbel+xjJhKysXPhtQ25fGc62xMfg+NE44DxcjLFEA
MFe5aI2BSHbF9ISzw8GDN502jHZp5qRhzOmetM9jaQiNyIjy5epqVpY0nCUCKi/
jx7rRuHGAirle9M1zQUYAcjI1wkfLKdqcCb0wcclixbCrYTgnXyR2LrUgUy8V2JBZy4/
BZw3x0V3DchWxPDk+kxxGxuY4CEUuFr2uMl/ClEEciju2KEciDRgcJLy5fLCK3BcSxvfvwkO3sRb5h
B7HEG3B4OP7weURYXIxCbEeKABxGcwIXFy4ODhrc+PjEJRzxWuTTWcDnsAZgJ/
uHbywVPB7rlfoqtflaCUdEjoW1YUBn8XVXczie1BBsoYBaZD11fvzEUoDdEouq/98JFImPEm3ihTn/
O6Wcl64W0aIjl7oGT0DAmE+0ixdi5gipI62jgY2/
GHD6gNRSIKQasOW6umHFYLSxotIykGzOIfDlo7tKwhY6U9OxPSTYb/D2LmwnLo8rJkiGbY2/
iXlUK25sT08GXM2oxrGXE0/tR2JPJGANJsr24JJgYNTCRLkXl+NNNhS54cyiSDyQJAM8yZjNGNyJhC
Z5wOPgIMI3cHC508NyXaixsF2ogTlCEV4UL7VDgrWHA8mG7c5EHjGZG01GfF1bN+JeNK3xacibCdk7
yQg2j97w0l1FcZU+oMhvivUuFtBVeFwvBr/xUGpVNg/
nMBxF8l4kAeGws+N4ijkeJDvgfGq9mqgViyAU9yndgSNSJ/fYYjE+Hs5gymQfDo/
Np1tifbHpkXACxhKu5nqQcziR/UngKdDcQ2Sk2I9/
OnJdicbXYhgYQMLLg8MAsGKfwxMn4uBUKvDlcGgG5vC9uEIBn/
oU1aE7TYT2fGIQvBeWxFXCdqUa93QjGun/D4F5NOs4E9fHM9a6EajENxx5AppF1nE5Yj7bg5zSkCCl
EoUFgln4IfZYEdlBgSCyL1mcQRBsNzcJ31VIbxGXdidDPJg4sVtrbEO2J46F0x6NDAs2Exn/
dELzkj6fwgWmJ5QlUGfpS1EeEuxF2GldBVgE/
j8wWM0W7mJz6R5sIZs8x5hAIJSalwiA1aJEnZIEEsa5hWpEMCEGK/JDg0AWxjRfQpwGUsh2wW77X/
S4cDg0/JyFbBcxjQC+szoRW8jZiXFAzKEpzk5CgQ7f9iX3hBQesGY4HHcKX2IxFrcv12GEmUX9V0z0

Tg3Mxq5HE4aQI/
LEKEPPi7HQmSXH151DkgZfwKAEVpE1tiS+rclSxaeSYBqH96YAgk9DLtpkRxFXLGFCE6/E5zDuKfLF
snlQ4uPNJTlUTKBRzHFy4wuw6X0JdlLc7YsnIharnwjPF68k873ZXOr9BJbchFwRA0IisTp1bbaHJ5
vrSoOLbEjZFJtKRwLAmy0kH74c6kQ0aNwphv8v+WJzLFNjHNNXRAbzBDiviHSNrA3JopR7EoElZAJB
A2diO30+zQLUuQn+/
dOShLok1TxNzzwBzvrkidksKhBJdIzGOULiKyT39fkzTfnY1EZsLwpLLhjXcExa4pzIFmI/IKtYCjW
IqDTT4aHruTwG9znEjxSozshaZtg2FAd2CBzp2m5cHoeht31BRyxBcMSDIDFbSFOkNUPY6CjxLApTb
hS8CVNyIWivzKQuymj7FhNRuoQTrZhGOgZbEiOi/
z2fQuZjeCA+iy29ke3NduMKuQTecLAzuMtliKgHm+/
L1BjURUlawiQHD5yiRrxnPXVyPrnvIJBQOKUsTcnrH3/Gcw3ZfDaGZyyTvqeQS4MJH4uezw5XH9TbV
llbG613sFpGFMBxEvAFHlwnRp1YKRiJuTQfuEjEEiEFdEpL8POphIVgQdiEJmIYoQrFliRz8BOM3v8
yBpdKh83EY7SILdqfEECcpzVIwYDX4DvPFwipKXCNQTTriI3Zp1kbjhBjFi1p8Eln8egxaGQq/
wcBZKodCWBSRrkJKTWVJ/
UD8Wh9SipNOdiK2HWdOZjMcSkNIc/4TMUhFjPlm8jJTYDpA14GK8ypj9LjI3iwedQ/
XQX0pCQH05ATcZxwoSCmmZrmXDwRky/q+Jx5Hmym9nMiOYBgKpcvETO5DoOcq5ABeMoO8HwjLrWvq4
BsxBXT8lHkyXZifAl7rbOQQ0GfhhM5XR8+sx0xHlNBPdi7+ugaxwejvAhnWMoPCf2kfI8UMwrEzWjF
SRRBSC3eiENlxVKYCISMk1OqQOpJsrgCIYAC4i8meCnMftXJcq5cIY96LdYVh6INg299zsBUD4yYWL
p/AMqgvZDD8GGmaiTrcykHXiWgNZtQ4MFojkc1qIGloWDCFxHtkqNwiPa8+ggM/aTRyeZ5E+yiOZsM
JfkLE0AawZQWKRAfpHjkJvFgU4XyCWsi9QGTv2mFTBCk5sANRgjs8nIi3VlkMezHTNoTeP4P3L1p5r
R2Uyfejwt/eUJq6HdP+t2dmBDHpZBL2QAt+RlQcRQIaI3k7SbgUeZsTas/
njpZ0A47GZlOQ5BuLCIJUsOJh5Wlx+ELmeqE4+NEuDYWS56U/SQDO3M8MU9iyC6Ofokzl4YVRjdcqv
aVvmIuw1m5IpGEIWI8iROlDJ5ExV44U+CMoUIaDWKuF8NRKV8n6KfIRAh1r/
X6mFwwCuTTis2VmpxAEdOsIP6OM1lfyfB/
xCUVFk1rOEtzmCwl9qZBay1kDSa1JYZasYTPJvGqTqm0hKkf/oEwAQ8m+xDRaLQo9tVbpIax/leeqN
NFXcj+fA52VhHBeeyJtrhQIbhO2BAxBRlFq8vVbNpVoZSHAg5JiCJdN7EHpYoSqlgrLB3l1SKCjoQ7
etNKHUORL+0BeVJaRCJJienQENgRYY9kKBpHSCm9uYQBUIKpviyZBSyZhSwZLZbMIpaMNktGhwyhsC
mzmCCYiEOKBI8+BYoFLJklLJmlTMnTJw6b1AhYS2v7kgefEAmi3b72h4aGBtNwobzXynot84wG4FoB
ZS+E/dJwxT5py3HFOhKpMw0SxvqUAlBIZNPsZMZ3ouxRyGHA2JNK50VRzJh6qjzRBPnJlGgEMdkiJp
QJDcVIj5kAJu18SsrpgeVJOidT3PkCbw2iQxc2ztIEup3YnsQezgInMpohARRhnUnc6Vv23aXH5Qso
whC2hX+Z40IAr+SJSyARU/MwTRJPjsCTqVgJ+VIgoURhgTAA5qeYKKOvuKYB6vBfA4oCCtbuLMpQDS
gXIcbpq1LNZnkxNbvbf1UpKRb6EezxoTTFkcMj+YMW5zz6xLtvXxsJzW+4Oqf0GKMTX0RCktTCNLMS
TZHUzmVqOyYc8BY2bAkFT0dM9GiXCjugGCtPmYC6wL2vv0XMjgcJSSlCf9Kyj0dKA9LYJISOpUgp4M
qVZFVXUi0pU23SkhS7s6ZmX58G+xp1Fw+maiPdJC5DZMXkMATY6Wyv/2o9vKPZLGqPPl8ljU1PTyHH
icsmLIlYsa92xR4voQ6HkxcxkkTIfIhZcpgVEW8zEpKvGuSykFwWkMsiCtZEJyLyU4tcdMhFmxjEgM
ERLBmmutRZVxEtMTbzYJMhNoyj4BlALvpM1vLBwuFfRkzhTQNbkZAVQiTwfXNysaAm5xD4Ij+pT9LQ
xD/sKEun2qUMiDJ+BdoYIW7PpsKu72vSeBALYFfHFiBeZ23PyMe00AgRI4TNjXH2VQJqOJEHm6FKHq
T9gSOFLfKkzsBhmrR9gaT8X4uMjOWSrCTnhbU5AkegWOy5bP58b29vDQ/srK5cVy55NpPycQFVPqGk
eE0SnbQzyqFuSUZpkMs8YhSzteTrQnJZQC465KLNcAxKuxQoupK7QMXhsX37ujH6fW6FQQ3/
MiMXCzLcFzNe/
N2GHBdrh6iNlPL4lj0tL5hsj+GLqT+JgfEzNrlwiUlWcQQ4DxI1cykscanGcLbjUTE86T0Rw+
+EXpTbkDu0Jegi5FBwMGMKExGHfqw1NSawY2NJcooG84/aiCiU0fEO0hXEQ90ENE+70Rplg4Ud+djh
6Uo7YCRQsVu4Mi7oxXRS+5NGAwZQVxE9jojyMj5DxPF62D7ktsCL6cKu51DygI/N0aCck8MR9/FEYy
tbaytmfYGEHtKDBqC+lRExBy4fMfUg22FvZLAfm89KwFAoggd4xrx5tIkgEHniUonWW94MomH8IfSp
H59pKiqR/qyrG8+37w5NphTo8ZqWlCFZ0nTD0M2+djIp1P8jLuSHCylySBOW1IIC5o3BTgmbRyg3mU

```
zNx8eMwYvkA5X/67TU9AR9SHj1FcGEudMqV0ODsSVmlQMIpGJ90z1JpsQD+5NdsLB9jbW+6KRBxqZq
WKBG7cXhUALoxuF5EqKE85eYLRGSGkbJ61+Lk4zWIPrwZaqXvt4M5UdujIpo18W5r9okIIhV7CgRYe
GZ7h31ZCX6IqfvBRHWKJehPJjSUu/09CD3qcIoP5I4E5xUJYAj1vhXweAJQgo5VHiGd5MMha1hb0dW
0dIgVxEtQShJoO1wyg05LkzfgDJw2njwpm1knLiN/2N05P2FJ4d2yomrs6mh7Ri/
ZzszrzdoRFHvxqIwNSkDuKK+ioOUvBR4LYz17YwZDCA+pkCcnoYqk+KwcKSml/chtqdmYMgHj1hdgX
ZsKBul0SSH6RVTf2LF4R+zaVuUEAR8YEyiiDUwjBOo4euqkZuLNGhz8V8s4Nsa5DKPXBaQC20W973X
6XNgJfoWSiii5RihH9TQGOglNCHiSUAu+n02FNHo/s/VFOh2ZIAZLa8kvjQjER5O/
cCUZPoBDAR409NiMb37zsxY05tN57OJsghJ7d/X4RYwLSsuM0ZEwIrp5TKvGslPMcEunEWobkknaDa
9TUGHLEFMbsT2Yt5O0NqAqhM7bp9IjJ6ov7KpydlMm4SEB+kuMWcWUtsQmk+8QYKrbGd6Ao6rq6iv9
+NO3YtJxANI59PFhSPsa7IarzO2JXI5UgBYZsTU9kKmp/Cvt0v81JtD/UmZUisux4tWV/r/
rEkckitiPjzoeszbAB/SgeHwnfockStimohMKiWsicKSFy6W2fT1og/Dh/
rR5gR1SZIaaVPMF9uCFoTOAoLxpIKlazDdMcJ6/usLUTa+isQrZT+k2cw0lSkJpbW2UMDMp7lWiVT/
TrgQIzpYsYK29UgXiIklDZYsX5emcz52bNJbotYgriFPijOmfvqXe4k3KBOawsQl0QjT+TL9r0bHqI
ZZCRaGvgzAWxIe4kOztmJfMU+M+a+334/JQrR7JKZXWoL0MUtKZ8hsDu0Vcel4+pqAvhZ35fx7VUo6
sgJKaQzZfKbkEHN55DyO1PENGBAh8UzaB4yJsBo0iLZxgqcUg4FB0umhoEehQZnMdWKTF0DkAfWCPh
rBETuRaX2UT4GCNkt2gSbdiRxTdsECIsF0HfxtIf7Twn+L8J82/
tOhTxbgb4vx3xL8t5QlK9KlLRrSJaOknsuQNtK0k12oif/
wKgvxKgsX9fEEpz7QtyB8FCsI52jZhUuJpEYCSvcI1RARyBvwP+igQUrTGaECfJxTZEV8/
OdJKx2msGE6RRJqEQ96OFPm/x/
YcZh6nsdm0pWBL6Mqdxo9TJ+D1ngaTrS65WOiwJL10qUJhx4Mn5tLk6edGw3E/5KXvBfN0aTJQL/
TNzmMJmjSJPdcmFf9rpQo/UsP8iT/kCuftgNMuSxZH13yG9sCH6mPwZF+gYDWlH0nI15LrmYmfZ7JF
TLdNJrv3bnO1BPJG0lFn3+46sW8FcP3+1htH6G26Hud44Gxk/
YcqIKot1hx6Os1RmOKTPlJg0tEE4mINFIpTGos0tYh/
Xs3sQflHWZMz4rLvLM0oK1LLwZCyFMmBTAUxYxSJQ/
mOKK+9juH7oL9RrGvb0P6UlOZupzX12sU0WD815Ams13IKDVaFYiJ/+Ff6rSNSdlCP+Y/
GpDdNFyJSgnC4CHaNIdhUsxlBnn2WVrDmki5UHMhM5aKqbGOrM5wC/
LTjixgxpSzHKZxTLOZAiGUXszLeF9HDpPxaNtTw5CptRlVc/g0nTJ3aCcNIxleks2UzMzLFPL/
Wki+JL1T5l0Am2Acgfl/bwRp54bhy9YE1Ek0CDH04UkcgYipGnEck8wm6PsfAfT1M31KGTytwuW9+t
Kss4C4CHkIRLOWlDppUKPSVyb9KM7TVzXMez6mE0k7ZbRBxjQrfUmu6OfDlCYK9DXPv9aaL0PFMZ1y
Jjjmw/SdiZdTZXDpq/kp//2jjBWHubII+56E/t8S/
PUf38O1uRq1NOnH0P6JMenzLiPw3lekKPa14snydEGcBf8PicbLk+JkCIbv/zUdaTdWnoyndAIPw7W
qMVGFIa1tbchlLbnQeoysiiVhUscUIONMKDObbcL2USPDPEnZtZYA879jkZYXiRk83ZNxP2JtcnZ9W
0NTMwN9C6NllMhTO2hMpVc1mmYEtKtOzTDw/6qKkdaI40SI0BRyUWf1F+n+V9sSgBPRXgyjkP7/
m0x7ijyaaUS6JKRJPc2nGUvjn9qV/g0nxzDBRSuXoVEAdNh/FLxP8wS9/1dX4/
UIW1Pw+RdXmIkSZv0/0fsx82jbTOzRh1qeLFXR/
zUMCZGF1CUpMOiT3Zjp+JchAy5i8t2aXGiQriMXOtqVXLjk4k4uHizV/097VAJ8aCjy/TpnaIIz5Jr
FJjSW3obMKVd+HI8fA7nq7FEn4uwgZ/2hvbadMyBfO2fSuepXUL+f81rS9QHyf84RKFQPhnIFtxD24
BzIP7H079sQKeScMryzLuYMlM50CB9zMAsKruhM2Bx5Hppsfu7/OlUHCiObbS/vPQmZeWbtUTueQu6
TDxayrH6Q/1T3zyNNZWgpu5M5ZUc5lIUa6R3fMQeKs7PiAxPf4nkLus+y1SFruOGY2/
HzobDiwzIr927IPWm0+0DlTCiJ/SUtjH0AdcutjDN/sKFgbOvM724hUNWx9Hx/xxGQ1fKRf36cFuRY
maaKRPi+4cdVCW2OUJhqvvTJCGuQTrzZw3JaAtIDBxa3vxgPad9Vpy6duRSyzv4avMA7CprPjl/Gnf
wY8hdb9wZYmkAW2mYZmX0Jiu5qVyyutIWPVnWqEbZdUK+4ZPnvmDTI9J977O+DUZDZdHLgiqFnIc9v
2ZCU2GTI7Z1h6qjqDWXTrkUah+6CwjTnmhG/
sR0WHCu+XXsGcuS2HF1Xn4blyHBRbdgL+T4qi6ev6oGipREq9u9GQv6ZhXmhmsPho9mG4WN/h0FWSv
```

```
rY81n9IUfxvGagszwUPv+ullxgC9mnBWNSEh2gaPeVvbV6fChbO8n/7hUtKNC5UjoirQfyZ4g3i4bZ
QqHFN/eTM/H5ndTVhyoNhCwlB0FLfgNk3SvLNBveAfm99tZbVIdD0fsy9vQ5g6Hg3uBXOexoqLD/
KVj+1QZK3j6UdbXJhHLeGO5USx4UVg4/GTcvAiq/997orpsO2eO/ptgNw+tdnX/Puy4QKrc4p3NSXk
HBjlInbdkkKPFJ+7R94l3IWb12K+fiG8i5snrr4SvaUGV3dvOSP3VQrZ9Vmt+uDtLnk9dOeZ4JBQcv
JMxcOhsKTXhRBzcDSA0CvJKv3oEy/zG8S2McId+pdPjoWkUotTK5tNj2FeSFXYzKMD8K+cNHbb+/
qQekMasSTm/SgEa/pxYeLYNBGjRIY++qGKi0tzO8fD0f8r5+XGPtnw0F21u9UifFwafS5A2//
TqhLIdnK7W5Bjn6iUu/a7Mh7+3AmAN3TKB096l5r196QJU/d/CjonwoCJ/
kopCbBtnG9x2G3TeHPM+ooiavMCgYsDPC8ehZKNuaPqjFexPkGc09trtWFrLLV37bMkMD+
+UXvUfvz0P2fY/e6fYckKZnxnea+oG0S+fSoXNykDK2KuAUNwCary29fuHbKaiK26sboHwC8o84TNa
wmQktr+SKXlwzhdrNh3tGDDSA0osCC9Oo7VB3RfFJeflAqAvuyKhqk4VC9QHKlYO+QfWildzaiyFQs
LNaonNmL1T0G1iXHZAJ+Wu2a7n8OQVl9WdUZIcrQv6PfW2vVuVAqfEsk4uuOlC6/vL8YycsoOjl3CW
q6VugUDCt9rWxKWQv2+AW8loAxa15B8fNegDS8Adz2i07oGjIO5+0jTMg12bptYapNVAoczEiXboIy
r3SPWTQGZCeUo4ZEiGAAtnj+8bO/gGls99cnLnHBPK2HY8bVlwLte8NT7TufQLSBzufT/
+jCtJFxg0nvj2E+slz2dzjrpAX+fHoW4wjUge/T/
wnFVDk7hzk5mIFZYuL23JuV0FxgtOEM1diQbqQE5v/
NwFy9UpMPvfKg3SdwecvGnegYJjboTZbIeSLyx5usZsG+QvdzkyvrYLC/
ftGfFVNgTzj0Orrs9KgQKpqVqK3HipeZVwPeLsdikMe9vwsPA/SQsW6+inFUOUexlpsdRCkxYVfxx/
LgnLN8U2FlsuhaOUMjVjvbVB5OvZGovkhKNrVZNhzXgOKZ3w9/evWeSj+OPpEa4sJ5E5sdhk48B2UW
Ey+tWVoPVSO6bxYXMCDAsdg5If9oOpK7+yv2iNBesTgh0zzJcipdUy+tFATcgdu+GoeOhqa0oeMlRl
zHj4MVLoeZSkD0pZTFvB1H5QFBEyJ+CgDHxLmcFK23ICaPV/
Up89thhLhRBW7p1ehKuXyI7tFmiBNtXt0fPNCqFwW/kCl5xZUqA7J+tH6B3Kt6/epeb+D9BqrxMx0S
8h6MtfV4WQMlLjXr/4lToaq74mPv916h/0rYPS0IBtovc17xvdQhZL4q5mtt7D9DSf98r7TA7kChUU
9v69C7u0vqualAPmKL967vWFD9u7yOWO2R+C4aGGzNdIhx2Zn8NKIemg/O3Upi4394GdP/
UhVM6iyemzJ9RuB42hK8Z5TeN3YgxPqM09Dzpllhbcmj IPcxucqN1oiIX/wweOHvESQu7/JOH7eIqi
+Vdvs92A75JmwusZ8nggpXUOfWiQ9hUz2lREXl+K8p9F6d7LPUyiq+bL7yvD7IL2kPpbrdAjyr8lGy
20JgcL13O+hH9ZAwfg4RYn7SMjqd6O/r8QRur4pHTi2ZgNk5yqPvGUXA9k7m1Yo23VBduFYs8+IC3l
n5usn66pB72BLWdnnbiBdPvLkq3XnoPnc/j9bzf9Axq1RB+aWF0DF6XOqqiO6IV+jQPvHkSooyB37b
eHIDVA1y8owcNpNqBk0vc1b9TJUKFZNPzgkCapPKRR99rkM+eoVafFrzkLNcL9b8+/dh+xUTtWr1Kl
Qo3aw37EZypAzLvH19j3HoWrdsLC7S15Agb1+7PNXAyBvx97aq4ecIPfDPfO2y+chrUyh+IKTDI6TS
csCVm6CD6VGFz7YO4H0qN6O3glu2E7PXgRdvQ25pd3cRXfmQV79/
P3ynvmQ/0rO9Q0aDFUHVDnDExshE+1tfBNsAB9DJvvMMuqFcj/
P8MQFi6Hq4oMXO7nYj4JkXuYvXgKlRtEDbyZNgiq/4TMaPHoh73u31uHoy/DrvXR4/EwjqGbFdDq98
Ie8G3MbJXtaIed2wJD9JSGQ89f+3rDg0ZB3ctn6yLJhkC8IdByf0AKV+XtZghUHcNx+CWftewO//Oc
eDO9Kh6Iz727bThoGxR5lR9dPsYe8KcX6LgOuQbFb5kbhRH+Q2k1nJxg0QIbrr0W5PZVQdmPLQVM7j
IOl6fMExr4g7fS6nKJ/FHLNh23r1jWGup5h0tBkP4wT2Xb5mXlQ5CRjMZA1Gv6s+xnRcHQhFAwOz7h
cZwvVexR+L1oiBwXmC4Z3nKyE7EVjzH7GNELxEhg+Uz4fUvPYK5TCRkGeyuFIl+EYd8aoSLfZb4fKO
zcyxPKmkM+y7je5ZDo0yB9VfmZ+FXISaytch4+BEulZxzs3f0Pe0bIB6hZjoJprN+P2aAMo7s3/
OMpeC7IjP30ODeuFAmWJ1uIZm6Eo0HmZfLID/JJO9Owni/PEi+K6SW+qoOxl0oTWsj2QfH6fdWi5De
ZXSt8rMtshrzNnqumwQyD1WNpun5sIBTYJOsYSHyh4cD32aM1cqA7Uf/
zJ1Q6KewRphxdNg9wrfh9NplpD+Zkfm78/eQDFbVlN6lY4f3bsUtmoEwpSP27B7SZfnJcPT+HNwfrc
dPjVOYtIaL4+cOxrzCek3Z2fxbzxkGuoJ619NhtKdOJSEm/5QN3bV6Ve/
ZdCw2p/90nXxkLx8oQF66YMgLryUS/
Zm6ogb1KmuZF2MFTpD9FcNWITxtGs5uBT9vD7R4CimWwplLi211g/
l4X2qdH+B5cNBelwGHRs9BfIdUiTjj85G/K3rlWOq74NH0fJHRdkvIKM2LiaXXmboHtJ06sp8ieg3v
```

OpguonrIch+3d3RR+HoiWnC5SqX0AZd2KNcS7Oq2snfz+7dyWW+3BA5d7lUH5u9Wi+Xx402e1sMHLv
hcLQQzFCJW8olh2bWr6vBPL36wzcebsQcmPDggfqGkC+8cHtU5YEQpXal7N/
k1dBeVBgw3ajIii+KGpf1vYUPq/c8OL60btQPbLy7IDv1lCkflxR7/UGKH7YefjwjwNQtL7U+
+Phbsy/OjIm/3oH2ZZ3TJbrLgPppyfBOmO/gDTSNSz7WDKkLmqo4J0fhZ+HPbkYfwPyPNpDi+6PgNz
m7FdX9NyhJnhtR9EDFyh2KPq2fH4gpJ8qvK29qwqK4o0TZL93gXRyqmyNhi1kvjTMa9wWA7m2r5O2F
V6CsugTrlsNC6Dh0eTnZmavoXRAjtPKzgDIT530TSulE9KS96Tfr5dCw3eW92P/
BZBTN9LBPSgQPr64BmVNBZDxelg/92ln8frZU1x250NOhP7OO/b7IDel/mrwCjOo3bq3cun0TMhdeC
ck8N1o+Mh7myzoVYLyOaeT0m7MgdxP2u0dMkpQEGvuNm2eCRRtGHFFX28g5oPizg87Z0FWuWdb0D4V
kO5xN/N9tg1yis9c0Yu5AeWWC+Fj2XvId5WZ+0RiAXnswPIKhwIou73aXm13KuTNEKrIKJZC6ZCxlx
ymDoUynu7LbQ88cP4YY+LcEwrZXg8fJv44DdIt3xpuLO6AbKVv7in1t6HS2H70kyl3oHj2s+TUw85Q
uC93vULtWKjNb93+yLMfzmejf68/ZIjxaZRZ21FzaN3E3mLcPR0KLm/Zp6TTDdl2A2M/
D10HhfaHXy6USwDpiWCtgTrfIK98yM0bRauhLXJwm+aMeMhRmC+MGTMXvhfcMpFZ3wC5R4Y/PvPkGX
wcus35m04X5HrKPM3TGAj5yXJ1838B5IimiOu3t0I9kskMVrsApYdG3R+XNwRK3Tidg9FZyDGfYet3
Kwpyu68fL894jvlST/W7DDbkVB7r3bS3AMr35I0a8n4ltATkOlX3qkOLmkPBlHARSM8/
KO3Sdf3P33NfHVqmoi+FwoDOqHVv1CDni8SWExn6H1/KGKDQNCtgExTWqDsM0siCXM5kaflrI6iwnR
xWvnIiFJ08vkIr8SAU+Vhs+iF3DAqf2S93+j4HCuIbK/3hPbQb7D3RdbgCSuVzno4WLYGCY23Pp5w/
AnVnTDcG9xbDxwi9ttnVOZBk/y3Y4cY0KNTQSnzvOg4aZn2/M9B3KWSHzHOFpRFQ5HjySG3GSCjqSA
gPuYvjzdO5yeDzBfhyqiNt0YhYSE91b1ZRvg95FySun2M6IXdq+aGb965B+xWdLk/
ziZBlF3NqlfEXKLAsteFvx/nQTuHW4RzMr47Ok1c/5gnFFQfef97YC0XT3v40SsI85fuzoe9ep0Jx/
9NLZ8oFoYYpq6rdfC7Cr+uVulrcQMi9J5Kodd+LlQsu9TK8JoMK1G7utNMYD6WG2+ZduzwXbPPO7Yc
FXhCnGXJk8fpklKYxLHTr8Uvw63jJ1n6jEyHUaLbPZ7ksFO9XN8yn9yoKeXrij+ybQKjz0GtwXLoD3
rF2mQ7iG6JalU52q7ARXRTmKkRWNkJJxcnehMnfUbXeuea5b/TgU2XersD+xahN/0CZ/1g9aLj1fH5
wfiDEhnxOKTjaHCs7tFcSb2UHsbnmt9OeHkBN03Z2Dn1pB9HiusiF2UdQXt72LAkrE7U9Lk8e5KYHl
bDb/HjLU1S1Muv4Mu4NaM+Qr7O0f4ASC2zeO21cC017l66Y4PARLp1XG1+dMRo+FO/
uFjppQOiDjIbQs2PRO+viooiYIAg3KJjWcXgDik5P+LP24luoUIt3Wda5FOVwPq1qwPzp5bg5/
qfs7NFH9osY3+mbUeQ91t/hndpQE+Hx0f7LWsi607lP6dRK9FXwZ0WL8jh4PvTboiNy6ejRnEtGCke
KIawy4fe7d90o9udSuUKWHWqx/SNpXawHbRtHeU5aoAwpgStWuEYeQF8sLu15ezgfYnR1lw6ytEXtn
rr3DsofQ82sEyHWVvugUfPkzJ6yGvTu9schkgwP+NSxY3TJOUCfkq2ye/
liqDgi+uH90Aw9yC7fozfpMORZR5frB85Ep0M7gx7Nno356/
Yi3eHq8PZWlfYN9VhU4Zs7JWjuUXh5aLiFnOoPlP0o/+OIPAThjtfVC1vNUdqNIut4lgLE9CuqfCDz
ETW3SseGOkejkJLJmtlhK6G9wv60ySYfuM47M67AxBuei+V/ti8Rocr7i9tGL2/SU9gdNGfLn2Go02
FKkIJaIrTMCHDMyhsMwV0lWR/7qYPRPV5+8p4D6Mdn9v4dH72g69ZATY+EXLgxvKtzvl8+ZC6ykR/4
6xWqKu9Ken8qHH6Wzt3jqvgSok4nxc8dZo+qtHVGau2ZFyvjfG3/6J9ToeLooZDpbcehveX78eP7/
FBEcGB08Z9RkHnRW2VAyB/Upfb6fNwheaiKefMo0ycplpXtY7Xo0iJIujE5XdYU2+9j3oUfx0dAxMx
Zz5vXzEYJ8xfx38+7jCKGDut349IgSHmgH/
bmwDyIWROelRAyChWOywesOIhN17pQ+pyHWlYO5LUlV0BJU/8O//vpqCvJ1OTPg+EQUWc97PWhNpS9
7dsQXtQaiM0wnfV8xQdUeWR868n0XMjaKPEfnchHLc/3J15+F44uR2p2znpcDl+9LDfvSdyLaksfHZ
9chfP02K27/N67QXxmmUpr3irU+3tHumn5eXj85fnW/NPGaE+9drtfcC0KSdTqSEq5BAVx7MPBoU7o
vPVhcd0sP0i5OnjYMpn98CTz1cWfjVcB8fVHxe14DQ9+BY07u90LiuMWV5vfGQcP38QX2kbtgVf757
pODciAaMOFagMC8tCXfUF75DfMg/c3b35Lf6eNvp4YmJp2SAayYTx7lWBvrIKR+0St7ROQ8Jy2eW71
YUDlKv6jwAu9Yq3TmD/5AzSIn8RxJApI8HJ0mqdGJaSHfPvWfngeqmm5033vbg18X7HVaJT2dHi0sc
ff/9xMqNcYqhuQtAyiQox5WybfRJU5+86rrW+B17ePrr/7h42KPzZtvav1Eq612jZ7RQVArHfw/
fjprRDJdV0gUCpBNScaP62R2QRBcRpLZL8UwtcXbssmluij0L8POs3OF0Ct/
bkPl9avRpkrtR+nXKiEzuDICVZBpih5+8dHCk110DXxXO7RYUMh6Ylt1OBNcejr+q2/b9xsQMk/

```
xNG+eZjvXjRf+f1gFEodEbvD5fIN+LpqZddOX2O4lMZ/FJwzHIon1twS/
WpAr15vPKWp7w7Fu640PneORMgnlXUgOw5a9lhnLK5JhYhTJQINiSHK+bAs6/
TpUEg5fffEq+m2qPF6fez+iSbwyq5u7rbgUSi751tIidY8VDojfljhkE5o+TOuJcO/
HaUYTbpasPcDtL99cvGzYwM8KZGGRdz1Qs+dHfopbFKA9yV6ox0eSFFDinL8+dw6lKB/KmvaSQk0O3
6NWqU3HELOXf6dEDIUXu8LKzI1i0SFObNuGiUtgVaFmbd4U0pRcFJzZmf+KChci34+OfcJdf2M0hmh
cAlKai1sTwVeQCXy2a79cN1W39CxIVf5IMpO5hvcuP4VamIFS34r+UPCi00vRu/
pRt9vbJf5lhMK6aZb3LXXTEA/NL6asRZNRS/HNgy8GPkIaleYWy0tGIKy1b3a99Yu1pMJG3qeF+eAM
rq7bn26dAt+WNUOqv5RBHHSz0csLy1GVbGXm1K3ZqMTRy8/1bD6CA0b9jU7WyyC+Lvml9zNp6LOzsC
e4yePQty6gXJjbIeg0icr9PVmDEBxzgEHCq+vgqpmhROJ0p3wqLFb1jtuA9ysbG4QL7CEGyZz9dbs6
wfv1N+6L6mWoPeHRAb8hEPwPtJu3Nyub7EqBttNZI3l4NMomTXDQ8NRp/Gr/
gGXSqFS23Sed11NrExkPCe/
Yj903En48tZpG+puyknO7H8F2toWfA8L84lVVgyJf3qzBpp4ekljCovRzurrH22tj0HL/
f0d8x95xso15Ax+fNYG2pc7hqYrBKHfeXHBm6oyoObg7vFdRyNQl1bYgtlvpPDJ5Iuuq9EMmJx/aWV
TVymkbOPZvf29EH2e/1k3V4jrKJcW16CIGHjHzgu6fCYYlR3oGP+tPhMS1l2N/6kRhmps3mrsFPlDT
GJTNOtcOWpwTUvNPvcKFf44l1+5HvPcSN+IV3OE6KfWa7S5Xyx0by4o7dyvhz48fe8g9/QodLn/srD
6lROrdOGH78pvLdDuNsNPW6UMIpbYGqYM3Yzedw4IbayaDzm5m2631v1FX3hDO5pDNWLlRiwZUt4RC
J991tTlVxWjt7fzU2JUR0O04lyHvXFJ0NQ7f+CLznZUuuq7UiMSwad3ISGVEe2oUc9gb7T0BLxsmT3
0hFErylb75F1evzZWtuxq113RcKi+ZzF7xJ5QiN/ecHhfjz7quuMyt2KZOQR0mLCmhMrDHc1f2XkN+
fDUY/7Y5185cL+Gvd50kgHsXbTalsW1QeYH61wbo2+iT80H9gZmmEONwzlphYsnRFs7BeqnyqKeqwc
c+EPuQq1S86bxoffR12Gq9vuP3oREvVDlB4bJqErTpfMy3i/
WzsuhqjAGNShnj9k85CdErgxzVTpWicoStk/
QPPsQ3ujnrlbumoVqq+3Pzl3rCbGi4LnGPcGovFIyMS3lBkT7Da/
vjn+BSjNd21+dMkc5V+sy78Xtgjab3tu/Q2zg9bnB2/bn70Dx9rYJ03np6NWYdR33eFZQfXTlnW3fP
eFg+zzX5ePjoSGj461l3RyI7djlW3TRGtX5WYYmTslHz0tWZis9PAFVhpvPGZ48DOnBsSNDNNagQsO
cbnVPR5Qsb58efUIV2rvX/QhaogNSw7hF/fQ/xCqcPDL5xnxFdC3tp8aM8LlQNeuypv64DhQuXRtc6
hgLtU0aw96qPoao2Kvb6nmyqLS6Zu+RB13Q3t6gsskzDgXtPf57rqwR+q55ZU2V/
U+okum4sbi+ADk5CNY7LOFBcN2tOl9cB0lDvM7c+lqG2p/
MHy2ReQjxn3fem877hj5x83ONH3qj5hvn7lV+qYX22nNrTV6Fo/
ycreO0nspC5YOYsazruSg3ZdSiM/oJenKtW/WDokwheVnIoJ9nHFH5mHjDoRa+KHPV6Jj6Feug/
LNXUW7hQdQ0eX7phydLoVrmtVaOohtKGTxdM+LCO/
hxUWvgs+r1kMSZebXUIhl16rwa3FpiDR+sTU//
juxAn8x4k17aVcFd0+q6hfFacHPNuOXL1xWi3sObUsIGB0O93DjdFUGP4NDft25jfTZDeqx9p7vTX/
TdMnQve9xz+JxpExv/AFCD0e06PWEe1D0rFpfoBsfKfDMtv/
ZnKHQ2Zwx5G56Dfu9UfRUowPatWbZnQ/kxCE34Myfyxyu47W94yc2lACJD9H9PDD2FKi38/
xqZdKLLa1BclIEhNHv/zYpufAoRTfePTlIVovTzb6OO635C0tI1Uwyv+UCdJ2hoRExE0U2X9n+77gU
1QefHJugoofaSBdojDPZDw83k/JTM9+ijprr02/
oeKPO0Gmxh1wHvr2gFrnYIRSWsuMVys+egeg9DzjtNf/hx5/4TucgTqHvql9qOxAOQf+lhbHn4B5T1
VCTevNkPvtf9ueHM1UIHh99wW78sEq5vvmxeqxmAYtdLf6VFT4JmbZfpnQX3UNWIdw2H9tyFTxbNvf
OulaG03xujxsXehcYl+wbcEE2HeHidHcZZj5qHblS56eIAhXfzBlUr3kRVW8Ob3oqPQsKJYQbsYa4Q
NstRvfd8I+bTHwba+k+MZTl+uLXBsg3FXz78pFVlNHTPUO85pXEKbv6KOzyVPRud3x6xpnTcUpSyZl
ngu3WPofC038h5Q3SgdnRYnLHCBNS6+1dV1NpMlPts0iP/
F4MhpzZZQXCmBcVs+sLqkJ0LdZVR5VM2/0Uf7/soG18aivmsqp1r4yVUtNOTJbdxH3REtL1NTl2PCn
dOs1X78xhKurvDD+wrh0cqvJX292ei0E3tw8+usUcFip8Xjts3Hwon6MX0bkhCP26/MnjuXQpNoxuG
6nuoQViU4FOKxR6UtmLVrvlyHyFVOkP4qPor6n66c6ZKxVtUPr72xurDEVAdVnDrzPiXKFrh5L1Gt3
```

```
ZoWe7/QOr7F5VFRjyJf/wNKpIX7W/fKEZhf5YPHeXcHzoWshta5qlC+Kqtxq/
qklGo1oYNtZgfNegsmXdIOhK+K50L8cs9iaKmzBo90GI7vD/pZ7GgSYDySi13Pwg1gy6LiUkhiWmo7
fmF0QtXZEDLuQ3XYtnPYPBpcVXeuGzIaVPPQc4qkDHUyCZbpQj9mrS2502RHcrwCpqu3dMNv2LN7Ga
VtqHifWd36chNgvyIzCtjEl+ggyN2i3hTSyDvk9GnheaL0NeKw9eDLl2H1lsZAdlf4+Gy6Vvz/
mNmQHB01uKL2xPQ+6q789pDJkIZl3tihKwYPci5UyiTyoW84TUlg3e8RB12CWhErSp0xZxb/
GbWmVilkZvqxjuGwWdlw3e/
LF1RQ17NqadTt0K9rNxZh+AA1BXlXLVvrRx8nec1J2iEDupU9KttnK6lJ/91z/
XnxYqoTDZVOGdnpp4cf+Wp2rZlKCJ5o8294zVQOz3bs/
hoL9wbuWHhnKTBKDjqXojOwqdQGDjdrk6nFHUu442UbPRFddtTlS5k+kJVm/UZ/36X0Pt2R/
nAftaQHeJR88VMBlWGjyvKL66A/
PHtHk80GlFr5HTPiz+V4KffdUPzCTtRkpdSHuttFvzQqN8+MjgFQk97dOqNTEbvt7Z9neW/
OlY5ZeuisJ2noaBs8/z2d48gt6c4oWieCfqWHN6t0XEBfX59O/7btdGY784YobMrC3XUbny2Zu1KqE
yx3a3lIkS9yFh25IWl0BAxVT9yzFYk9ZbftzJqNfzxSX97RGMQBrDxKleDdqDPfhuvBd53geTFnIzp
Kc/Qp4+vmhuMzVDgn56Rii2HoM61aNJd/wA4PEr2cI+qHiSPuu855tEJeGO5cVizaSMqaRSfvxm5Eo
Va1saanu0HVR2u15+vHo+iz3WJxGbzITtn65fuGetRV2OH9hG9Sii7dsSOdUQJ3d1lVLTVxgA6/
RUmvispRmhx6BTnQ9Ogof3nyewgY3h31uDL04fzUfPJM89+Kc6Ht/uH+YvuXUD1Q4P6t/2eBy+zpEb
Sva9RXmfG+UBvJ7jBTtIylV+Hri2MnGNsZg0vP99T2mV1C6XHiF8k7F0B710PDtb4NRP1dtrd3VLLR
gV3xr9h3Zmkx/rkF6GhcgL+FNW7mFR+QO0VX9N+G7LQwwXqPUUHxFC93HSzm+wSSIk/
eLpIzwk1plsuOvBwCPo6O1fxzKXdUKOTJ17iowwvj/mv2zvcAOX/vVKg/
PQsPPqrM8tifxl6pj4uhLPSCaLHVfzsX/UV+8FrweCUexBx/JZHUT6gnLEnuZIZc6FgsUL3iuuK6Lf
avXlrDpuhz8MjlMqaiqHm+T2laOFBFHt49smIsiyIX37r9M6rHyFtS8+WW5ZL0Le/
BZ2pl2sgKqygwmeMHipeHf3m5eOREJe3x+B4igmqHna8u/JUP/
RLw+JN61YHqFYcdedb2CN4ceKbiYa8DUrY41Q83P02JGw3ePi1qx+qPDn/z47e++iZdJ/
bToVi+P5szH6RahKuH1Z/
nrj7E9T3TPFa+ek6ujV33rwfyaZQPeF1yFzLpahSdeO9ov0r4ZNpZ/8f9n5wSmZDzQPNK/A86e/
bwls7wLXT0a9rwA1IG+SWrB7YgBpux79V/VYANfN2zldbkg4xXWu0Nzs5oGJJr9KjE6mQrpK/a0d/
Map3f1F7OTsA0nM/s8o31KLux9uP31njAu88l/1evFIf/UwqurpsYyLK1JOObM0+DN27i/f5BflCmN
q0kjVpHihBeXTx1AVLUdzTrualKROgyiYyUbDoIUyqtQ0snaACWai2wjs0GRVllbarvXOCVrs8L9Up
qRDDWqGs7YxQTff5wO0vyiC43Tfg2n5NdFdi9bzW+gWq/7mksfURgJTjo95j24vipk+1tQvdCJ+6Fb
Y3zNwJeY6Hf1gcc0ENM9Ynnp2E1zMN5cmnrUbt5kNbBq48Git7jD9OZ2A7ND33rOw9XQkfJvQ/
dk75J2pYxItYVr0gVmamWqZp53UoC+s+1NVvHY6bdbYJbwzR92MjG/sdbEd34/
afWnFtHsRnBavWGmnFjljZJHZxPgbSuj3qUPsCpEu/pKycEYLyXi0OtJx2FGXcuXJi0R85KLV8dF1h
yxEoc7xdeu9zE2obsfNv3eMdkJh6udVy6Az0dYrq2+WPvVDENOW6tJ5N0HzdPtYrdiNqhuFzb/stgN
8KR44Oi89G4YEu202LDkLq593as6V5KErvlot56wn4BaMu9AsIj1XSkq4Q1syAsij3PSkWzyF6QUnt
hBEYpw8aGmmPz4X4W4EnDFzmo79Nq6sXPLdHEZpigx6XYEhb8VLTdcssiD2nJ/
ex2g61T9W4Wtl2Ed6oP1OpNytElbYNi35HzEUvPj5/MS+hDDqs4sOH/x4DMT4v56u+2IuaRxpbmzkP
gXDbP9WNUVKU1FS3+NGHRoictMI2zfQFyu09qLNV6xuqOm19pFuqAfkHuC73YCxYs97D7sjpUJ3Vmt
nmrgsvXmcHzBh9EaHxLsKAgnKUr9151mHpVshLW3LdxPZWrMzldz3Tzt+Eqo/1stqYh7T9uOA6lF8E
Xz447zwSqwuJvxo8dJ5Homq542sP9p8FH/
rLXp9YYYc+LYhbULY6CKJX8Od5vHmCOjZ4HGh+9wVdXXdVU/ItHxpN21+OsU9CqeGfkqpqK6Du4c2b
Y9JWQmyPVeD1WSNQpbaM7pnreyHvtcDl4rHxqP36nAWBBwdCTNGmwDSpFmp/1zTsYLMQ9n1v221lYg
V1A/bPO2SwEM6ejPDfsaYY3j/+0P/3xB/wzMaqqafiNjQ89/3zwKgRPZ/DTj6WaQDVAw/bV76agnLj
5W+4v3KG7pgZBxpkDqDO7PwpWvNNoNo/2yJGyxAielGok/9olDOzJC6rxwhe1GCKmNqKwjVHXsw5MB
x9UHlr9dzjF3T7/3A0/ToZXsV9H7alYzUq2vVo9MaDEsiJs/2rYzQTdXsEx2pj/
```

ErSfjjqaUQhahpRHZ660BlVFNe7l/jPgc7UWfMMTO5B3OCsRB2LGajm8+P5KfLRKHWm+c/
GEXXwKfzl79Ntc1Gu9OjQ+T/7g1R9450JH4oA5Ulm6g8bhppEV/
euepgIdYt/3f1+Khe1WAf12750IiQcPD5usupf1PV6s/eqtL3o3uh5km05HlCbMyWJveAtKj28+dja
GS3wud74qlaSCWrY/9vP1uc4dA+ZoPL8yQR4rzcosdemCUnTz7C+NntCxEAdz00Gv+Bg59PTr7RN0c
vrSWu1exUhZ6LphXpPNYhqO/720HMfVLvXnpVhcwaF1agF9BYGQLv1KbbxTiXY9Wn3+jUyB6BSNa9S
M/wNhG+5K2mL6UIfvOrzxlnqojdfNk+t/xYHZQNfHdoUeBLQ/
ktrEsvPo6qgWfJl7+PRoxjxogmlGlB7M9KicMhvSO9e/
mPeoqhYuczw+gnXt0H8Dm2jk2NwHTY0StP5XBmKf/LpiS+cgdrXHaXlq8vR5cf7BKd6rOCzYaVddoQ
mZAdXTh19LBTVHfo2a0rFerg19rbtjvn9IZvvGL7lizmEHAl8sOqPHnq63e6B8tUdEC8r1/
qy6CTqHdibViz7GbIbZW8HORijnrK8jZrXhqM3VTXpb2qyoTlzbYd8zBwoGtnfUXClG/UmbPXf43oH
FXU6Tt2NPsCvPxI5g4wTqOJ0Tk9HsCuUOqg9dJs2HWUeM+989DoOWn+GTvP2G4c6In7tfznFEcqOzU
5ZFVAAcdMWn7aeFYbaDd/
KDdiqid7eXPWw6Xw9vJt7LynZ4DdI//7co1c8MHbQ3N1RZ1aqQOqNfUPK08tR/
YNVQ18r2kHYy1WGseu0UHLwHfvId2KUNOaU/jffGihP8Zjk0pINV+yejao7rgxRcdIn13gvUfhRA7S
39Ba0LJMqR+hPROffV+ud31YCFZl6PwPHecGbFuOro2MfoIJs7+07MvgQNum3m93pASjad/0HA4vhK
LQwtTxy1A340hune0vWDvzdrcpGzUmED7f9JjQLXNDTtVtWBp8/BxEVtp2BvdOhxDN0+Qe1gNj+pTv
txq7zhkKLFNWtS0rQD+X9t6aPtoYo760nQgyqUO2CjGuJTtWoeW1mrv8dN/geXgV+E2vhenTCsJrjA
RBd2GI04d4WePf7aMkf6yhUxtqjMMFrCrrx8GV155MdUNgz3OCszg5U2JRqvlFFFzJlraWOivWQNDC
10L7MEHVFvJqkbiePPk4/PW4rzIOurJ2RsxU/QZjHFuPg5AqUKCdRy/+0Al4WFB44nHMJfWw/
tylyczjEPpR7HFz5FX3qt8CgbPsU9OrvWmu28kNomD7qx+gD0RAud/tu8LMYXO/
uH7C9HNe3fzIPFW2aCLU3rnmsW/IXotdkfMw4aIsaF/4enHp0Mzqza5Pr9IhZUDXsz+5aI190e6f+5
Y7HW6Hs05OngeofIHa9d1DCoxJUbVXXdM1lNOo20w2+NfsNfBmza9f8vOcgvWAzFTVfg7PXo+o2btV
F+UvmnllVyIdmV5VzmUYm8HLDk5y9p7+jvMkz4o7+kEF5iRfXztXbBVWzRp7bZHscwuJqBJZPp6EEu
4IRyytOQPzUu99ZhlfR7/Tj7po2ayDs5qfvXyI6UXLmyqDrnbNQdXZ+nM8cIbTpiTNPrjRB+TXf94n
zB0Pz30Hs49MqUUp6+tu6FDYkqE9wHHDRB116J2O+xDYN4h92/
v4S1ANZt1/93DzrBiqa5tZ1YdliuJmmmXtn1g9IUjRrTS4/
gpok68TLY9ohry64iR8XDxHjhm426F2CEiffNW3Y4wfxJ8uy/OekoOZYgZxZ2X548PVkp2J0EAqN+X
V7aVs5KlbafODdrZ9Qkpxu9VHNAsU768z5YfYMclUn5dSEt0Nvk//GlzabUG5YzjSvkFvwzFjqPOrs
VfTO1LnQpasKpOfaSnTHymCev0jhw5C3UCC3cP4l0yeoeEYU+0OFKgR0FNexzl6GiLh5ZoVqCyFym6
KL5rJWVKuUmNdytwjehjzNCpu3GzX4uw02vKAOMVyNkWtXpaEvf4fGHokvR63DWl7U9t6Cmld2OxNP
34ei8n3XZpfZoD8/rol1l/+Aqnfj1duP2KNG2cpV9no8SLjz2O7v7ouorJ/658hJuShncNMWmF4Lf7
bWj+Mskodo27sDMjYsRlJLj7PqNslorXPYgvqchZAjEB0SmS2B1IU3lYNcNqLqZ1XP7KZy4Kn5fTVh
RQw80Y77FXJzOyrfbig/+N4qaLrUzyHQ9ywq7d/vjVSUAx8Sw37JXX6EUgdL3/HuL4Y/AR2/
vwZHwTXD6qXn45dgvsgd1q1chipD9R7nfbSC+qaSR/XDeyDPT14mE3nHqly9O9f+5ymUf/9x/IIhMp
DvPdbviygAnXrU5ZCE+HAnqfvGiIF2kLhRU01UpoNaTxkED9pThF7M8Zt+6HQWZLRwgwRXK+BtwsOB
V5asR/W3vQe+Hl4Bya2aO7/
tl6LPKfLFb0YEwJvExwOsvnqgCsuh626tHAgRUydlH87ah7LHbenZevoDhG98r/
vTZiZKvTnVNiRNA+K0loW1DG9BNbe/
+p5PzEJnY63vvZ2UA7FPHVZ3nulFu6b3JqTaXYdG4ami+mgOahsiZ/zu3R4o/dEafuhAPYQWrfkdrT
cURWysHrFKiv1x0YbNJuXz0JcjIZ+7Zx+FN49K1hZiXK0783he8xhFdPPM2a8T1X3go+7tJypRI9CD
ETLuT0bfhfrvPdWpYcchIbBuW0rkKVT7Zce27qRtEPnAfnmAZzwqjf3a0KDKARSu/
Xxs91HU4jnE+aj7T0DR68f/WGCApBFT68N3jYfH9anHRg7YjY60O02PlKmFVMPVI7YpeKE6l6Q1rll
HIDJmw7bx23+i8m9blkxfaoE6FrFt0lyOgHR29ejFG84AWnVA6+eZ36h24qhzButY8LaoaVSclisqr
PbbMi7rK8pvGerweNEPyDe5dvjFzVT03jBi+5TSPGiKLnNlj1BErSO/C0M1RFBXhYJWND6DRHs/

R32fgagz53fVjNEB6ENlx4oNi5ZC8/bX1zVGTYWIs4mrvvr/RUmSA6dbeCqoxPiR7deZD6Bh+eyaE4
PU4fb5mMeDVvGgylOn2Wr5clS0LLxw5mhzkC7ZoD9BZRcqOZh3dvc0O2h6onZu2/
s9qC4oaCzr1x34HCn03z60DiLuJo8WhyWglP6ii9d7/6Kc+OSBPqMH6bFsYlkd/ZJQr/
x0da8gG5Cqnspf3X0KZT0eo7hHcgakK1ziHPerwI3y4knaAzQgtjzwRTZXDe5cd1XMqHgBvilOw+/J
DEZ39zufH7vSCLKN3kaq5e5FCXzPla5TF0L77yXWrUMGwDPOxpf3qq4AGnGq86x+AsrYb5qfZa4Enb
N3WiepPIB7OfVjrK8koivvb/w1KbmGrjmV1k4K0IAim/iFw15aQkfBSdGW8w4ofJRHSva3HVCaBkHn
NS+hP8fUbo+deAceqZ5JyZdYoccPoj+akv9PYPDi7PkvbujdtfGr7lxaDvEpr5d2bbuLvk1s5E3smQ
iRXNF05FaGiu0KwyszhiC/
sDtO6a1qWN+NNjojv0O8VuyvsKSFqOPGsfBfi63hXRt6UxpVg74crogY3/UI4lH//
L9OL1DTG1OrUzfLUNwlYcPEST7Q/
PjahpsuV4FzTXmYQ28NFLz80Tn7zE4UvbNkU1OpCdRwQlQ4vk8g/vjyDqP2+6hj3xJf5eY4kPruvuN
WfTtWxu5b8IeGw5B6+nHGmikTUd2em9efrTWAxLrmOx8U4lC1gXuu+SAbSHwd7aRStwm1PYiSyE4ZB
m9XLd+i9mw1+uwWcX7ctF9QPOfojtn+g1Gj4LCvXMVVFMXRM/p/
RZb5P9R5HMenKY1rcyvK0SVFzlpK0/vboanNat1pKiSSx1aTWlSUpVKiaxt3lIjCxBSt8zOZMCUhR4
4wYjBfZYwxBjO0+8vaf+D5ej6eP72kiw5Bc1XtxOWt+TDKr1alJz1Hfw9PhZwiOkD5guNnWbfkUb+s
e6I+2AJe/Vx1OuIcAb0X9U/mRbWjCV37yLKyHdBddrlx+91jwNFxynLfocyS21wusHKjwdBMfHL3by
ZQJKLRiEFkYLeWK6ZxrZGQIHLPmLVEg1gHabFjDEjOFK7xkm2AEUms68m9YtR7x/
Dl1cYDLLl9x0wYxCLoW+hfMRtnAS+k2+K0qcGoUPRJ76ipJ4sQIrEcLGqH/oGDa49oYlC5dd/
AVIwc4pniBnWsciSKjs2msIXQlUgMM6VUwnSdZU8zVxVx/thJlsqzUSnloj5njRsIu7GkFzdsITvNT
nibnY+Kjhr5FZybQWP8uA8Ef0cQeHhHmj2zAFZQgcRkiy4ac9yb8NH4IKRaqt6/52gFNQ3Hl+qHmrI
WFMTGcHuXwoSZHd6c3wllNmqJsUHeiH/d/
PA9AhtKE0SxtWxAfTay4BOkNijRzFa0k+xHvV8DPjltSobitK+7M71GUbtvlIenExEKn76pqPKph/I
9+YVZTmrozS8LFT5yBSDQK2aef2GESs88FAVvmIUxcpQLn+cJj+KZ60D9FOQUZv6VkHQRHifRqaoqO
6DJpN5lTSMG1cr7THTWKSJhZvXMWkYklBcPLY5qKUXcpiS9/dsWI0YKL0SjgQS4mKZUsjIGpXYGuqq
WPIFvXVdy8mmnEFf5BMFJ7iQMuTVdMRdsR81x1PMX+FrAo22zTDNmAXPyo3Vu+mH0clPuWKD17xCa+
6X9Wh4DBM7LndVIL1FmxDVnU1UiNB7jdLCyYhAnLNxyU3U8cI8365N7mv/9F8wNSzzU4AOzdLkpskT
cmkbDyUw+SNR5zq6GOGQHNMz0/9iFLnjG6Id5noNKem7l3pY9aLpHK0DmHAiNfWEnvmnHIXyaqfSWa
oNeWdvw3AyfAR61x7ojxQ9aUyoUza89YxG8Bqe0VHPQqdat3xV8i6E08tF4vJcqKiBrUnJXO4BQg51
D6tWCvKZuNNdUiZ6n5b22mg2HfN7VIgPKDLzx6bzBL3oIZd0RS4Y2ExC+whbfJx5EVSkrtgStTAacT
WeD0UpUdfVQij39NYgZ7mYhzaaQba/906y8GDHtHys8HHVAlV4qIdOvTaGWQbMbt+2D17z8uuBloaj
1nbPPnzgLip8qUxVTvFE7J3KZWfwDwG/QW/psM0BG0nmTP/UEpjpW10PJAIgSit6bv/WDqfGId3n+Z
EydAq68kGCMqJRANbBYBVN+9x1SNVpgKGqth+WRcJhYb/800XY5RlBMyHqcOIgtsqLIusIcQNBa5xb
owgSRE3URcZUiSI0HvrhsscEUDLXquYZ3MSVawmepfilMP2m7kbPBByP4k8gbO34Aru3CxDnFMP5RL
aNTkj7PEdxCVWLZaUylt+st7cGP//msOoW4+GPw7dZl77tAAf5JD/fkkc0wd9Ml+ozKMpjTOVAnvDI
Ms0ksc4O3NSCOGBgTEsMxQgMuS6oYweQutRm4+xiDdPlcMEMsB5O7UtO/
UnbCzO5o3T6yHIxy9TM7xc+xn3QNVKq1mDAZMujP2iYPgon0xCvfSSAaDzXZ7F4+3+M/L+HNmiCe9h
LAfbN0+JpWIGwP14xwGAZckx4lerURJOeU6LfzTgKferE9g9EMwhKOaPHKNpjKDdxztj4apj/
zCq+XqIOYzE3YrbIQJtouxbe/
s4Hhbl9p44VZmGx6YfhJPRTEcyZofZYefIs+bsWvr4Vp1+ReC94BjGD8qUV3+xDI/
LoWWNn5zftx5VPWnKnpx0hx6BInYzX0BGiQl8i/A6HR86If36/DMInaoM4ont8fDJ2mXv6uDMMZbJp
jnBV89faek6PiIHG8Yyc9JIKxGt9BvYqNILw68it94RgIVoy61hbcBqmFY7bZ9QAYqXAMPrFfDVvQ6
erusbPnH1BLBwj3Pzl+b00AAF5fAABQSwECFAAUAAgICACCTohR4GAdPOYAAAA0AQAAEwAAAAAAAAA
AAAAAAAAAAAAAbWFuaWZlc3QucHJvcGVydGllc1BLAQIUABQACAgIAIJOiFH3Pzl+b00AAF5fAAAKA
AAAAAAAAAAAAAAACcBAABzZW50Lm1vZGVsUEsFBgAAAAACAAIAeQAAAM5OAAAAAA==

langchain4j\src\test\java\dev\langchain4j\agent\tool\ToolExecutorTest.java

```
package dev.langchain4j.agent.tool;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
import java.math.BigDecimal;
import java.math.BigInteger;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
class ToolExecutorTest {
    TestTool testTool = new TestTool();
    private static class TestTool {
        @Tool
        double doubles(double arg0, Double arg1) {
            return arg0 + arg1;
        }
        @Tool
        float floats(float arg0, Float arg1) {
            return arg0 + arg1;
        }
        @Tool
        BigDecimal bigDecimals(BigDecimal arg0, BigDecimal arg1) {
            return arg0.add(arg1);
        }
        @Tool
        long longs(long arg0, Long arg1) {
            return arg0 + arg1;
        }
        @Tool
        int ints(int arg0, Integer arg1) {
            return arg0 + arg1;
        }
        @Tool
        short shorts(short arg0, Short arg1) {
            return (short) (arg0 + arg1);
        }
        @Tool
        byte bytes(byte arg0, Byte arg1) {
            return (byte) (arg0 + arg1);
        }
        @Tool
        BigInteger bigIntegers(BigInteger arg0, BigInteger arg1) {
            return arg0.add(arg1);
        }
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2}",
            "{\"arg0\": 2.0, \"arg1\": 2.0}",
            "{\"arg0\": 1.9, \"arg1\": 2.1}",
    })
    void should_execute_tool_with_parameters_of_type_double(String arguments)
throws NoSuchMethodException {
        executeAndAssert(arguments, "doubles", double.class, Double.class,
"4.0");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2}",
            "{\"arg0\": 2.0, \"arg1\": 2.0}",
```

```
            "{\"arg0\": 1.9, \"arg1\": 2.1}",
    })
    void should_execute_tool_with_parameters_of_type_float(String arguments)
throws NoSuchMethodException {
        executeAndAssert(arguments, "floats", float.class, Float.class,
"4.0");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": " + Float.MAX_VALUE + "}",
            "{\"arg0\": 2, \"arg1\": " + -Double.MAX_VALUE + "}"
    })
    void should_fail_when_argument_does_not_fit_into_float_type(String
arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "floats", float.class,
Float.class, "is out of range for the float type");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2}",
            "{\"arg0\": 2.0, \"arg1\": 2.0}",
            "{\"arg0\": 1.9, \"arg1\": 2.1}",
    })
    void should_execute_tool_with_parameters_of_type_BigDecimal(String
arguments) throws NoSuchMethodException {
        executeAndAssert(arguments, "bigDecimals", BigDecimal.class,
BigDecimal.class, "4.0");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2}",
            "{\"arg0\": 2.0, \"arg1\": 2.0}"
    })
    void should_execute_tool_with_parameters_of_type_long(String arguments)
throws NoSuchMethodException {
        executeAndAssert(arguments, "longs", long.class, Long.class, "4");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2.1}",
            "{\"arg0\": 2.1, \"arg1\": 2}"
    })
    void should_fail_when_argument_is_fractional_number_for_parameter_of_type_
long(String arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "longs", long.class, Long.class,
"argument type mismatch");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": " + Double.MAX_VALUE + "}",
            "{\"arg0\": 2, \"arg1\": " + -Double.MAX_VALUE + "}"
    })
    void should_fail_when_argument_does_not_fit_into_long_type(String
arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "longs", long.class, Long.class,
"is out of range for the long type");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2}",
            "{\"arg0\": 2.0, \"arg1\": 2.0}"
```

```
    })
    void should_execute_tool_with_parameters_of_type_int(String arguments)
throws NoSuchMethodException {
        executeAndAssert(arguments, "ints", int.class, Integer.class, "4");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2.1}",
            "{\"arg0\": 2.1, \"arg1\": 2}"
    })
    void should_fail_when_argument_is_fractional_number_for_parameter_of_type_
int(String arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "ints", int.class, Integer.class,
"argument type mismatch");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": " + Double.MAX_VALUE + "}",
            "{\"arg0\": 2, \"arg1\": " + -Double.MAX_VALUE + "}"
    })
    void should_fail_when_argument_does_not_fit_into_int_type(String
arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "ints", int.class, Integer.class,
"is out of range for the integer type");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2}",
            "{\"arg0\": 2.0, \"arg1\": 2.0}"
    })
    void should_execute_tool_with_parameters_of_type_short(String arguments)
throws NoSuchMethodException {
        executeAndAssert(arguments, "shorts", short.class, Short.class, "4");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2.1}",
            "{\"arg0\": 2.1, \"arg1\": 2}"
    })
    void should_fail_when_argument_is_fractional_number_for_parameter_of_type_
short(String arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "shorts", short.class,
Short.class, "argument type mismatch");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": " + Double.MAX_VALUE + "}",
            "{\"arg0\": 2, \"arg1\": " + -Double.MAX_VALUE + "}"
    })
    void should_fail_when_argument_does_not_fit_into_short_type(String
arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "shorts", short.class,
Short.class, "is out of range for the short type");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2}",
            "{\"arg0\": 2.0, \"arg1\": 2.0}"
    })
    void should_execute_tool_with_parameters_of_type_byte(String arguments)
throws NoSuchMethodException {
```

```
        executeAndAssert(arguments, "bytes", byte.class, Byte.class, "4");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2.1}",
            "{\"arg0\": 2.1, \"arg1\": 2}"
    })
    void should_fail_when_argument_is_fractional_number_for_parameter_of_type_
byte(String arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "bytes", byte.class, Byte.class,
"argument type mismatch");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": " + Double.MAX_VALUE + "}",
            "{\"arg0\": 2, \"arg1\": " + -Double.MAX_VALUE + "}"
    })
    void should_fail_when_argument_does_not_fit_into_byte_type(String
arguments) throws NoSuchMethodException {
        executeAndExpectFailure(arguments, "bytes", byte.class, Byte.class,
"is out of range for the byte type");
    }
    @ParameterizedTest
    @ValueSource(strings = {
            "{\"arg0\": 2, \"arg1\": 2}",
            "{\"arg0\": 2.0, \"arg1\": 2.0}"
    })
    void should_execute_tool_with_parameters_of_type_BigInteger(String
arguments) throws NoSuchMethodException {
        executeAndAssert(arguments, "bigIntegers", BigInteger.class,
BigInteger.class, "4");
    }
    private void executeAndAssert(String arguments, String methodName, Class<?
> arg0Type, Class<?> arg1Type, String expectedResult) throws
NoSuchMethodException {
        ToolExecutionRequest request = ToolExecutionRequest.builder()
                .arguments(arguments)
                .build();
        ToolExecutor toolExecutor = new ToolExecutor(testTool,
TestTool.class.getDeclaredMethod(methodName, arg0Type, arg1Type));
        String result = toolExecutor.execute(request);
        assertThat(result).isEqualTo(expectedResult);
    }
    private void executeAndExpectFailure(String arguments, String methodName,
Class<?> arg0Type, Class<?> arg1Type, String expectedError) throws
NoSuchMethodException {
        ToolExecutionRequest request = ToolExecutionRequest.builder()
                .arguments(arguments)
                .build();
        ToolExecutor toolExecutor = new ToolExecutor(testTool,
TestTool.class.getDeclaredMethod(methodName, arg0Type, arg1Type));
        assertThatThrownBy(() -> toolExecutor.execute(request))
                .isExactlyInstanceOf(IllegalArgumentException.class)
                .hasMessageContaining(expectedError);
    }
}
```

langchain4j\src\test\java\dev\langchain4j\chain\ConversationalChainTest.java

```
package dev.langchain4j.chain;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.NullSource;
import org.junit.jupiter.params.provider.ValueSource;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static java.util.Collections.singletonList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
import static org.mockito.Mockito.*;
public class ConversationalChainTest {
    @Test
    public void should_store_user_and_ai_messages_in_chat_memory() {
        // Given
        ChatLanguageModel chatLanguageModel = mock(ChatLanguageModel.class);
        String aiMessage = "Hi there";
        when(chatLanguageModel.generate(anyList())).thenReturn(Response.from(a
iMessage(aiMessage)));
        ChatMemory chatMemory =
spy(MessageWindowChatMemory.withMaxMessages(10));
        ConversationalChain chain = ConversationalChain.builder()
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(chatMemory)
                .build();
        String userMessage = "Hello";
        // When
        String response = chain.execute(userMessage);
        // Then
        assertThat(response).isEqualTo(aiMessage);
        verify(chatMemory).add(userMessage(userMessage));
        verify(chatMemory, times(3)).messages();

verify(chatLanguageModel).generate(singletonList(userMessage(userMessage)));
        verify(chatMemory).add(aiMessage(aiMessage));
        verifyNoMoreInteractions(chatMemory);
        verifyNoMoreInteractions(chatLanguageModel);
    }
    @ParameterizedTest
    @NullSource
    @ValueSource(strings = {"", " "})
    public void should_fail_when_user_message_is_null_or_blank(String
userMessage) {
        // Given
        ChatLanguageModel chatLanguageModel = mock(ChatLanguageModel.class);
        ChatMemory chatMemory = mock(ChatMemory.class);
        ConversationalChain chain = ConversationalChain.builder()
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(chatMemory)
                .build();
        // When & Then
        assertThatThrownBy(() -> chain.execute(userMessage))
                .isExactlyInstanceOf(IllegalArgumentException.class)
                .hasMessage("userMessage cannot be null or blank");
```

```
    }
}
```

langchain4j\src\test\java\dev\langchain4j\classification\EmbeddingModelTextClassifierTest.java

```
package dev.langchain4j.classification;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static dev.langchain4j.classification.EmbeddingModelTextClassifierTest.CustomerServiceCategory.*;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
class EmbeddingModelTextClassifierTest {
    enum CustomerServiceCategory {
        BILLING_AND_PAYMENTS,
        TECHNICAL_SUPPORT,
        ACCOUNT_MANAGEMENT,
        PRODUCT_INFORMATION,
        ORDER_STATUS,
        RETURNS_AND_EXCHANGES,
        FEEDBACK_AND_COMPLAINTS
    }
    private final Map<CustomerServiceCategory, List<String>> examples = new HashMap<>();
    {
        examples.put(BILLING_AND_PAYMENTS, asList(
                "Can I pay using PayPal?",
                "Do you accept Bitcoin?",
                "Is it possible to pay via wire transfer?",
                "I keep getting an error message when I try to pay.",
                "My card was charged twice, can you help?",
                "Why was my payment declined?",
                "How can I request a refund?",
                "When will I get my refund?",
                "Can I get a refund if I cancel my subscription?",
                "Can you send me an invoice for my last order?",
                "I didn't receive a receipt for my purchase.",
                "Is the invoice sent to my email automatically?",
                "How do I upgrade my subscription?",
                "What are the differences between the Basic and Premium plans?",
                "How do I cancel my subscription?",
                "Can I switch to a monthly plan from an annual one?",
                "I want to downgrade my subscription, how do I go about it?",
                "Is there a penalty for downgrading my plan?"
        ));
        examples.put(TECHNICAL_SUPPORT, asList(
                "The app keeps crashing whenever I open it.",
                "I can't save changes in the settings.",
                "Why is the search function not working?",
                "The installer is stuck at 50%.",
                "I keep getting an ‘Installation Failed' message.",
                "How do I install this on a Mac?",
                "I can't connect to the server.",
                "Why am I constantly getting disconnected?",
                "My Wi-Fi works, but your app says no internet connection.",
                "Why is the app so slow?",
                "I'm experiencing lag during video calls.",
```

```
        "The website keeps freezing on my browser.",
        "I get a ‘404 Not Found' error.",
        "What does the ‘Permission Denied' error mean?",
        "Why am I seeing an ‘Insufficient Storage' warning?",
        "Is this compatible with Windows 11?",
        "The app doesn't work on my Android phone.",
        "Do you have a browser extension for Safari?"
));
examples.put(ACCOUNT_MANAGEMENT, asList(
        "I forgot my password, how can I reset it?",
        "I didn't receive a password reset email.",
        "Is there a way to change my password from within the app?",
        "How do I set up two-factor authentication?",
        "I lost my phone, how can I log in now?",
        "Why am I not getting the 2FA code?",
        "My account has been locked, what do I do?",
        "Is there a limit on login attempts?",
        "I've been locked out for no reason, can you help?",
        "How do I change my email address?",
        "Can I update my profile picture?",
        "How do I edit my shipping address?",
        "Can I share my account with family?",
        "How do I give admin access to my team member?",
        "Is there a guest access feature?",
        "How do I delete my account?",
        "What happens to my data if I deactivate my account?",
        "Can I reactivate my account later?"
));
examples.put(PRODUCT_INFORMATION, asList(
        "What does the ‘Sync' feature do?",
        "How does the privacy mode work?",
        "Can you explain the real-time tracking feature?",
        "When will the new model be in stock?",
        "Do you have this item in a size medium?",
        "Are you restocking the sold-out items soon?",
        "What's the difference between version 1.0 and 2.0?",
        "Is the Pro version worth the extra cost?",
        "Do older versions support the new update?",
        "Is this product compatible with iOS?",
        "Will this work with a 220V power supply?",
        "Do you have options for USB-C?",
        "Are there any accessories included?",
        "Do you sell protective cases for this model?",
        "What add-ons would you recommend?",
        "What does the warranty cover?",
        "How do I claim the warranty?",
        "Is the warranty international?"
));
examples.put(ORDER_STATUS, asList(
        "Where is my order right now?",
        "Can you give me a tracking number?",
        "How do I know my order has been shipped?",
        "Can I change the shipping method?",
        "Do you offer overnight shipping?",
        "Is pickup from the store an option?",
        "When will my order arrive?",
        "Why is my delivery delayed?",
        "Can I specify a delivery date?",
        "It's past the delivery date, where is my order?",
        "Will I be notified if there's a delay?",
        "How long will the weather delay my shipment?",
```

```
                "I received my order, but an item is missing.",
                "The package was empty when it arrived.",
                "I got the wrong item, what should I do?",
                "Will all my items arrive at the same time?",
                "Why did I receive only part of my order?",
                "Is it possible to get the remaining items faster?"
        ));
        examples.put(RETURNS_AND_EXCHANGES, asList(
                "What's your return policy?",
                "Is the return shipping free?",
                "Do I need the original packaging to return?",
                "How do I get a return label?",
                "Do I need to call customer service for a return?",
                "Is an RMA number required?",
                "I need to exchange for a different size.",
                "Can I exchange a gift?",
                "How long does the exchange process take?",
                "My item arrived damaged, what do I do?",
                "The product doesn't work as described.",
                "There's a part missing, can you send it?",
                "I received the wrong item, how can I get it corrected?",
                "I didn't order this, why did I receive it?",
                "You sent me two of the same item by mistake.",
                "Is there a restocking fee for returns?",
                "Will I get a full refund?",
                "How much will be deducted for restocking?"
        ));
        examples.put(FEEDBACK_AND_COMPLAINTS, asList(
                "The material quality is not as advertised.",
                "The product broke after a week of use.",
                "The colors faded after the first wash.",
                "The representative was rude to me.",
                "I was on hold for 30 minutes, this is unacceptable.",
                "Your customer service resolved my issue quickly, thank you!",
                "Your website is hard to navigate.",
                "The app keeps crashing, it's frustrating.",
                "The checkout process is confusing.",
                "You should offer a chat feature for quicker help.",
                "Can you add a wishlist feature?",
                "Please make a mobile-friendly version of the website.",
                "I found a bug in your software.",
                "There's a typo on your homepage.",
                "The payment page has a glitch.",
                "Can you start offering this in a gluten-free option?",
                "Please add support for Linux.",
                "I wish you had more colors to choose from."
        ));
    }
    @Test
    @Disabled
    void should_return_one_category_by_default() {
        TextClassifier<CustomerServiceCategory> classifier = new
EmbeddingModelTextClassifier<>(
                new AllMiniLmL6V2QuantizedEmbeddingModel(),
                examples
        );
        List<CustomerServiceCategory> categories = classifier.classify("Yo
where is my order?");
        assertThat(categories).containsExactly(ORDER_STATUS);
    }
    @Test
```

```
    @Disabled
    void should_return_multiple_categories() {
        TextClassifier<CustomerServiceCategory> classifier = new
EmbeddingModelTextClassifier<>(
                new AllMiniLmL6V2QuantizedEmbeddingModel(),
                examples,
                2,
                0,
                0.5
        );
        List<CustomerServiceCategory> categories = classifier.classify("Bro,
this product is crap");
        assertThat(categories).containsExactly(RETURNS_AND_EXCHANGES,
FEEDBACK_AND_COMPLAINTS);
    }
    @Test
    @Disabled
    void should_classify_respecting_minScore() {
        TextClassifier<CustomerServiceCategory> classifier = new
EmbeddingModelTextClassifier<>(
                new AllMiniLmL6V2QuantizedEmbeddingModel(),
                examples,
                2,
                0.64,
                0.5
        );
        List<CustomerServiceCategory> categories = classifier.classify("Bro,
this product is crap");
        assertThat(categories).containsExactly(RETURNS_AND_EXCHANGES);
    }
    @Test
    @Disabled
    void should_classify_respecting_meanToMaxScoreRatio_1() {
        TextClassifier<CustomerServiceCategory> classifier = new
EmbeddingModelTextClassifier<>(
                new AllMiniLmL6V2QuantizedEmbeddingModel(),
                examples,
                1,
                0,
                1
        );
        List<CustomerServiceCategory> categories = classifier.classify("Bro,
this product is crap");
        assertThat(categories).containsExactly(FEEDBACK_AND_COMPLAINTS);
    }
    @Test
    @Disabled
    void should_classify_respecting_meanToMaxScoreRatio_0() {
        TextClassifier<CustomerServiceCategory> classifier = new
EmbeddingModelTextClassifier<>(
                new AllMiniLmL6V2QuantizedEmbeddingModel(),
                examples,
                1,
                0,
                0
        );
        List<CustomerServiceCategory> categories = classifier.classify("Bro,
this product is crap");
        assertThat(categories).containsExactly(RETURNS_AND_EXCHANGES);
    }
}
```

langchain4j\src\test\java\dev\langchain4j\code\JavaScriptCodeFixerTest.java

```
package dev.langchain4j.code;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class JavaScriptCodeFixerTest {
    @Test
    void should_fix_multi_line_code_when_result_is_not_logged_to_console() {
        String code = "const startDate = new Date('Feb 21, 1988 17:00:00');
\nconst endDate = new Date('Apr 12, 2014 04:00:00');\nconst diff = endDate -
startDate;\nconst result = diff / (1000*60*60);\n;const startDate = new
Date('Feb 21, 1988 17:00:00');\nconst endDate = new Date('Apr 12, 2014
04:00:00');\nconst diff = endDate - startDate;\nconst result = diff /
(1000*60*60);\n";
        String result = JavaScriptCodeFixer.fixIfNoLogToConsole(code +
"result");
        assertThat(result).isEqualTo(code + "console.log(result);");
    }
    @Test
    void should_not_fix_multi_line_code_when_result_is_logged_to_console() {
        String code = "const startDate = new Date('Feb 21, 1988 17:00:00');
\nconst endDate = new Date('Apr 12, 2014 04:00:00');\nconst diff = endDate -
startDate;\nconst result = diff / (1000*60*60);\nconsole.log(result);";
        String result = JavaScriptCodeFixer.fixIfNoLogToConsole(code);
        assertThat(result).isEqualTo(code);
    }
    @Test
    void should_fix_one_line_code_when_result_is_not_logged_to_console() {
        String code = "const start = new Date('1988-02-21T17:00:00'); const
end = new Date('2014-04-12T04:00:00'); const hours = (end - start) / (1000 *
60 * 60); ";
        String result = JavaScriptCodeFixer.fixIfNoLogToConsole(code +
"hours");
        assertThat(result).isEqualTo(code + "console.log(hours);");
    }
    @Test
    void should_not_fix_one_line_code_when_result_is_logged_to_console() {
        String code = "const start = new Date('1988-02-21T17:00:00'); const
end = new Date('2014-04-12T04:00:00'); const hours = (end - start) / (1000 *
60 * 60); console.log(hours);";
        String result = JavaScriptCodeFixer.fixIfNoLogToConsole(code);
        assertThat(result).isEqualTo(code);
    }
    @Test
    void should_fix_one_statement_when_result_is_not_logged_to_console() {
        String code = "Math.sqrt(49506838032859)";
        String result = JavaScriptCodeFixer.fixIfNoLogToConsole(code);

assertThat(result).isEqualTo("console.log(Math.sqrt(49506838032859));");
    }
    @Test
    void should_not_fix_one_statement_when_result_is_logged_to_console() {
        String code = "console.log(Math.sqrt(49506838032859));";
        String result = JavaScriptCodeFixer.fixIfNoLogToConsole(code);
        assertThat(result).isEqualTo(code);
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\FileSystemDocumentLoaderTest.java

```
package dev.langchain4j.data.document;
import org.junit.jupiter.api.Test;
import java.net.URISyntaxException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import static dev.langchain4j.data.document.Document.DOCUMENT_TYPE;
import static dev.langchain4j.data.document.Document.FILE_NAME;
import static dev.langchain4j.data.document.DocumentType.UNKNOWN;
import static
dev.langchain4j.data.document.FileSystemDocumentLoader.loadDocument;
import static
dev.langchain4j.data.document.FileSystemDocumentLoader.loadDocuments;
import static java.util.stream.Collectors.toList;
import static org.assertj.core.api.Assertions.assertThat;
class FileSystemDocumentLoaderTest {
    @Test
    void should_load_text_document() {
        Document document = loadDocument(toPath("test-file-utf8.txt"));
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("file_name")).isEqualTo("test-file-utf8.txt");

assertThat(Paths.get(metadata.get("absolute_directory_path"))).isAbsolute();
    }
    @Test
    void should_load_pdf_document() {
        Document document = loadDocument(toPath("test-file.pdf"));
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("file_name")).isEqualTo("test-file.pdf");

assertThat(Paths.get(metadata.get("absolute_directory_path"))).isAbsolute();
    }
    @Test
    void should_load_ppt_document() {
        Document document = loadDocument(toPath("test-file.ppt"));
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("file_name")).isEqualTo("test-file.ppt");

assertThat(Paths.get(metadata.get("absolute_directory_path"))).isAbsolute();
    }
    @Test
    void should_load_pptx_document() {
        Document document = loadDocument(toPath("test-file.pptx"));
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("file_name")).isEqualTo("test-file.pptx");

assertThat(Paths.get(metadata.get("absolute_directory_path"))).isAbsolute();
    }
```

```
    @Test
    void should_load_doc_document() {
        Document document = loadDocument(toPath("test-file.doc"));
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("file_name")).isEqualTo("test-file.doc");

assertThat(Paths.get(metadata.get("absolute_directory_path"))).isAbsolute();
    }
    @Test
    void should_load_docx_document() {
        Document document = loadDocument(toPath("test-file.docx"));
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("file_name")).isEqualTo("test-file.docx");

assertThat(Paths.get(metadata.get("absolute_directory_path"))).isAbsolute();
    }
    @Test
    void should_load_xls_document() {
        Document document = loadDocument(toPath("test-file.xls"));
        assertThat(document.text()).isEqualToIgnoringWhitespace("Sheet1\ntest
content\nSheet2\ntest content");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("file_name")).isEqualTo("test-file.xls");

assertThat(Paths.get(metadata.get("absolute_directory_path"))).isAbsolute();
    }
    @Test
    void should_load_xlsx_document() {
        Document document = loadDocument(toPath("test-file.xlsx"));
        assertThat(document.text()).isEqualToIgnoringWhitespace("Sheet1\ntest
content\nSheet2\ntest content");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("file_name")).isEqualTo("test-file.xlsx");

assertThat(Paths.get(metadata.get("absolute_directory_path"))).isAbsolute();
    }
    @Test
    void
should_load_documents_from_directory_including_unknown_document_types() {
        String userDir = System.getProperty("user.dir");
        Path resourceDirectory = Paths.get(userDir, "langchain4j/src/test/
resources");
        if (!Files.exists(resourceDirectory)) {
            resourceDirectory = Paths.get(userDir, "src/test/resources");
        }
        List<Document> documents = loadDocuments(resourceDirectory);
        assertThat(documents).hasSize(10);
        List<Document> documentsWithUnknownType = documents.stream()
                .filter(document ->
document.metadata(DOCUMENT_TYPE).equals(UNKNOWN.toString()))
                .collect(toList());
        assertThat(documentsWithUnknownType).hasSize(1);
        assertThat(documentsWithUnknownType.get(0).metadata(FILE_NAME)).isEqua
lTo("test-file.banana");
    }
    private Path toPath(String fileName) {
        try {
```

```
            return
Paths.get(getClass().getClassLoader().getResource(fileName).toURI());
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\parser\MsOfficeDocumentParserTest.java

```
package dev.langchain4j.data.document.parser;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentParser;
import org.junit.jupiter.api.Test;
import java.io.InputStream;
import static dev.langchain4j.data.document.DocumentType.*;
import static org.assertj.core.api.Assertions.assertThat;
public class MsOfficeDocumentParserTest {
    @Test
    void should_parse_ppt_file() {
        DocumentParser parser = new MsOfficeDocumentParser(PPT);
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("test-file.ppt");
        Document document = parser.parse(inputStream);
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        assertThat(document.metadata("document_type")).isEqualTo("PPT");
    }
    @Test
    void should_parse_pptx_file() {
        DocumentParser parser = new MsOfficeDocumentParser(PPT);
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("test-file.pptx");
        Document document = parser.parse(inputStream);
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        assertThat(document.metadata("document_type")).isEqualTo("PPT");
    }
    @Test
    void should_parse_doc_file() {
        DocumentParser parser = new MsOfficeDocumentParser(DOC);
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("test-file.doc");
        Document document = parser.parse(inputStream);
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        assertThat(document.metadata("document_type")).isEqualTo("DOC");
    }
    @Test
    void should_parse_docx_file() {
        DocumentParser parser = new MsOfficeDocumentParser(DOC);
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("test-file.docx");
        Document document = parser.parse(inputStream);
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        assertThat(document.metadata("document_type")).isEqualTo("DOC");
    }
    @Test
    void should_parse_xls_file() {
        DocumentParser parser = new MsOfficeDocumentParser(XLS);
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("test-file.xls");
        Document document = parser.parse(inputStream);
        assertThat(document.text()).isEqualToIgnoringWhitespace("Sheet1\ntest
content\nSheet2\ntest content");
        assertThat(document.metadata("document_type")).isEqualTo("XLS");
```

```
    }
    @Test
    void should_parse_xlsx_file() {
        DocumentParser parser = new MsOfficeDocumentParser(XLS);
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("test-file.xlsx");
        Document document = parser.parse(inputStream);
        assertThat(document.text()).isEqualToIgnoringWhitespace("Sheet1\ntest
content\nSheet2\ntest content");
        assertThat(document.metadata("document_type")).isEqualTo("XLS");
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\parser\TextDocumentParserTest.java

```
package dev.langchain4j.data.document.parser;
import dev.langchain4j.data.document.Document;
import org.junit.jupiter.api.Test;
import java.io.InputStream;
import static dev.langchain4j.data.document.DocumentType.TXT;
import static java.nio.charset.StandardCharsets.ISO_8859_1;
import static org.assertj.core.api.Assertions.assertThat;
class TextDocumentParserTest {
    @Test
    void should_parse_with_utf8_charset_by_default() {
        TextDocumentParser parser = new TextDocumentParser(TXT);
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("test-file-utf8.txt");
        Document document = parser.parse(inputStream);
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        assertThat(document.metadata("document_type")).isEqualTo("TXT");
    }
    @Test
    void should_parse_with_specified_charset() {
        TextDocumentParser parser = new TextDocumentParser(TXT, ISO_8859_1);
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("test-file-iso-8859-1.txt");
        Document document = parser.parse(inputStream);
        assertThat(document.text()).isEqualToIgnoringWhitespace("test
content");
        assertThat(document.metadata("document_type")).isEqualTo("TXT");
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\S3DirectoryLoaderIT.java

```
package dev.langchain4j.data.document;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.testcontainers.containers.localstack.LocalStackContainer;
import org.testcontainers.utility.DockerImageName;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.core.sync.RequestBody;
import java.util.List;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static
org.testcontainers.containers.localstack.LocalStackContainer.Service.S3;
@Disabled("To run this test, you need a Docker-API compatible container
runtime, such as using Testcontainers Cloud or installing Docker locally.")
public class S3DirectoryLoaderIT {
    private LocalStackContainer s3Container;
    private S3Client s3Client;
    @BeforeAll
    public static void setUpClass() {
        System.setProperty("aws.region", "us-east-1");
    }
    @BeforeEach
    public void setUp() {
        s3Container = new
LocalStackContainer(DockerImageName.parse("localstack/localstack:2.0"))
                .withServices(S3)
                .withEnv("DEFAULT_REGION", "us-east-1");
        s3Container.start();
        s3Client = S3Client.builder()
                .endpointOverride(s3Container.getEndpointOverride(S3))
                .build();
        s3Client.createBucket(CreateBucketRequest.builder().bucket("test-
bucket").build());
    }
    @Test
    public void should_load_empty_list() {
        S3DirectoryLoader s3DirectoryLoader = S3DirectoryLoader.builder()
                .bucket("test-bucket")
                .endpointUrl(s3Container.getEndpointOverride(S3).toString())
                .build();
        List<Document> documents = s3DirectoryLoader.load();
        assertTrue(documents.isEmpty());
    }
    @Test
    public void should_load_multiple_files_without_prefix() {
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("file1.txt").build(),
                RequestBody.fromString("Hello, World!"));
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("directory/file2.txt").build(),
                RequestBody.fromString("Hello, again!"));
```

```
        S3DirectoryLoader s3DirectoryLoader = S3DirectoryLoader.builder()
                .bucket("test-bucket")
                .endpointUrl(s3Container.getEndpointOverride(S3).toString())
                .build();
        List<Document> documents = s3DirectoryLoader.load();
        assertEquals(2, documents.size());
        assertEquals("Hello, again!", documents.get(0).text());
        assertEquals("s3://test-bucket/directory/file2.txt",
documents.get(0).metadata("source"));
        assertEquals("Hello, World!", documents.get(1).text());
        assertEquals("s3://test-bucket/file1.txt",
documents.get(1).metadata("source"));
    }
    @Test
    public void should_load_multiple_files_with_prefix() {
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("other_directory/file1.txt").build(),
                RequestBody.fromString("You cannot load me!"));
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("directory/file2.txt").build(),
                RequestBody.fromString("Hello, World!"));
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("directory/file3.txt").build(),
                RequestBody.fromString("Hello, again!"));
        S3DirectoryLoader s3DirectoryLoader = S3DirectoryLoader.builder()
                .bucket("test-bucket")
                .prefix("directory")
                .endpointUrl(s3Container.getEndpointOverride(S3).toString())
                .build();
        List<Document> documents = s3DirectoryLoader.load();
        assertEquals(2, documents.size());
        assertEquals("Hello, World!", documents.get(0).text());
        assertEquals("s3://test-bucket/directory/file2.txt",
documents.get(0).metadata("source"));
        assertEquals("Hello, again!", documents.get(1).text());
        assertEquals("s3://test-bucket/directory/file3.txt",
documents.get(1).metadata("source"));
    }
    @Test
    public void should_load_accepting_unknown_types() {
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("directory/file2.unknown").build(),
                RequestBody.fromString("I am unknown."));
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("directory/file3.txt").build(),
                RequestBody.fromString("Hello, World!"));
        S3DirectoryLoader s3DirectoryLoader = S3DirectoryLoader.builder()
                .bucket("test-bucket")
                .prefix("directory")
                .endpointUrl(s3Container.getEndpointOverride(S3).toString())
                .build();
        List<Document> documents = s3DirectoryLoader.load();
        assertEquals(2, documents.size());
        assertEquals("I am unknown.", documents.get(0).text());
        assertEquals("s3://test-bucket/directory/file2.unknown",
documents.get(0).metadata("source"));
        assertEquals("Hello, World!", documents.get(1).text());
        assertEquals("s3://test-bucket/directory/file3.txt",
documents.get(1).metadata("source"));
    }
    @AfterEach
```

```
    public void tearDown() {
        s3Container.stop();
    }
    @AfterAll
    public static void tearDownClass() {
        System.clearProperty("aws.region");
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\S3DirectoryLoaderTest.java

```
package dev.langchain4j.data.document;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.S3Object;
import java.io.ByteArrayInputStream;
import java.util.Arrays;
import java.util.List;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.when;
@ExtendWith(MockitoExtension.class)
public class S3DirectoryLoaderTest {
    @Mock
    private S3Client s3Client;
    @Mock
    private ListObjectsV2Response listObjectsV2Response;
    @Mock
    private GetObjectResponse getObjectResponse;
    private S3DirectoryLoader s3DirectoryLoader;
    @BeforeEach
    public void setUp() {
        s3DirectoryLoader = S3DirectoryLoader.builder()
                .bucket("langchain4j")
                .prefix("testPrefix")
                .build();
    }
    @Test
    public void should_load_documents_from_directory() {
        S3Object s3Object1 = S3Object.builder().key("testPrefix/
testKey1.txt").size(10L).build();
        S3Object s3Object2 = S3Object.builder().key("testPrefix/
testKey2.txt").size(20L).build();

when(listObjectsV2Response.contents()).thenReturn(Arrays.asList(s3Object1,
s3Object2));
        when(s3Client.listObjectsV2(any(ListObjectsV2Request.class))).thenRetu
rn(listObjectsV2Response);
        ResponseInputStream<GetObjectResponse> responseInputStream1 = new
ResponseInputStream<>(getObjectResponse, new
ByteArrayInputStream("test1".getBytes()));
        ResponseInputStream<GetObjectResponse> responseInputStream2 = new
ResponseInputStream<>(getObjectResponse, new
ByteArrayInputStream("test2".getBytes()));
        when(s3Client.getObject(any(GetObjectRequest.class))).thenReturn(respo
nseInputStream1).thenReturn(responseInputStream2);
```

```
        List<Document> documents = s3DirectoryLoader.load(s3Client);
        assertEquals(2, documents.size());
        assertEquals("test1", documents.get(0).text());
        assertEquals("test2", documents.get(1).text());
        assertEquals("s3://langchain4j/testPrefix/testKey1.txt",
documents.get(0).metadata("source"));
        assertEquals("s3://langchain4j/testPrefix/testKey2.txt",
documents.get(1).metadata("source"));
    }
    @Test
    public void
should_load_documents_from_directory_accepting_unknown_types() {
        S3Object s3Object1 = S3Object.builder().key("testPrefix/
testKey1.txt").size(10L).build();
        S3Object s3Object2 = S3Object.builder().key("testPrefix/
testKey2.txt").size(20L).build();
        S3Object s3Object3 = S3Object.builder().key("testPrefix/
testKey3.unknown").size(30L).build();

when(listObjectsV2Response.contents()).thenReturn(Arrays.asList(s3Object1,
s3Object2, s3Object3));
        when(s3Client.listObjectsV2(any(ListObjectsV2Request.class))).thenRetu
rn(listObjectsV2Response);
        ResponseInputStream<GetObjectResponse> responseInputStream1 = new
ResponseInputStream<>(getObjectResponse, new
ByteArrayInputStream("test1".getBytes()));
        ResponseInputStream<GetObjectResponse> responseInputStream2 = new
ResponseInputStream<>(getObjectResponse, new
ByteArrayInputStream("test2".getBytes()));
        ResponseInputStream<GetObjectResponse> responseInputStream3 = new
ResponseInputStream<>(getObjectResponse, new
ByteArrayInputStream("unknown".getBytes()));
        when(s3Client.getObject(any(GetObjectRequest.class))).thenReturn(respo
nseInputStream1).thenReturn(responseInputStream2).thenReturn(responseInputStre
am3);
        List<Document> documents = s3DirectoryLoader.load(s3Client);
        assertEquals(3, documents.size());
        assertEquals("test1", documents.get(0).text());
        assertEquals("test2", documents.get(1).text());
        assertEquals("unknown", documents.get(2).text());
        assertEquals("s3://langchain4j/testPrefix/testKey1.txt",
documents.get(0).metadata("source"));
        assertEquals("s3://langchain4j/testPrefix/testKey2.txt",
documents.get(1).metadata("source"));
        assertEquals("s3://langchain4j/testPrefix/testKey3.unknown",
documents.get(2).metadata("source"));
    }
    @Test
    public void should_return_empty_list_when_no_objects() {
        when(listObjectsV2Response.contents()).thenReturn(Arrays.asList());
        when(s3Client.listObjectsV2(any(ListObjectsV2Request.class))).thenRetu
rn(listObjectsV2Response);
        List<Document> documents = s3DirectoryLoader.load(s3Client);
        assertTrue(documents.isEmpty());
    }
    @Test
    public void should_throw_s3_exception() {
        when(s3Client.listObjectsV2(any(ListObjectsV2Request.class))).thenThro
w(S3Exception.builder().message("S3 error").build());
        assertThrows(RuntimeException.class, () ->
s3DirectoryLoader.load(s3Client));
```

```
    }
    @Test
    public void should_throw_invalid_bucket() {
        assertThrows(IllegalArgumentException.class, () ->
S3DirectoryLoader.builder()
                .prefix("testPrefix")
                .build());
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\S3FileLoaderIT.java

```
package dev.langchain4j.data.document;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.testcontainers.containers.localstack.LocalStackContainer;
import org.testcontainers.utility.DockerImageName;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.core.sync.RequestBody;
import static
org.testcontainers.containers.localstack.LocalStackContainer.Service.S3;
import static org.junit.jupiter.api.Assertions.*;
@Disabled("To run this test, you need a Docker-API compatible container
runtime, such as using Testcontainers Cloud or installing Docker locally.")
public class S3FileLoaderIT {
    private LocalStackContainer s3Container;
    private S3Client s3Client;
    private static final DockerImageName localstackImage =
DockerImageName.parse("localstack/localstack:2.0");
    @BeforeAll
    public static void setUpClass() {
        System.setProperty("aws.region", "us-east-1");
    }
    @BeforeEach
    public void setUp() {
        s3Container = new LocalStackContainer(localstackImage)
                .withServices(S3)
                .withEnv("DEFAULT_REGION", "us-east-1");
        s3Container.start();
        s3Client = S3Client.builder()
                .endpointOverride(s3Container.getEndpointOverride(LocalStackCo
ntainer.Service.S3))
                .build();
        s3Client.createBucket(CreateBucketRequest.builder().bucket("test-
bucket").build());
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("test-file.txt").build(),
                RequestBody.fromString("Hello, World!"));
    }
    @Test
    public void should_load_document() {
        S3FileLoader s3FileLoader = S3FileLoader.builder()
                .bucket("test-bucket")
                .key("test-file.txt")
                .endpointUrl(s3Container.getEndpointOverride(LocalStackContain
er.Service.S3).toString())
                .build();
        Document document = s3FileLoader.load();
        assertNotNull(document);
        assertEquals("Hello, World!", document.text());
        assertEquals("s3://test-bucket/test-file.txt",
document.metadata("source"));
    }
    @Test
```

```
    public void should_load_document_unknown_type() {
        S3Client s3Client = S3Client.builder()
                .endpointOverride(s3Container.getEndpointOverride(LocalStackCo
ntainer.Service.S3))
                .build();
        s3Client.createBucket(CreateBucketRequest.builder().bucket("test-
bucket").build());
        s3Client.putObject(PutObjectRequest.builder().bucket("test-
bucket").key("unknown-test-file.unknown").build(),
                RequestBody.fromString("Hello, World! I am Unknown"));
        S3FileLoader s3FileLoader = S3FileLoader.builder()
                .bucket("test-bucket")
                .key("unknown-test-file.unknown")
                .endpointUrl(s3Container.getEndpointOverride(LocalStackContain
er.Service.S3).toString())
                .build();
        Document document = s3FileLoader.load();
        assertNotNull(document);
        assertEquals("Hello, World! I am Unknown", document.text());
        assertEquals("s3://test-bucket/unknown-test-file.unknown",
document.metadata("source"));
    }
    @AfterEach
    public void tearDown() {
        s3Container.stop();
    }
    @AfterAll
    public static void tearDownClass() {
        System.clearProperty("aws.region");
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\S3FileLoaderTest.java

```
package dev.langchain4j.data.document;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import java.io.ByteArrayInputStream;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.when;
@ExtendWith(MockitoExtension.class)
public class S3FileLoaderTest {
    @Mock
    private S3Client s3Client;
    @Mock
    private GetObjectResponse getObjectResponse;
    private S3FileLoader s3FileLoader;
    @BeforeEach
    public void setUp() {
        s3FileLoader = S3FileLoader.builder()
                .bucket("langchain4j")
                .key("key.txt")
                .build();
    }
    @Test
    public void should_load_document() {
        ResponseInputStream<GetObjectResponse> responseInputStream = new
ResponseInputStream<>(getObjectResponse, new
ByteArrayInputStream("test".getBytes()));
        when(s3Client.getObject(any(GetObjectRequest.class))).thenReturn(respo
nseInputStream);
        Document result = s3FileLoader.load(s3Client);
        assertNotNull(result);
        assertEquals("test", result.text());
        assertEquals("s3://langchain4j/key.txt", result.metadata("source"));
    }
    @Test
    public void should_throw_s3_exception() {
        when(s3Client.getObject(any(GetObjectRequest.class))).thenThrow(S3Exce
ption.builder().message("S3 error").build());
        assertThrows(RuntimeException.class, () ->
s3FileLoader.load(s3Client));
    }
    @Test
    public void should_throw_invalid_key() {
        assertThrows(IllegalArgumentException.class, () ->
S3FileLoader.builder()
                .bucket("testBucket")
                .build());
    }
    @Test
```

```
    public void should_throw_invalid_bucket() {
        assertThrows(IllegalArgumentException.class, () ->
S3FileLoader.builder()
                .key("testKey")
                .build());
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\splitter\DocumentByParagraphSplitterTest.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.data.document.Metadata.metadata;
import static dev.langchain4j.data.segment.TextSegment.textSegment;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static java.lang.String.format;
import static org.assertj.core.api.Assertions.assertThat;
class DocumentByParagraphSplitterTest {
    @Test
    void should_split_into_segments_with_one_paragraph_per_segment() {
        int maxSegmentSize = 30;
        String firstParagraph = "This is a first paragraph.";
        assertThat(firstParagraph).hasSizeLessThan(maxSegmentSize);
        String secondParagraph = "This is a second paragraph.";
        assertThat(secondParagraph).hasSizeLessThan(maxSegmentSize);
        assertThat(firstParagraph + "\n \n" +
secondParagraph).hasSizeGreaterThan(maxSegmentSize);
        Document document = Document.from(
                format(" %s \n \n %s ", firstParagraph, secondParagraph),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentByParagraphSplitter(maxSegmentSize, 0);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(firstParagraph, metadata("index",
"0").add("document", "0")),
                textSegment(secondParagraph, metadata("index",
"1").add("document", "0"))
        );
    }
    @Test
    void should_split_into_segments_with_multiple_paragraphs_per_segment() {
        int maxSegmentSize = 60;
        String firstParagraph = "This is a first paragraph.";
        String secondParagraph = "This is a second paragraph.";
        assertThat(firstParagraph +
secondParagraph).hasSizeLessThan(maxSegmentSize);
        String thirdParagraph = "This is a third paragraph.";
        assertThat(thirdParagraph).hasSizeLessThan(maxSegmentSize);
        assertThat(firstParagraph + secondParagraph + thirdParagraph)
                .hasSizeGreaterThan(maxSegmentSize);
        Document document = Document.from(
                format(" %s \n \n %s \n \n %s ", firstParagraph,
secondParagraph, thirdParagraph),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
```

```
DocumentByParagraphSplitter(maxSegmentSize, 0);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(firstParagraph + "\n\n" + secondParagraph,
metadata("index", "0").add("document", "0")),
                textSegment(thirdParagraph, metadata("index",
"1").add("document", "0"))
        );
    }
    @Test
    void
should_split_paragraph_into_sentences_if_it_does_not_fit_into_segment() {
        int maxSegmentSize = 50;
        String firstParagraph = "This is a first paragraph.";
        assertThat(firstParagraph).hasSizeLessThan(maxSegmentSize);
        String firstSentenceOfSecondParagraph = "This is a fist sentence of a
second paragraph.";

assertThat(firstSentenceOfSecondParagraph).hasSizeLessThan(maxSegmentSize);
        String secondSentenceOfSecondParagraph = "This is a second sentence
of a second paragraph.";

assertThat(secondSentenceOfSecondParagraph).hasSizeLessThan(maxSegmentSize);
        String secondParagraph = firstSentenceOfSecondParagraph + " " +
secondSentenceOfSecondParagraph;
        assertThat(secondParagraph).hasSizeGreaterThan(maxSegmentSize);
        String thirdParagraph = "This is a third paragraph.";
        assertThat(thirdParagraph).hasSizeLessThan(maxSegmentSize);
        Document document = Document.from(
                format(" %s \n \n %s \n \n %s ", firstParagraph,
secondParagraph, thirdParagraph),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentByParagraphSplitter(maxSegmentSize, 0);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(firstParagraph, metadata("index",
"0").add("document", "0")),
                textSegment(firstSentenceOfSecondParagraph, metadata("index",
"1").add("document", "0")),
                textSegment(secondSentenceOfSecondParagraph,
metadata("index", "2").add("document", "0")),
                textSegment(thirdParagraph, metadata("index",
"3").add("document", "0"))
        );
    }
    @Test
    void should_split_sample_text_containing_multiple_paragraphs() {
        int maxSegmentSize = 65;
        Tokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        String p1 = "In a small town nestled between two vast mountains,
there was a shop unlike any other. " +
                "A unique haven. " +
                "Visitors would often comment on its peculiar charm, always
```

```
slightly different from what they " +
                "remembered on their previous visits. " +
                "The store stood as a testament to the passage of time and
the ever-changing landscape of tales.";
        assertThat(tokenizer.estimateTokenCountInText(p1)).isEqualTo(62);
        String p2p1 = "Upon entering, the first thing to strike you was the
enormity of it all. " +
                "Every inch of space was occupied with books. " +
                "Some stood tall and regal on the highest shelves, looking as
if they had witnessed epochs come and go. " +
                "They were leather-bound, with pages yellowed by age.";
        assertThat(tokenizer.estimateTokenCountInText(p2p1)).isEqualTo(60);
        String p2p2 = "Others, smaller and brightly adorned, were reminiscent
of summer days and childhood laughter. " +
                "But these physical objects were mere vessels. " +
                "It was the stories inside that held power.";
        assertThat(tokenizer.estimateTokenCountInText(p2p2)).isEqualTo(33);
        String p3 = "Mrs. Jenkins ran the shop. " +
                "A mystery in her own right. " +
                "Her silver hair cascaded like a waterfall, and her eyes
seemed to see more than most. " +
                "With just a glance, she'd find the perfect story for you.";
        assertThat(tokenizer.estimateTokenCountInText(p3)).isEqualTo(47);
        String p4p1 = "One wet afternoon, Eli entered. " +
                "He was just a boy, lost in the vastness of the store. " +
                "Between the aisles, his small fingers danced on the spines
of books, feeling the heartbeat of " +
                "countless tales. " +
                "Then, a simple brown-covered book whispered to him.";
        assertThat(tokenizer.estimateTokenCountInText(p4p1)).isEqualTo(56);
        String p4p2 = "Without grandeur or pretense, it beckoned. " +
                "And he listened.";
        assertThat(tokenizer.estimateTokenCountInText(p4p2)).isEqualTo(15);
        String p5 = "He read. " +
                "And read. " +
                "The world around him melted.";
        assertThat(tokenizer.estimateTokenCountInText(p5)).isEqualTo(12);
        String p6 = "When Mrs. Jenkins approached, night had fallen. " +
                "She gently remarked, \"Books have a way of finding their
reader.\" " +
                "Eli simply nodded, understanding the profound truth in her
words.";
        assertThat(tokenizer.estimateTokenCountInText(p6)).isEqualTo(36);
        String p7 = "Some places and stories remain etched in our souls,
offering lessons and moments of sheer wonder. " +
                "They defy definition.";
        assertThat(tokenizer.estimateTokenCountInText(p7)).isEqualTo(23);
        Document document = Document.from(
                format("%s\n\n%s %s\n\n%s\n\n%s %s\n\n%s\n\n%s\n\n%s", p1,
p2p1, p2p2, p3, p4p1, p4p2, p5, p6, p7),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentByParagraphSplitter(maxSegmentSize, 0, tokenizer);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->
                assertThat(tokenizer.estimateTokenCountInText(segment.text()))
.isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(p1, metadata("index", "0").add("document", "0")),
                textSegment(p2p1, metadata("index", "1").add("document",
```

```
"0")),
                textSegment(p2p2, metadata("index", "2").add("document",
"0")),
                textSegment(p3, metadata("index", "3").add("document", "0")),
                textSegment(p4p1, metadata("index", "4").add("document",
"0")),
                textSegment(p4p2, metadata("index", "5").add("document",
"0")),
                textSegment(p5 + "\n\n" + p6, metadata("index",
"6").add("document", "0")),
                textSegment(p7, metadata("index", "7").add("document", "0"))
        );
    }
    @Test
    void
should_split_sample_text_containing_multiple_paragraphs_with_overlap() {
        int maxSegmentSize = 65;
        int maxOverlapSize = 15;
        Tokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        String s1 = "In a small town nestled between two vast mountains,
there was a shop unlike any other.";
        String s2 = "A unique haven.";
        String s3 = "Visitors would often comment on its peculiar charm,
always slightly different from what they remembered on their previous
visits.";
        String s4 = "The store stood as a testament to the passage of time
and the ever-changing landscape of tales.";
        String s5 = "Upon entering, the first thing to strike you was the
enormity of it all.";
        String s6 = "Every inch of space was occupied with books.";
        String s7 = "Some stood tall and regal on the highest shelves,
looking as if they had witnessed epochs come and go.";
        String s8 = "They were leather-bound, with pages yellowed by age.";
        String s9 = "Others, smaller and brightly adorned, were reminiscent
of summer days and childhood laughter.";
        String s10 = "But these physical objects were mere vessels.";
        String s11 = "It was the stories inside that held power.";
        String s12 = "Mrs. Jenkins ran the shop.";
        String s13 = "A mystery in her own right.";
        String s14 = "Her silver hair cascaded like a waterfall, and her eyes
seemed to see more than most.";
        String s15 = "With just a glance, she'd find the perfect story for
you.";
        String s16 = "One wet afternoon, Eli entered.";
        String s17 = "He was just a boy, lost in the vastness of the store.";
        String s18 = "Between the aisles, his small fingers danced on the
spines of books, feeling the heartbeat of countless tales.";
        String s19 = "Then, a simple brown-covered book whispered to him.";
        String s20 = "Without grandeur or pretense, it beckoned.";
        String s21 = "And he listened.";
        String s22 = "He read.";
        String s23 = "And read.";
        String s24 = "The world around him melted.";
        String s25 = "When Mrs. Jenkins approached, night had fallen.";
        String s26 = "She gently remarked, \"Books have a way of finding
their reader.\"";
        String s27 = "Eli simply nodded, understanding the profound truth in
her words.";
        String s28 = "Some places and stories remain etched in our souls,
offering lessons and moments of sheer wonder.";
        String s29 = "They defy definition.";
```

```
        Document document = Document.from(
                format("%s %s %s %s\n\n%s %s %s %s %s %s %s\n\n%s %s %s
%s\n\n%s %s %s %s %s %s %s %s\n\n%s %s %s %s %s",
                        s1, s2, s3, s4,
                        s5, s6, s7, s8, s9, s10, s11,
                        s12, s13, s14, s15,
                        s16, s17, s18, s19, s20, s21, s22, s23, s24,
                        s25, s26, s27, s28, s29
                ),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentByParagraphSplitter(maxSegmentSize, maxOverlapSize, tokenizer);
        List<TextSegment> segments = splitter.split(document);
        assertThat(segments).containsExactly(
                textSegment(format("%s %s %s %s", s1, s2, s3, s4),
metadata("index", "0").add("document", "0")),
                textSegment(format("%s %s %s %s", s5, s6, s7, s8),
metadata("index", "1").add("document", "0")),
                textSegment(format("%s %s %s %s", s8, s9, s10, s11),
metadata("index", "2").add("document", "0")),
                textSegment(format("%s\n\n%s %s %s %s", s11, s12, s13, s14,
s15), metadata("index", "3").add("document", "0")),
                textSegment(format("%s %s %s %s", s15, s16, s17, s18),
metadata("index", "4").add("document", "0")),
                textSegment(format("%s %s %s %s %s %s", s19, s20, s21, s22,
s23, s24), metadata("index", "5").add("document", "0")),
                textSegment(format("%s %s %s %s %s %s", s22, s23, s24, s25,
s26, s27), metadata("index", "6").add("document", "0")),
                textSegment(format("%s %s %s", s27, s28, s29),
metadata("index", "7").add("document", "0"))
        );
    }
    @Test
    void should_split_sample_text_without_paragraphs() {
        int maxSegmentSize = 100;
        Tokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        String segment1 = "In a small town nestled between two vast
mountains, there was a shop unlike any other. " +
                "A unique haven. " +
                "Visitors would often comment on its peculiar charm, always
slightly different from what they " +
                "remembered on their previous visits. " +
                "The store stood as a testament to the passage of time and
the ever-changing landscape of tales. " +
                "Upon entering, the first thing to strike you was the
enormity of it all. " +
                "Every inch of space was occupied with books.";
        String segment2 = "Some stood tall and regal on the highest shelves,
" +
                "looking as if they had witnessed epochs come and go. " +
                "They were leather-bound, with pages yellowed by age. " +
                "Others, smaller and brightly adorned, were reminiscent of
summer days and childhood laughter. " +
                "But these physical objects were mere vessels. " +
                "It was the stories inside that held power. " +
                "Mrs. Jenkins ran the shop. " +
                "A mystery in her own right.";
        String segment3 = "Her silver hair cascaded like a waterfall, and her
eyes seemed to see more than most. " +
                "With just a glance, she'd find the perfect story for you. " +
```

```
                "One wet afternoon, Eli entered. " +
                "He was just a boy, lost in the vastness of the store. " +
                "Between the aisles, his small fingers danced on the spines
of books, feeling the heartbeat of " +
                "countless tales. " +
                "Then, a simple brown-covered book whispered to him.";
        String segment4 = "Without grandeur or pretense, it beckoned. " +
                "And he listened. " +
                "He read. " +
                "And read. " +
                "The world around him melted. " +
                "When Mrs. Jenkins approached, night had fallen. " +
                "She gently remarked, \"Books have a way of finding their
reader.\" " +
                "Eli simply nodded, understanding the profound truth in her
words. " +
                "Some places and stories remain etched in our souls, offering
lessons and moments of sheer wonder. " +
                "They defy definition.";
        Document document = Document.from(
                format("%s %s %s %s", segment1, segment2, segment3, segment4),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentByParagraphSplitter(maxSegmentSize, 0, tokenizer);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->
                assertThat(tokenizer.estimateTokenCountInText(segment.text()))
.isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(segment1, metadata("index", "0").add("document",
"0")),
                textSegment(segment2, metadata("index", "1").add("document",
"0")),
                textSegment(segment3, metadata("index", "2").add("document",
"0")),
                textSegment(segment4, metadata("index", "3").add("document",
"0"))
        );
    }
    @Test
    void should_split_sample_text_without_paragraphs_with_overlap() {
        int maxSegmentSize = 100;
        int maxOverlapSize = 25;
        Tokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        String s1 = "In a small town nestled between two vast mountains,
there was a shop unlike any other.";
        String s2 = "A unique haven.";
        String s3 = "Visitors would often comment on its peculiar charm,
always slightly different from what they remembered on their previous
visits.";
        String s4 = "The store stood as a testament to the passage of time
and the ever-changing landscape of tales.";
        String s5 = "Upon entering, the first thing to strike you was the
enormity of it all.";
        String s6 = "Every inch of space was occupied with books.";
        String s7 = "Some stood tall and regal on the highest shelves,
looking as if they had witnessed epochs come and go.";
        String s8 = "They were leather-bound, with pages yellowed by age.";
        String s9 = "Others, smaller and brightly adorned, were reminiscent
of summer days and childhood laughter.";
```

```
        String s10 = "But these physical objects were mere vessels.";
        String s11 = "It was the stories inside that held power.";
        String s12 = "Mrs. Jenkins ran the shop.";
        String s13 = "A mystery in her own right.";
        String s14 = "Her silver hair cascaded like a waterfall, and her eyes
seemed to see more than most.";
        String s15 = "With just a glance, she'd find the perfect story for
you.";
        String s16 = "One wet afternoon, Eli entered.";
        String s17 = "He was just a boy, lost in the vastness of the store.";
        String s18 = "Between the aisles, his small fingers danced on the
spines of books, feeling the heartbeat of countless tales.";
        String s19 = "Then, a simple brown-covered book whispered to him.";
        String s20 = "Without grandeur or pretense, it beckoned.";
        String s21 = "And he listened.";
        String s22 = "He read.";
        String s23 = "And read.";
        String s24 = "The world around him melted.";
        String s25 = "When Mrs. Jenkins approached, night had fallen.";
        String s26 = "She gently remarked, \"Books have a way of finding
their reader.\"";
        String s27 = "Eli simply nodded, understanding the profound truth in
her words.";
        String s28 = "Some places and stories remain etched in our souls,
offering lessons and moments of sheer wonder.";
        String s29 = "They defy definition.";
        Document document = Document.from(
                format("%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s %s %s %s %s %s %s %s %s %s %s",
                        s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12,
s13, s14, s15, s16, s17, s18, s19, s20,
                        s21, s22, s23, s24, s25, s26, s27, s28, s29),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentByParagraphSplitter(maxSegmentSize, maxOverlapSize, tokenizer);
        List<TextSegment> segments = splitter.split(document);
        assertThat(segments).containsExactly(
                textSegment(format("%s %s %s %s %s %s", s1, s2, s3, s4, s5,
s6), metadata("index", "0").add("document", "0")),
                textSegment(format("%s %s %s %s %s %s %s %s", s6, s7, s8, s9,
s10, s11, s12, s13), metadata("index", "1").add("document", "0")),
                textSegment(format("%s %s %s %s %s %s %s", s11, s12, s13,
s14, s15, s16, s17), metadata("index", "2").add("document", "0")),
                textSegment(format("%s %s %s %s %s %s %s %s %s %s", s16, s17,
s18, s19, s20, s21, s22, s23, s24, s25), metadata("index",
"3").add("document", "0")),
                textSegment(format("%s %s %s %s %s %s %s %s", s22, s23, s24,
s25, s26, s27, s28, s29), metadata("index", "4").add("document", "0"))
        );
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\splitter\DocumentByRegexSplitterTest.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
import java.util.List;
import static dev.langchain4j.data.document.Metadata.metadata;
import static dev.langchain4j.data.segment.TextSegment.textSegment;
import static java.lang.String.format;
import static org.assertj.core.api.Assertions.assertThat;
class DocumentByRegexSplitterTest {
    @ParameterizedTest
    @ValueSource(strings = {" ", ",", "\n", "\n\n"})
    void should_split_by(String separator) {
        String text = format("one%stwo%sthree", separator, separator);
        Document document = Document.from(text, metadata("document", "0"));
        int maxSegmentSize = 5;
        DocumentSplitter splitter = new DocumentByRegexSplitter(separator,
separator, maxSegmentSize, 0);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment("one", metadata("index", "0").add("document",
"0")),
                textSegment("two", metadata("index", "1").add("document",
"0")),
                textSegment("three", metadata("index", "2").add("document",
"0"))
        );
    }
    @Test
    void should_fit_multiple_parts_into_the_same_segment() {
        Document document = Document.from("one two three",
metadata("document", "0"));
        int maxSegmentSize = 10;
        DocumentSplitter splitter = new DocumentByRegexSplitter(" ", "\n",
maxSegmentSize, 0);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment("one\ntwo", metadata("index",
"0").add("document", "0")),
                textSegment("three", metadata("index", "1").add("document",
"0"))
        );
    }
    @Test
    void should_split_part_into_sub_parts_if_it_does_not_fit_into_segment() {
        Document document = Document.from(
                "This is a first line.\nThis is a second line.\n\nThis is a
third line.",
```

```
                metadata("document", "0")
        );
        int maxSegmentSize = 15;
        DocumentSplitter subSplitter = new
DocumentByWordSplitter(maxSegmentSize, 0);
        DocumentSplitter splitter = new DocumentByRegexSplitter("\n", "\n",
maxSegmentSize, 0, subSplitter);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment("This is a first", metadata("index",
"0").add("document", "0")),
                textSegment("line.", metadata("index", "1").add("document",
"0")),
                textSegment("This is a", metadata("index",
"2").add("document", "0")),
                textSegment("second line.", metadata("index",
"3").add("document", "0")),
                textSegment("This is a third", metadata("index",
"4").add("document", "0")),
                textSegment("line.", metadata("index", "5").add("document",
"0"))
        );
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\splitter\DocumentBySentenceSplitterTest.java

```
package dev.langchain4j.data.document.splitter;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.data.document.Metadata.metadata;
import static dev.langchain4j.data.segment.TextSegment.textSegment;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static java.lang.String.format;
import static org.assertj.core.api.Assertions.assertThat;
class DocumentBySentenceSplitterTest {
    @Test
    void should_split_into_segments_with_one_sentence_per_segment() {
        int maxSegmentSize = 30;
        String firstSentence = "This is a first sentence.";
        assertThat(firstSentence).hasSizeLessThan(maxSegmentSize);
        String secondSentence = "This is a second sentence.";
        assertThat(secondSentence).hasSizeLessThan(maxSegmentSize);
        assertThat(firstSentence + " " +
secondSentence).hasSizeGreaterThan(maxSegmentSize);
        Document document = Document.from(
                format(" %s  %s ", firstSentence, secondSentence),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentBySentenceSplitter(maxSegmentSize, 0);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(firstSentence, metadata("index",
"0").add("document", "0")),
                textSegment(secondSentence, metadata("index",
"1").add("document", "0"))
        );
    }
    @Test
    void should_split_into_segments_with_multiple_sentences_per_segment() {
        int maxSegmentSize = 60;
        String firstSentence = "This is a first sentence.";
        String secondSentence = "This is a second sentence.";
        assertThat(firstSentence + " " +
secondSentence).hasSizeLessThan(maxSegmentSize);
        String thirdSentence = "This is a third sentence.";
        assertThat(firstSentence + " " + secondSentence + " " + thirdSentence)
                .hasSizeGreaterThan(maxSegmentSize);
        Document document = Document.from(
                format(" %s  %s  %s ", firstSentence, secondSentence,
thirdSentence),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentBySentenceSplitter(maxSegmentSize, 0);
        List<TextSegment> segments = splitter.split(document);
```

```
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(firstSentence + " " + secondSentence,
metadata("index", "0").add("document", "0")),
                textSegment(thirdSentence, metadata("index",
"1").add("document", "0"))
        );
    }
    @Test
    void should_split_sentence_if_it_does_not_fit_into_segment() {
        int maxSegmentSize = 40;
        String firstSentence = "This is a short sentence.";
        assertThat(firstSentence).hasSizeLessThan(maxSegmentSize);
        String secondSentence = "This is a very long sentence that does not
fit into segment.";
        assertThat(secondSentence).hasSizeGreaterThan(maxSegmentSize);
        String thirdSentence = "This is another short sentence.";
        assertThat(thirdSentence).hasSizeLessThan(maxSegmentSize);
        Document document = Document.from(
                format(" %s  %s  %s ", firstSentence, secondSentence,
thirdSentence),
                metadata("document", "0")
        );
        DocumentSplitter splitter = new
DocumentBySentenceSplitter(maxSegmentSize, 0);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->

assertThat(segment.text().length()).isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(firstSentence, metadata("index",
"0").add("document", "0")),
                textSegment("This is a very long sentence that does",
metadata("index", "1").add("document", "0")),
                textSegment("not fit into segment.", metadata("index",
"2").add("document", "0")),
                textSegment(thirdSentence, metadata("index",
"3").add("document", "0"))
        );
    }
    @Test
    void should_split_sample_text() {
        String s1 = "In a sleepy hamlet, where the trees towered high, there
lived a young boy named Elias.";
        String s2 = "He loved exploring.";
        String s3 = "Fields of gold stretched as far as the eye could see,
punctuated by tiny blossoms.";
        String s4 = "The wind whispered.";
        String s5p1 = "Sometimes, it would carry fragrances from the
neighboring towns, which included chocolate, " +
                "freshly baked bread, and the salty tang of";
        String s5p2 = "the sea.";
        String s6 = "In the middle of the town, a single lamppost stood.";
        String s7 = "Cats lounged beneath it, stretching languidly in the
dappled sunlight.";
        String s8 = "Elias had a dream: to build a flying machine.";
        String s9 = "Some days, it felt impossible.";
        String s10 = "Yet, every evening, he would pull out his sketches,
tinkering and toiling away.";
```

```
        String s11 = "There was a resilience in his spirit.";
        String s12 = "Birds often stopped to watch.";
        String s13 = "Curiosity is the spark of invention.";
        String s14 = "He believed.";
        String s15 = "And one day, with the town gathered around him, Elias
soared.";
        String s16 = "The horizon awaited.";
        String s17 = "Life is full of surprises.";
        String s18 = "Embrace them.";
        Document document = Document.from(
                format(
                        "%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s %s",
                        s1, s2, s3, s4, s5p1, s5p2, s6, s7, s8, s9, s10, s11,
s12, s13, s14, s15, s16, s17, s18
                ),
                metadata("document", "0")
        );
        int maxSegmentSize = 26;
        OpenAiTokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        DocumentSplitter splitter = new
DocumentBySentenceSplitter(maxSegmentSize, 0, tokenizer);
        List<TextSegment> segments = splitter.split(document);
        segments.forEach(segment ->
                assertThat(tokenizer.estimateTokenCountInText(segment.text()))
.isLessThanOrEqualTo(maxSegmentSize));
        assertThat(segments).containsExactly(
                textSegment(s1 + " " + s2, metadata("index",
0).add("document", "0")),
                textSegment(s3 + " " + s4, metadata("index",
1).add("document", "0")),
                textSegment(s5p1, metadata("index", 2).add("document", "0")),
                textSegment(s5p2, metadata("index", 3).add("document", "0")),
                textSegment(s6, metadata("index", 4).add("document", "0")),
                textSegment(s7, metadata("index", 5).add("document", "0")),
                textSegment(s8 + " " + s9, metadata("index",
6).add("document", "0")),
                textSegment(s10, metadata("index", 7).add("document", "0")),
                textSegment(s11 + " " + s12 + " " + s13 + " " + s14,
metadata("index", 8).add("document", "0")),
                textSegment(s15 + " " + s16 + " " + s17, metadata("index",
9).add("document", "0")),
                textSegment(s18, metadata("index", 10).add("document", "0"))
        );
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\transformer\HtmlTextExtractorTest.java

```
package dev.langchain4j.data.document.transformer;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.Metadata;
import org.junit.jupiter.api.Test;
import java.util.HashMap;
import java.util.Map;
import static dev.langchain4j.data.document.Document.DOCUMENT_TYPE;
import static dev.langchain4j.data.document.DocumentType.HTML;
import static org.assertj.core.api.Assertions.assertThat;
class HtmlTextExtractorTest {
    private static final String SAMPLE_HTML = "<html>" +
            "<body>" +
            "<h1 id=\"title\">Title</h1>" +
            "<p id=\"p1\">Paragraph 1<br>Something</p>" +
            "<p id=\"p2\">Paragraph 2</p>" +
            "<p id=\"p3\">More details <a href=\"http://example.org\">here</
a>.</p>" +
            "List:<ul><li>Item one</li><li>Item two</li></ul>" +
            "</body>" +
            "</html>";
    @Test
    void should_extract_all_text_from_html() {
        HtmlTextExtractor transformer = new HtmlTextExtractor();
        Document htmlDocument = Document.from(SAMPLE_HTML,
Metadata.from(DOCUMENT_TYPE, HTML));
        Document transformedDocument = transformer.transform(htmlDocument);
        assertThat(transformedDocument.text()).isEqualTo(
                "Title\n" +
                        "\n" +
                        "Paragraph 1\n" +
                        "Something\n" +
                        "\n" +
                        "Paragraph 2\n" +
                        "\n" +
                        "More details here.\n" +
                        "List:\n" +
                        " * Item one\n" +
                        " * Item two"
        );
    }
    @Test
    void should_extract_text_from_html_by_css_selector() {
        HtmlTextExtractor transformer = new HtmlTextExtractor("#p1", null,
false);
        Document htmlDocument = Document.from(SAMPLE_HTML,
Metadata.from(DOCUMENT_TYPE, HTML));
        Document transformedDocument = transformer.transform(htmlDocument);
        assertThat(transformedDocument.text()).isEqualTo("Paragraph
1\nSomething");
        assertThat(transformedDocument.metadata(DOCUMENT_TYPE)).isEqualTo(HTML
.toString());
    }
    @Test
    void should_extract_text_and_metadata_from_html_by_css_selectors() {
        Map<String, String> metadataCssSelectors = new HashMap<>();
        metadataCssSelectors.put("title", "#title");
        HtmlTextExtractor transformer = new HtmlTextExtractor("#p1",
```

```
metadataCssSelectors, false);
        Document htmlDocument = Document.from(SAMPLE_HTML,
Metadata.from(DOCUMENT_TYPE, HTML));
        Document transformedDocument = transformer.transform(htmlDocument);
        assertThat(transformedDocument.text()).isEqualTo("Paragraph
1\nSomething");
        assertThat(transformedDocument.metadata().asMap()).hasSize(2);
        assertThat(transformedDocument.metadata(DOCUMENT_TYPE)).isEqualTo(HTML
.toString());
        assertThat(transformedDocument.metadata("title")).isEqualTo("Title");
    }
    @Test
    void should_extract_text_with_links_from_html() {
        HtmlTextExtractor transformer = new HtmlTextExtractor(null, null,
true);
        Document htmlDocument = Document.from(SAMPLE_HTML,
Metadata.from(DOCUMENT_TYPE, HTML));
        Document transformedDocument = transformer.transform(htmlDocument);
        assertThat(transformedDocument.text()).isEqualTo(
                "Title\n" +
                        "\n" +
                        "Paragraph 1\n" +
                        "Something\n" +
                        "\n" +
                        "Paragraph 2\n" +
                        "\n" +
                        "More details here <http://example.org>.\n" +
                        "List:\n" +
                        " * Item one\n" +
                        " * Item two"
        );
        assertThat(transformedDocument.metadata(DOCUMENT_TYPE)).isEqualTo(HTML
.toString());
    }
}
```

langchain4j\src\test\java\dev\langchain4j\data\document\UrlDocumentLoaderTest.java

```
package dev.langchain4j.data.document;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class UrlDocumentLoaderTest {
    @Test
    void should_load_text_document() {
        String url = "https://raw.githubusercontent.com/langchain4j/
langchain4j/main/langchain4j/src/test/resources/test-file-utf8.txt";
        Document document = UrlDocumentLoader.load(url);
        assertThat(document.text()).isEqualTo("test\ncontent");
        Metadata metadata = document.metadata();
        assertThat(metadata.get("url")).isEqualTo(url);
    }
}
```

langchain4j\src\test\java\dev\langchain4j\internal\RetryUtilsTest.java

```
package dev.langchain4j.internal;
import org.junit.jupiter.api.Test;
import java.util.concurrent.Callable;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
import static org.mockito.Mockito.*;
class RetryUtilsTest {
    @Test
    void testSuccessfulCall() throws Exception {
        Callable<String> mockAction = mock(Callable.class);
        when(mockAction.call()).thenReturn("Success");
        String result = withRetry(mockAction, 3);
        assertThat(result).isEqualTo("Success");
        verify(mockAction).call();
        verifyNoMoreInteractions(mockAction);
    }
    @Test
    void testRetryThenSuccess() throws Exception {
        Callable<String> mockAction = mock(Callable.class);
        when(mockAction.call())
                .thenThrow(new RuntimeException())
                .thenReturn("Success");
        long startTime = System.currentTimeMillis();
        String result = withRetry(mockAction, 3);
        long endTime = System.currentTimeMillis();
        long duration = endTime - startTime;
        assertThat(result).isEqualTo("Success");
        verify(mockAction, times(2)).call();
        verifyNoMoreInteractions(mockAction);
        assertThat(duration).isGreaterThanOrEqualTo(1000);
    }
    @Test
    void testMaxAttemptsReached() throws Exception {
        Callable<String> mockAction = mock(Callable.class);
        when(mockAction.call()).thenThrow(new RuntimeException());
        assertThatThrownBy(() -> withRetry(mockAction, 3))
                .isInstanceOf(RuntimeException.class);
        verify(mockAction, times(3)).call();
        verifyNoMoreInteractions(mockAction);
    }
}
```

langchain4j\src\test\java\dev\langchain4j\internal\TestUtils.java

```
package dev.langchain4j.internal;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import lombok.val;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
import java.util.ArrayList;
import java.util.List;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static dev.langchain4j.data.message.SystemMessage.systemMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static org.assertj.core.api.Assertions.assertThat;
public class TestUtils {
    private static final int EXTRA_TOKENS_PER_EACH_MESSAGE =
            3 /* extra tokens for each message */ + 1 /* extra token for
'role' */;
    private static final OpenAiTokenizer TOKENIZER = new
OpenAiTokenizer(GPT_3_5_TURBO);
    @ParameterizedTest
    @ValueSource(ints = {5, 10, 25, 50, 100, 250, 500, 1000})
    void should_create_system_message_with_tokens(int numberOfTokens) {
        SystemMessage systemMessage = systemMessageWithTokens(numberOfTokens);
        assertThat(TOKENIZER.estimateTokenCountInMessage(systemMessage)).isEqu
alTo(numberOfTokens);
    }
    public static SystemMessage systemMessageWithTokens(int numberOfTokens) {
        return systemMessage(textWithTokens(numberOfTokens -
EXTRA_TOKENS_PER_EACH_MESSAGE));
    }
    @ParameterizedTest
    @ValueSource(ints = {5, 10, 25, 50, 100, 250, 500, 1000})
    void should_create_user_message_with_tokens(int numberOfTokens) {
        UserMessage userMessage = userMessageWithTokens(numberOfTokens);
        assertThat(TOKENIZER.estimateTokenCountInMessage(userMessage)).isEqual
To(numberOfTokens);
    }
    public static UserMessage userMessageWithTokens(int numberOfTokens) {
        return userMessage(textWithTokens(numberOfTokens -
EXTRA_TOKENS_PER_EACH_MESSAGE));
    }
    @ParameterizedTest
    @ValueSource(ints = {5, 10, 25, 50, 100, 250, 500, 1000})
    void should_create_ai_message_with_tokens(int numberOfTokens) {
        AiMessage aiMessage = aiMessageWithTokens(numberOfTokens);
        assertThat(TOKENIZER.estimateTokenCountInMessage(aiMessage)).isEqualTo
(numberOfTokens);
    }
    public static AiMessage aiMessageWithTokens(int numberOfTokens) {
        return aiMessage(textWithTokens(numberOfTokens -
EXTRA_TOKENS_PER_EACH_MESSAGE));
    }
    @ParameterizedTest
    @ValueSource(ints = {1, 2, 5, 10, 25, 50, 100, 250, 500, 1000})
    void should_generate_tokens(int numberOfTokens) {
```

```
        String text = textWithTokens(numberOfTokens);
        assertThat(TOKENIZER.estimateTokenCountInText(text)).isEqualTo(numberO
fTokens);
    }
    private static String textWithTokens(int n) {
        String text = String.join(" ", repeat("one two", n));
        return TOKENIZER.decode(TOKENIZER.encode(text, n));
    }
    @Test
    void should_repeat_n_times() {
        assertThat(repeat("word", 1))
                .hasSize(1)
                .containsExactly("word");
        assertThat(repeat("word", 2))
                .hasSize(2)
                .containsExactly("word", "word");
        assertThat(repeat("word", 3))
                .hasSize(3)
                .containsExactly("word", "word", "word");
    }
    public static List<String> repeat(String s, int n) {
        val result = new ArrayList<String>();
        for (int i = 0; i < n; i++) {
            result.add(s);
        }
        return result;
    }
}
```

langchain4j\src\test\java\dev\langchain4j\memory\chat\MessageWindowChatMemoryTest.java

```
package dev.langchain4j.memory.chat;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.memory.ChatMemory;
import org.junit.jupiter.api.Test;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static dev.langchain4j.data.message.SystemMessage.systemMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static org.assertj.core.api.Assertions.assertThat;
class MessageWindowChatMemoryTest {
    @Test
    void should_keep_specified_number_of_messages_in_chat_memory() {
        ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(3);
        UserMessage firstUserMessage = userMessage("hello");
        chatMemory.add(firstUserMessage);
        assertThat(chatMemory.messages())
                .hasSize(1)
                .containsExactly(firstUserMessage);
        AiMessage firstAiMessage = aiMessage("hi");
        chatMemory.add(firstAiMessage);
        assertThat(chatMemory.messages())
                .hasSize(2)
                .containsExactly(firstUserMessage, firstAiMessage);
        UserMessage secondUserMessage = userMessage("sup");
        chatMemory.add(secondUserMessage);
        assertThat(chatMemory.messages())
                .hasSize(3)
                .containsExactly(
                        firstUserMessage,
                        firstAiMessage,
                        secondUserMessage
                );
        AiMessage secondAiMessage = aiMessage("not much");
        chatMemory.add(secondAiMessage);
        assertThat(chatMemory.messages())
                .hasSize(3)
                .containsExactly(
                        // firstUserMessage was removed
                        firstAiMessage,
                        secondUserMessage,
                        secondAiMessage
                );
    }
    @Test
    void should_not_remove_system_message_from_chat_memory() {
        ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(3);
        SystemMessage systemMessage = systemMessage("You are a helpful
assistant");
        chatMemory.add(systemMessage);
        UserMessage firstUserMessage = userMessage("Hello");
        chatMemory.add(firstUserMessage);
        AiMessage firstAiMessage = aiMessage("Hi, how can I help you?");
        chatMemory.add(firstAiMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                firstUserMessage,
```

```
                firstAiMessage
        );
        UserMessage secondUserMessage = userMessage("Tell me a joke");
        chatMemory.add(secondUserMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                // firstUserMessage was removed
                firstAiMessage,
                secondUserMessage
        );
        AiMessage secondAiMessage = aiMessage("Why did the Java developer
wear glasses? Because they didn't see sharp!");
        chatMemory.add(secondAiMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                // firstAiMessage was removed
                secondUserMessage,
                secondAiMessage
        );
    }
    @Test
    void should_keep_only_the_latest_system_message_in_chat_memory() {
        ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(3);
        SystemMessage firstSystemMessage = systemMessage("You are a helpful
assistant");
        chatMemory.add(firstSystemMessage);
        UserMessage firstUserMessage = userMessage("Hello");
        chatMemory.add(firstUserMessage);
        AiMessage firstAiMessage = aiMessage("Hi, how can I help you?");
        chatMemory.add(firstAiMessage);
        assertThat(chatMemory.messages()).containsExactly(
                firstSystemMessage,
                firstUserMessage,
                firstAiMessage
        );
        SystemMessage secondSystemMessage = systemMessage("You are an
unhelpful assistant");
        chatMemory.add(secondSystemMessage);
        assertThat(chatMemory.messages()).containsExactly(
                // firstSystemMessage was removed
                firstUserMessage,
                firstAiMessage,
                secondSystemMessage
        );
    }
    @Test
    void should_not_add_the_same_system_message_to_chat_memory_if_it_is_alread
y_there() {
        ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(3);
        SystemMessage systemMessage = systemMessage("You are a helpful
assistant");
        chatMemory.add(systemMessage);
        UserMessage userMessage = userMessage("Hello");
        chatMemory.add(userMessage);
        AiMessage aiMessage = aiMessage("Hi, how can I help you?");
        chatMemory.add(aiMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                userMessage,
                aiMessage
        );
```

```
        chatMemory.add(systemMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                userMessage,
                aiMessage
        );
    }
}
```

langchain4j\src\test\java\dev\langchain4j\memory\chat\TokenWindowChatMemoryTest.java

```
package dev.langchain4j.memory.chat;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import org.junit.jupiter.api.Test;
import static dev.langchain4j.data.message.SystemMessage.systemMessage;
import static dev.langchain4j.internal.TestUtils.*;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static org.assertj.core.api.Assertions.assertThat;
class TokenWindowChatMemoryTest {
    @Test
    void should_keep_specified_number_of_tokens_in_chat_memory() {
        OpenAiTokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        ChatMemory chatMemory = TokenWindowChatMemory.withMaxTokens(33,
tokenizer);
        UserMessage firstUserMessage = userMessageWithTokens(10);
        chatMemory.add(firstUserMessage);
        assertThat(chatMemory.messages()).containsExactly(firstUserMessage);
        // @formatter:off
        assertThat(tokenizer.estimateTokenCountInMessages(chatMemory.messages(
))).isEqualTo(
                  10 // firstUserMessage
                + 3  // overhead
        );
        // @formatter:on
        AiMessage firstAiMessage = aiMessageWithTokens(10);
        chatMemory.add(firstAiMessage);
        assertThat(chatMemory.messages()).containsExactly(firstUserMessage,
firstAiMessage);
        // @formatter:off
        assertThat(tokenizer.estimateTokenCountInMessages(chatMemory.messages(
))).isEqualTo(
                  10 // firstUserMessage
                + 10 // firstAiMessage
                + 3  // overhead
        );
        // @formatter:on
        UserMessage secondUserMessage = userMessageWithTokens(10);
        chatMemory.add(secondUserMessage);
        assertThat(chatMemory.messages()).containsExactly(
                firstUserMessage,
                firstAiMessage,
                secondUserMessage
        );
        // @formatter:off
        assertThat(tokenizer.estimateTokenCountInMessages(chatMemory.messages(
))).isEqualTo(
                  10 // firstUserMessage
                + 10 // firstAiMessage
                + 10 // secondUserMessage
                + 3  // overhead
        );
        // @formatter:on
        AiMessage secondAiMessage = aiMessageWithTokens(10);
        chatMemory.add(secondAiMessage);
```

```
        assertThat(chatMemory.messages()).containsExactly(
                // firstUserMessage was removed
                firstAiMessage,
                secondUserMessage,
                secondAiMessage
        );
        // @formatter:off
        assertThat(tokenizer.estimateTokenCountInMessages(chatMemory.messages(
))).isEqualTo(
                 10 // firstAiMessage
               + 10 // secondUserMessage
               + 10 // secondAiMessage
               + 3  // overhead
        );
        // @formatter:on
    }
    @Test
    void should_not_remove_system_message_from_chat_memory() {
        OpenAiTokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        ChatMemory chatMemory = TokenWindowChatMemory.withMaxTokens(33,
tokenizer);
        SystemMessage systemMessage = systemMessageWithTokens(10);
        chatMemory.add(systemMessage);
        UserMessage firstUserMessage = userMessageWithTokens(10);
        chatMemory.add(firstUserMessage);
        AiMessage firstAiMessage = aiMessageWithTokens(10);
        chatMemory.add(firstAiMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                firstUserMessage,
                firstAiMessage
        );
        UserMessage secondUserMessage = userMessageWithTokens(10);
        chatMemory.add(secondUserMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                // firstUserMessage was removed
                firstAiMessage,
                secondUserMessage
        );
        AiMessage secondAiMessage = aiMessageWithTokens(10);
        chatMemory.add(secondAiMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                // firstAiMessage was removed
                secondUserMessage,
                secondAiMessage
        );
    }
    @Test
    void should_keep_only_the_latest_system_message_in_chat_memory() {
        OpenAiTokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        ChatMemory chatMemory = TokenWindowChatMemory.withMaxTokens(40,
tokenizer);
        SystemMessage firstSystemMessage = systemMessage("You are a helpful
assistant");
        chatMemory.add(firstSystemMessage);
        UserMessage firstUserMessage = userMessageWithTokens(10);
        chatMemory.add(firstUserMessage);
        AiMessage firstAiMessage = aiMessageWithTokens(10);
        chatMemory.add(firstAiMessage);
```

```
        assertThat(chatMemory.messages()).containsExactly(
                firstSystemMessage,
                firstUserMessage,
                firstAiMessage
        );
        SystemMessage secondSystemMessage = systemMessage("You are an
unhelpful assistant");
        chatMemory.add(secondSystemMessage);
        assertThat(chatMemory.messages()).containsExactly(
                // firstSystemMessage was removed
                firstUserMessage,
                firstAiMessage,
                secondSystemMessage
        );
    }
    @Test
    void should_not_add_the_same_system_message_to_chat_memory_if_it_is_alread
y_there() {
        OpenAiTokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        ChatMemory chatMemory = TokenWindowChatMemory.withMaxTokens(33,
tokenizer);
        SystemMessage systemMessage = systemMessageWithTokens(10);
        chatMemory.add(systemMessage);
        UserMessage userMessage = userMessageWithTokens(10);
        chatMemory.add(userMessage);
        AiMessage aiMessage = aiMessageWithTokens(10);
        chatMemory.add(aiMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                userMessage,
                aiMessage
        );
        chatMemory.add(systemMessage);
        assertThat(chatMemory.messages()).containsExactly(
                systemMessage,
                userMessage,
                aiMessage
        );
    }
}
```

langchain4j\src\test\java\dev\langchain4j\service\AiServicesIT.java

```
package dev.langchain4j.service;
import dev.langchain4j.agent.tool.JsonSchemaProperty;
import dev.langchain4j.agent.tool.P;
import dev.langchain4j.agent.tool.Tool;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.ToolExecutionResultMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.ChatMemoryProvider;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.input.structured.StructuredPrompt;
import dev.langchain4j.model.moderation.ModerationModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.openai.OpenAiModerationModel;
import dev.langchain4j.model.output.structured.Description;
import dev.langchain4j.store.memory.chat.ChatMemoryStore;
import lombok.Builder;
import lombok.ToString;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Spy;
import org.mockito.junit.jupiter.MockitoExtension;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static dev.langchain4j.agent.tool.JsonSchemaProperty.NUMBER;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static
dev.langchain4j.data.message.ChatMessageDeserializer.messagesFromJson;
import static
dev.langchain4j.data.message.ChatMessageSerializer.messagesToJson;
import static dev.langchain4j.data.message.SystemMessage.systemMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.service.AiServicesIT.Sentiment.POSITIVE;
import static java.time.Month.JULY;
import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
import static org.mockito.Mockito.*;
@ExtendWith(MockitoExtension.class)
public class AiServicesIT {
    @Spy
    ChatLanguageModel chatLanguageModel = OpenAiChatModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .temperature(0.0)
            .logRequests(true)
            .logResponses(true)
            .build();
    @Spy
    ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(10);
    @Spy
```

```
    ModerationModel moderationModel = OpenAiModerationModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .build();
    @AfterEach
    void afterEach() {
        verifyNoMoreInteractions(chatLanguageModel);
        verifyNoMoreInteractions(chatMemory);
        verifyNoMoreInteractions(moderationModel);
    }
    interface Humorist {
        @UserMessage("Tell me a joke about {{it}}")
        String joke(String topic);
    }
    @Test
    void test_simple_instruction_with_single_argument() {
        Humorist humorist = AiServices.create(Humorist.class,
chatLanguageModel);
        String joke = humorist.joke("AI");
        assertThat(joke).isNotBlank();
        System.out.println(joke);
        verify(chatLanguageModel).generate(singletonList(userMessage("Tell me
a joke about AI")));
    }
    interface DateTimeExtractor {
        @UserMessage("Extract date from {{it}}")
        LocalDate extractDateFrom(String text);
        @UserMessage("Extract time from {{it}}")
        LocalTime extractTimeFrom(String text);
        @UserMessage("Extract date and time from {{it}}")
        LocalDateTime extractDateTimeFrom(String text);
    }
    @Test
    void test_extract_date() {
        DateTimeExtractor dateTimeExtractor =
AiServices.create(DateTimeExtractor.class, chatLanguageModel);
        String text = "The tranquility pervaded the evening of 1968, just
fifteen minutes shy of midnight, following the celebrations of Independence
Day.";
        LocalDate date = dateTimeExtractor.extractDateFrom(text);
        assertThat(date).isEqualTo(LocalDate.of(1968, JULY, 4));
        verify(chatLanguageModel).generate(singletonList(userMessage(
                "Extract date from " + text + "\n" +
                        "You must answer strictly in the following format:
2023-12-31")));
    }
    @Test
    void test_extract_time() {
        DateTimeExtractor dateTimeExtractor =
AiServices.create(DateTimeExtractor.class, chatLanguageModel);
        String text = "The tranquility pervaded the evening of 1968, just
fifteen minutes shy of midnight, following the celebrations of Independence
Day.";
        LocalTime time = dateTimeExtractor.extractTimeFrom(text);
        assertThat(time).isEqualTo(LocalTime.of(23, 45, 0));
        verify(chatLanguageModel).generate(singletonList(userMessage(
                "Extract time from " + text + "\n" +
                        "You must answer strictly in the following format:
23:59:59")));
    }
    @Test
    void test_extract_date_time() {
```

```
        DateTimeExtractor dateTimeExtractor =
AiServices.create(DateTimeExtractor.class, chatLanguageModel);
        String text = "The tranquility pervaded the evening of 1968, just
fifteen minutes shy of midnight, following the celebrations of Independence
Day.";
        LocalDateTime dateTime = dateTimeExtractor.extractDateTimeFrom(text);
        assertThat(dateTime).isEqualTo(LocalDateTime.of(1968, JULY, 4, 23,
45, 0));
        verify(chatLanguageModel).generate(singletonList(userMessage(
                "Extract date and time from " + text + "\n" +
                        "You must answer strictly in the following format:
2023-12-31T23:59:59")));
    }
    enum Sentiment {
        POSITIVE, NEUTRAL, NEGATIVE
    }
    interface SentimentAnalyzer {
        @UserMessage("Analyze sentiment of {{it}}")
        Sentiment analyzeSentimentOf(String text);
    }
    @Test
    void test_extract_enum() {
        SentimentAnalyzer sentimentAnalyzer =
AiServices.create(SentimentAnalyzer.class, chatLanguageModel);
        String customerReview = "This LaptopPro X15 is wicked fast and that
4K screen is a dream.";
        Sentiment sentiment =
sentimentAnalyzer.analyzeSentimentOf(customerReview);
        assertThat(sentiment).isEqualTo(POSITIVE);
        verify(chatLanguageModel).generate(singletonList(userMessage(
                "Analyze sentiment of " + customerReview + "\n" +
                        "You must answer strictly in the following format:
one of [POSITIVE, NEUTRAL, NEGATIVE]")));
    }
    static class Person {
        private String firstName;
        private String lastName;
        private LocalDate birthDate;
    }
    interface PersonExtractor {
        @UserMessage("Extract information about a person from {{it}}")
        Person extractPersonFrom(String text);
    }
    @Test
    void test_extract_custom_POJO() {
        PersonExtractor personExtractor =
AiServices.create(PersonExtractor.class, chatLanguageModel);
        String text = "In 1968, amidst the fading echoes of Independence Day,
"
                + "a child named John arrived under the calm evening sky. "
                + "This newborn, bearing the surname Doe, marked the start of
a new journey.";
        Person person = personExtractor.extractPersonFrom(text);
        assertThat(person.firstName).isEqualTo("John");
        assertThat(person.lastName).isEqualTo("Doe");
        assertThat(person.birthDate).isEqualTo(LocalDate.of(1968, JULY, 4));
        verify(chatLanguageModel).generate(singletonList(userMessage(
                "Extract information about a person from " + text + "\n" +
                        "You must answer strictly in the following JSON
format: {\n" +
                        "\"firstName\": (type: string),\n" +
```

```
                        "\"lastName\": (type: string),\n" +
                        "\"birthDate\": (type: date string (2023-12-31)),\n" +
                        "}")));
    }
    @ToString
    static class Recipe {
        private String title;
        private String description;
        @Description("each step should be described in 4 words, steps should
rhyme")
        private String[] steps;
        private Integer preparationTimeMinutes;
    }
    interface Chef {
        @UserMessage("Create recipe using only {{it}}")
        Recipe createRecipeFrom(String... ingredients);
        Recipe createRecipeFrom(CreateRecipePrompt prompt);
        @SystemMessage("You are very {{character}} chef")
        Recipe createRecipeFrom(@UserMessage CreateRecipePrompt prompt,
@V("character") String character);
    }
    @Test
    void test_create_recipe_from_list_of_ingredients() {
        Chef chef = AiServices.create(Chef.class, chatLanguageModel);
        Recipe recipe = chef.createRecipeFrom("cucumber", "tomato", "feta",
"onion", "olives");
        assertThat(recipe.title).isNotBlank();
        assertThat(recipe.description).isNotBlank();
        assertThat(recipe.steps).isNotEmpty();
        assertThat(recipe.preparationTimeMinutes).isPositive();
        System.out.println(recipe);
        verify(chatLanguageModel).generate(singletonList(userMessage(
                "Create recipe using only [cucumber, tomato, feta, onion,
olives]\n" +
                        "You must answer strictly in the following JSON
format: {\n" +
                        "\"title\": (type: string),\n" +
                        "\"description\": (type: string),\n" +
                        "\"steps\": (each step should be described in 4
words, steps should rhyme; type: array of string),\n" +
                        "\"preparationTimeMinutes\": (type: integer),\n" +
                        "}")));
    }
    @Builder
    @StructuredPrompt("Create a recipe of a {{dish}} that can be prepared
using only {{ingredients}}")
    static class CreateRecipePrompt {
        private String dish;
        private List<String> ingredients;
    }
    @Test
    void test_create_recipe_using_structured_prompt() {
        Chef chef = AiServices.create(Chef.class, chatLanguageModel);
        CreateRecipePrompt prompt = CreateRecipePrompt.builder()
                .dish("salad")
                .ingredients(asList("cucumber", "tomato", "feta", "onion",
"olives"))
                .build();
        Recipe recipe = chef.createRecipeFrom(prompt);
        assertThat(recipe.title).isNotBlank();
        assertThat(recipe.description).isNotBlank();
```

```
        assertThat(recipe.steps).isNotEmpty();
        assertThat(recipe.preparationTimeMinutes).isPositive();
        System.out.println(recipe);
        verify(chatLanguageModel).generate(singletonList(userMessage(
                "Create a recipe of a salad that can be prepared using only
[cucumber, tomato, feta, onion, olives]\n" +
                        "You must answer strictly in the following JSON
format: {\n" +
                        "\"title\": (type: string),\n" +
                        "\"description\": (type: string),\n" +
                        "\"steps\": (each step should be described in 4
words, steps should rhyme; type: array of string),\n" +
                        "\"preparationTimeMinutes\": (type: integer),\n" +
                        "}")));
    }
    @Test
    void test_create_recipe_using_structured_prompt_and_system_message() {
        Chef chef = AiServices.create(Chef.class, chatLanguageModel);
        CreateRecipePrompt prompt = CreateRecipePrompt
                .builder()
                .dish("salad")
                .ingredients(asList("cucumber", "tomato", "feta", "onion",
"olives"))
                .build();
        Recipe recipe = chef.createRecipeFrom(prompt, "funny");
        assertThat(recipe.title).isNotBlank();
        assertThat(recipe.description).isNotBlank();
        assertThat(recipe.steps).isNotEmpty();
        assertThat(recipe.preparationTimeMinutes).isPositive();
        System.out.println(recipe);
        verify(chatLanguageModel).generate(asList(
                systemMessage("You are very funny chef"),
                userMessage("Create a recipe of a salad that can be prepared
using only [cucumber, tomato, feta, onion, olives]\n" +
                        "You must answer strictly in the following JSON
format: {\n" +
                        "\"title\": (type: string),\n" +
                        "\"description\": (type: string),\n" +
                        "\"steps\": (each step should be described in 4
words, steps should rhyme; type: array of string),\n" +
                        "\"preparationTimeMinutes\": (type: integer),\n" +
                        "}")
        ));
    }
    interface ProfessionalChef {
        @SystemMessage("You are a professional chef. You are friendly, polite
and concise.")
        String answer(String question);
    }
    @Test
    void test_with_system_message() {
        ProfessionalChef chef = AiServices.create(ProfessionalChef.class,
chatLanguageModel);
        String question = "How long should I grill chicken?";
        String answer = chef.answer(question);
        assertThat(answer).isNotBlank();
        System.out.println(answer);
        verify(chatLanguageModel).generate(asList(
                systemMessage("You are a professional chef. You are friendly,
polite and concise."),
                userMessage(question)
```

```
        ));
    }
    interface Translator {
        @SystemMessage("You are a professional translator into {{language}}")
        @UserMessage("Translate the following text: {{text}}")
        String translate(@V("text") String text, @V("language") String
language);
    }
    @Test
    void test_with_system_and_user_messages() {
        Translator translator = AiServices.create(Translator.class,
chatLanguageModel);
        String text = "Hello, how are you?";
        String translation = translator.translate(text, "german");
        assertThat(translation).isEqualTo("Hallo, wie geht es dir?");
        verify(chatLanguageModel).generate(asList(
                systemMessage("You are a professional translator into
german"),
                userMessage("Translate the following text: Hello, how are
you?")
        ));
    }
    interface Summarizer {
        @SystemMessage("Summarize every message from user in {{n}} bullet
points. Provide only bullet points.")
        List<String> summarize(@UserMessage String text, @V("n") int n);
    }
    @Test
    void test_with_system_message_and_user_message_as_argument() {
        Summarizer summarizer = AiServices.create(Summarizer.class,
chatLanguageModel);
        String text = "AI, or artificial intelligence, is a branch of
computer science that aims to create " +
                "machines that mimic human intelligence. This can range from
simple tasks such as recognizing " +
                "patterns or speech to more complex tasks like making
decisions or predictions.";
        List<String> bulletPoints = summarizer.summarize(text, 3);
        assertThat(bulletPoints).hasSize(3);
        System.out.println(bulletPoints);
        verify(chatLanguageModel).generate(asList(
                systemMessage("Summarize every message from user in 3 bullet
points. Provide only bullet points."),
                userMessage(text + "\nYou must put every item on a separate
line.")
        ));
    }
    interface ChatWithModeration {
        @Moderate
        String chat(String message);
    }
    @Test
    void should_throw_when_text_is_flagged() {
        ChatWithModeration chatWithModeration =
AiServices.builder(ChatWithModeration.class)
                .chatLanguageModel(chatLanguageModel)
                .moderationModel(moderationModel)
                .build();
        String message = "I WILL KILL YOU!!!";
        assertThatThrownBy(() -> chatWithModeration.chat(message))
                .isExactlyInstanceOf(ModerationException.class)
```

```
                .hasMessage("Text \"" + message + "\" violates content
policy");

verify(chatLanguageModel).generate(singletonList(userMessage(message)));
        verify(moderationModel).moderate(singletonList(userMessage(message)));
    }
    @Test
    void should_not_throw_when_text_is_not_flagged() {
        ChatWithModeration chatWithModeration =
AiServices.builder(ChatWithModeration.class)
                .chatLanguageModel(chatLanguageModel)
                .moderationModel(moderationModel)
                .build();
        String message = "I will hug them!";
        String response = chatWithModeration.chat(message);
        assertThat(response).isNotBlank();

verify(chatLanguageModel).generate(singletonList(userMessage(message)));
        verify(moderationModel).moderate(singletonList(userMessage(message)));
    }
    interface ChatWithMemory {
        String chatWithoutSystemMessage(String userMessage);
        @SystemMessage("You are helpful assistant")
        String chatWithSystemMessage(String userMessage);
        @SystemMessage("You are funny assistant")
        String chatWithAnotherSystemMessage(String userMessage);
    }
    @Test
    void should_keep_chat_memory() {
        ChatWithMemory chatWithMemory =
AiServices.builder(ChatWithMemory.class)
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(chatMemory)
                .build();
        String firstUserMessage = "Hello, my name is Klaus";
        String firstAiMessage =
chatWithMemory.chatWithoutSystemMessage(firstUserMessage);
        verify(chatMemory).add(userMessage(firstUserMessage));
        verify(chatLanguageModel).generate(singletonList(userMessage(firstUser
Message)));
        verify(chatMemory).add(aiMessage(firstAiMessage));
        String secondUserMessage = "What is my name?";
        String secondAiMessage =
chatWithMemory.chatWithoutSystemMessage(secondUserMessage);
        assertThat(secondAiMessage).contains("Klaus");
        verify(chatMemory).add(userMessage(secondUserMessage));
        verify(chatLanguageModel).generate(asList(
                userMessage(firstUserMessage),
                aiMessage(firstAiMessage),
                userMessage(secondUserMessage)
        ));
        verify(chatMemory).add(aiMessage(secondAiMessage));
        verify(chatMemory, times(6)).messages();
    }
    @Test
    void should_keep_chat_memory_and_not_duplicate_system_message() {
        ChatWithMemory chatWithMemory =
AiServices.builder(ChatWithMemory.class)
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(chatMemory)
                .build();
```

```
        String systemMessage = "You are helpful assistant";
        String firstUserMessage = "Hello, my name is Klaus";
        String firstAiMessage =
chatWithMemory.chatWithSystemMessage(firstUserMessage);
        verify(chatLanguageModel).generate(asList(
                systemMessage(systemMessage),
                userMessage(firstUserMessage)
        ));
        String secondUserMessage = "What is my name?";
        String secondAiMessage =
chatWithMemory.chatWithSystemMessage(secondUserMessage);
        assertThat(secondAiMessage).contains("Klaus");
        verify(chatLanguageModel).generate(asList(
                systemMessage(systemMessage),
                userMessage(firstUserMessage),
                aiMessage(firstAiMessage),
                userMessage(secondUserMessage)
        ));
        verify(chatMemory, times(2)).add(systemMessage(systemMessage));
        verify(chatMemory).add(userMessage(firstUserMessage));
        verify(chatMemory).add(aiMessage(firstAiMessage));
        verify(chatMemory).add(userMessage(secondUserMessage));
        verify(chatMemory).add(aiMessage(secondAiMessage));
        verify(chatMemory, times(8)).messages();
    }
    @Test
    void should_keep_chat_memory_and_add_new_system_message() {
        ChatWithMemory chatWithMemory =
AiServices.builder(ChatWithMemory.class)
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(chatMemory)
                .build();
        String firstSystemMessage = "You are helpful assistant";
        String firstUserMessage = "Hello, my name is Klaus";
        String firstAiMessage =
chatWithMemory.chatWithSystemMessage(firstUserMessage);
        verify(chatLanguageModel).generate(asList(
                systemMessage(firstSystemMessage),
                userMessage(firstUserMessage)
        ));
        String secondSystemMessage = "You are funny assistant";
        String secondUserMessage = "What is my name?";
        String secondAiMessage =
chatWithMemory.chatWithAnotherSystemMessage(secondUserMessage);
        assertThat(secondAiMessage).contains("Klaus");
        verify(chatLanguageModel).generate(asList(
                userMessage(firstUserMessage),
                aiMessage(firstAiMessage),
                systemMessage(secondSystemMessage),
                userMessage(secondUserMessage)
        ));
        verify(chatMemory).add(systemMessage(firstSystemMessage));
        verify(chatMemory).add(userMessage(firstUserMessage));
        verify(chatMemory).add(aiMessage(firstAiMessage));
        verify(chatMemory).add(aiMessage(secondAiMessage));
        verify(chatMemory).add(systemMessage(secondSystemMessage));
        verify(chatMemory).add(userMessage(secondUserMessage));
        verify(chatMemory, times(8)).messages();
    }
    interface ChatWithSeparateMemoryForEachUser {
        String chat(@MemoryId int memoryId, @UserMessage String userMessage);
```

```
    }
    @Test
    void should_keep_separate_chat_memory_for_each_user_in_store() {
        // emulating persistent storage
        Map</* memoryId */ Object, String> persistentStorage = new
HashMap<>();
        ChatMemoryStore store = new ChatMemoryStore() {
            @Override
            public List<ChatMessage> getMessages(Object memoryId) {
                return messagesFromJson(persistentStorage.get(memoryId));
            }
            @Override
            public void updateMessages(Object memoryId, List<ChatMessage>
messages) {
                persistentStorage.put(memoryId, messagesToJson(messages));
            }
            @Override
            public void deleteMessages(Object memoryId) {
                persistentStorage.remove(memoryId);
            }
        };
        ChatMemoryProvider chatMemoryProvider = memoryId ->
MessageWindowChatMemory.builder()
                .id(memoryId)
                .maxMessages(10)
                .chatMemoryStore(store)
                .build();
        int firstMemoryId = 1;
        int secondMemoryId = 2;
        ChatWithSeparateMemoryForEachUser chatWithMemory =
AiServices.builder(ChatWithSeparateMemoryForEachUser.class)
                .chatLanguageModel(chatLanguageModel)
                .chatMemoryProvider(chatMemoryProvider)
                .build();
        String firstMessageFromFirstUser = "Hello, my name is Klaus";
        String firstAiResponseToFirstUser =
chatWithMemory.chat(firstMemoryId, firstMessageFromFirstUser);
        verify(chatLanguageModel).generate(singletonList(userMessage(firstMess
ageFromFirstUser)));
        String firstMessageFromSecondUser = "Hello, my name is Francine";
        String firstAiResponseToSecondUser =
chatWithMemory.chat(secondMemoryId, firstMessageFromSecondUser);
        verify(chatLanguageModel).generate(singletonList(userMessage(firstMess
ageFromSecondUser)));
        String secondMessageFromFirstUser = "What is my name?";
        String secondAiResponseToFirstUser =
chatWithMemory.chat(firstMemoryId, secondMessageFromFirstUser);
        assertThat(secondAiResponseToFirstUser).contains("Klaus");
        verify(chatLanguageModel).generate(asList(
                userMessage(firstMessageFromFirstUser),
                aiMessage(firstAiResponseToFirstUser),
                userMessage(secondMessageFromFirstUser)
        ));
        String secondMessageFromSecondUser = "What is my name?";
        String secondAiResponseToSecondUser =
chatWithMemory.chat(secondMemoryId, secondMessageFromSecondUser);
        assertThat(secondAiResponseToSecondUser).contains("Francine");
        verify(chatLanguageModel).generate(asList(
                userMessage(firstMessageFromSecondUser),
                aiMessage(firstAiResponseToSecondUser),
                userMessage(secondMessageFromSecondUser)
```

```
        ));
        assertThat(persistentStorage).containsOnlyKeys(firstMemoryId,
secondMemoryId);
        List<ChatMessage> persistedMessagesOfFirstUser =
messagesFromJson(persistentStorage.get(firstMemoryId));
        assertThat(persistedMessagesOfFirstUser).containsExactly(
                userMessage(firstMessageFromFirstUser),
                aiMessage(firstAiResponseToFirstUser),
                userMessage(secondMessageFromFirstUser),
                aiMessage(secondAiResponseToFirstUser)
        );
        List<ChatMessage> persistedMessagesOfSecondUser =
messagesFromJson(persistentStorage.get(secondMemoryId));
        assertThat(persistedMessagesOfSecondUser).containsExactly(
                userMessage(firstMessageFromSecondUser),
                aiMessage(firstAiResponseToSecondUser),
                userMessage(secondMessageFromSecondUser),
                aiMessage(secondAiResponseToSecondUser)
        );
    }
    interface Assistant {
        String chat(String userMessage);
    }
    static class Calculator {
        @Tool("calculates the square root of the provided number")
        double squareRoot(@P("number to operate on") double number) {
            return Math.sqrt(number);
        }
    }
    @Test
    void should_execute_tool_then_answer() {
        Calculator calculator = spy(new Calculator());
        ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(10);
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(chatMemory)
                .tools(calculator)
                .build();
        String userMessage = "What is the square root of 485906798473894056
in scientific notation?";
        String answer = assistant.chat(userMessage);
        assertThat(answer).contains("6.97");
        verify(calculator).squareRoot(485906798473894056.0);
        verifyNoMoreInteractions(calculator);
        List<ChatMessage> messages = chatMemory.messages();
        assertThat(messages).hasSize(4);
        assertThat(messages.get(0)).isInstanceOf(dev.langchain4j.data.message.
UserMessage.class);
        assertThat(messages.get(0).text()).isEqualTo(userMessage);
        assertThat(messages.get(1)).isInstanceOf(AiMessage.class);
        AiMessage aiMessage = (AiMessage) messages.get(1);

assertThat(aiMessage.toolExecutionRequest().name()).isEqualTo("squareRoot");
        assertThat(aiMessage.toolExecutionRequest().arguments())
                .isEqualToIgnoringWhitespace("{\"arg0\":
485906798473894056}");
        assertThat(messages.get(1).text()).isNull();

assertThat(messages.get(2)).isInstanceOf(ToolExecutionResultMessage.class);
        assertThat(messages.get(2).text()).isEqualTo("6.97070153193991E8");
        assertThat(messages.get(3)).isInstanceOf(AiMessage.class);
```

```
        assertThat(messages.get(3).text()).contains("6.97");
        verify(chatLanguageModel).generate(
                singletonList(messages.get(0)),
                singletonList(ToolSpecification.builder()
                        .name("squareRoot")
                        .description("calculates the square root of the
provided number")
                        .addParameter("arg0", NUMBER,
JsonSchemaProperty.description("number to operate on"))
                        .build())
        );
        verify(chatLanguageModel).generate(
                asList(messages.get(0), messages.get(1), messages.get(2)),
                singletonList(ToolSpecification.builder()
                        .name("squareRoot")
                        .description("calculates the square root of the
provided number")
                        .addParameter("arg0", NUMBER,
JsonSchemaProperty.description("number to operate on"))
                        .build())
        );
    }
}
```

langchain4j\src\test\java\dev\langchain4j\service\StreamingAiServicesIT.java

```
package dev.langchain4j.service;
import dev.langchain4j.agent.tool.Tool;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.ToolExecutionResultMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiStreamingChatModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.api.Test;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import static dev.langchain4j.model.output.FinishReason.STOP;
import static java.time.Duration.ofSeconds;
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.*;
public class StreamingAiServicesIT {
    StreamingChatLanguageModel streamingChatModel =
OpenAiStreamingChatModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .timeout(ofSeconds(30))
            .logRequests(true)
            .logResponses(true)
            .build();
    interface Assistant {
        TokenStream chat(String userMessage);
    }
    @Test
    void should_stream_answer() throws Exception {
        Assistant assistant = AiServices.create(Assistant.class,
streamingChatModel);
        StringBuilder answerBuilder = new StringBuilder();
        CompletableFuture<String> futureAnswer = new CompletableFuture<>();
        CompletableFuture<Response<AiMessage>> futureResponse = new
CompletableFuture<>();
        assistant.chat("What is the capital of Germany?")
                .onNext(answerBuilder::append)
                .onComplete(response -> {
                    futureAnswer.complete(answerBuilder.toString());
                    futureResponse.complete(response);
                })
                .onError(futureAnswer::completeExceptionally)
                .start();
        String answer = futureAnswer.get(30, SECONDS);
        Response<AiMessage> response = futureResponse.get(30, SECONDS);
        assertThat(answer).contains("Berlin");
        assertThat(response.content().text()).isEqualTo(answer);
        assertThat(response.tokenUsage().inputTokenCount()).isEqualTo(14);
        assertThat(response.tokenUsage().outputTokenCount()).isGreaterThan(1);
        assertThat(response.tokenUsage().totalTokenCount()).isGreaterThan(15);
        assertThat(response.finishReason()).isEqualTo(STOP);
    }
    @Test
    void should_stream_answers_with_memory() throws Exception {
        ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(10);
```

```
        Assistant assistant = AiServices.builder(Assistant.class)
                .streamingChatLanguageModel(streamingChatModel)
                .chatMemory(chatMemory)
                .build();
        String firstUserMessage = "Hi, my name is Klaus";
        CompletableFuture<Response<AiMessage>> firstResultFuture = new
CompletableFuture<>();
        assistant.chat(firstUserMessage)
                .onNext(System.out::println)
                .onComplete(firstResultFuture::complete)
                .onError(firstResultFuture::completeExceptionally)
                .start();
        Response<AiMessage> firstResponse = firstResultFuture.get(30,
SECONDS);
        assertThat(firstResponse.content().text()).contains("Klaus");
        String secondUserMessage = "What is my name?";
        CompletableFuture<Response<AiMessage>> secondResultFuture = new
CompletableFuture<>();
        assistant.chat(secondUserMessage)
                .onNext(System.out::println)
                .onComplete(secondResultFuture::complete)
                .onError(secondResultFuture::completeExceptionally)
                .start();
        Response<AiMessage> secondResponse = secondResultFuture.get(30,
SECONDS);
        assertThat(secondResponse.content().text()).contains("Klaus");
        List<ChatMessage> messages = chatMemory.messages();
        assertThat(messages).hasSize(4);
        assertThat(messages.get(0)).isInstanceOf(UserMessage.class);
        assertThat(messages.get(0).text()).isEqualTo(firstUserMessage);
        assertThat(messages.get(1)).isInstanceOf(AiMessage.class);
        assertThat(messages.get(1)).isEqualTo(firstResponse.content());
        assertThat(messages.get(2)).isInstanceOf(UserMessage.class);
        assertThat(messages.get(2).text()).isEqualTo(secondUserMessage);
        assertThat(messages.get(3)).isInstanceOf(AiMessage.class);
        assertThat(messages.get(3)).isEqualTo(secondResponse.content());
    }
    static class Calculator {
        @Tool
        double squareRoot(double number) {
            return Math.sqrt(number);
        }
    }
    @Test
    void should_execute_tool_then_stream_answer() throws Exception {
        Calculator calculator = spy(new Calculator());
        ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(10);
        Assistant assistant = AiServices.builder(Assistant.class)
                .streamingChatLanguageModel(streamingChatModel)
                .chatMemory(chatMemory)
                .tools(calculator)
                .build();
        StringBuilder answerBuilder = new StringBuilder();
        CompletableFuture<String> futureAnswer = new CompletableFuture<>();
        CompletableFuture<Response<AiMessage>> futureResponse = new
CompletableFuture<>();
        String userMessage = "What is the square root of 485906798473894056
in scientific notation?";
        assistant.chat(userMessage)
                .onNext(answerBuilder::append)
                .onComplete(response -> {
```

```
                    futureAnswer.complete(answerBuilder.toString());
                    futureResponse.complete(response);
                })
                .onError(futureAnswer::completeExceptionally)
                .start();
        String answer = futureAnswer.get(30, SECONDS);
        Response<AiMessage> response = futureResponse.get(30, SECONDS);
        assertThat(answer).contains("6.97");
        assertThat(response.content().text()).isEqualTo(answer);
        // tool arguments provided by the model (which contribute to the
input token count)
        // can have different formatting (extra spaces) which results in one
extra/missing token
        assertThat(response.tokenUsage().inputTokenCount()).isBetween(146,
147);
        assertThat(response.tokenUsage().outputTokenCount()).isGreaterThan(1);

assertThat(response.tokenUsage().totalTokenCount()).isGreaterThan(148);
        assertThat(response.finishReason()).isEqualTo(STOP);
        verify(calculator).squareRoot(485906798473894056.0);
        verifyNoMoreInteractions(calculator);
        List<ChatMessage> messages = chatMemory.messages();
        assertThat(messages).hasSize(4);
        assertThat(messages.get(0)).isInstanceOf(UserMessage.class);
        assertThat(messages.get(0).text()).isEqualTo(userMessage);
        assertThat(messages.get(1)).isInstanceOf(AiMessage.class);
        AiMessage aiMessage = (AiMessage) messages.get(1);

assertThat(aiMessage.toolExecutionRequest().name()).isEqualTo("squareRoot");
        assertThat(aiMessage.toolExecutionRequest().arguments())
                .isEqualToIgnoringWhitespace("{\"arg0\":
485906798473894056}");
        assertThat(messages.get(1).text()).isNull();

assertThat(messages.get(2)).isInstanceOf(ToolExecutionResultMessage.class);
        assertThat(messages.get(2).text()).isEqualTo("6.97070153193991E8");
        assertThat(messages.get(3)).isInstanceOf(AiMessage.class);
        assertThat(messages.get(3).text()).contains("6.97");
    }
}
```

langchain4j\src\test\java\dev\langchain4j\store\embedding\EmbeddingStoreIngestorTest.java

```
package dev.langchain4j.store.embedding;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.document.DocumentTransformer;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.data.segment.TextSegmentTransformer;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.data.segment.TextSegment.textSegment;
import static java.util.Arrays.asList;
import static org.mockito.Mockito.*;
class EmbeddingStoreIngestorTest {
    @Test
    public void should_extract_text_then_split_into_segments_then_embed_them_a
nd_store_in_embedding_store() {
        Document firstDocument = Document.from("First sentence.");
        Document secondDocument = Document.from("Second sentence. Third
sentence.");
        List<Document> documents = asList(firstDocument, secondDocument);
        DocumentTransformer documentTransformer =
mock(DocumentTransformer.class);

when(documentTransformer.transformAll(documents)).thenReturn(documents);
        DocumentSplitter documentSplitter = mock(DocumentSplitter.class);
        List<TextSegment> segments = asList(
                textSegment("First sentence."),
                textSegment("Second sentence."),
                textSegment("Third sentence.")
        );
        when(documentSplitter.splitAll(documents)).thenReturn(segments);
        TextSegmentTransformer textSegmentTransformer =
mock(TextSegmentTransformer.class);
        List<TextSegment> transformedSegments = asList(
                textSegment("Transformed first sentence."),
                textSegment("Transformed second sentence."),
                textSegment("Transformed third sentence.")
        );
        when(textSegmentTransformer.transformAll(segments)).thenReturn(transfo
rmedSegments);
        EmbeddingModel embeddingModel = mock(EmbeddingModel.class);
        List<Embedding> embeddings = asList(
                Embedding.from(new float[]{1}),
                Embedding.from(new float[]{2}),
                Embedding.from(new float[]{3})
        );
        when(embeddingModel.embedAll(transformedSegments)).thenReturn(Response
.from(embeddings));
        EmbeddingStore<TextSegment> embeddingStore =
mock(EmbeddingStore.class);
        EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
                .documentTransformer(documentTransformer)
                .documentSplitter(documentSplitter)
                .textSegmentTransformer(textSegmentTransformer)
                .embeddingModel(embeddingModel)
```

```
                .embeddingStore(embeddingStore)
                .build();
        ingestor.ingest(documents);
        verify(documentTransformer).transformAll(documents);
        verifyNoMoreInteractions(documentTransformer);
        verify(documentSplitter).splitAll(documents);
        verifyNoMoreInteractions(documentSplitter);
        verify(textSegmentTransformer).transformAll(segments);
        verifyNoMoreInteractions(textSegmentTransformer);
        verify(embeddingModel).embedAll(transformedSegments);
        verifyNoMoreInteractions(embeddingModel);
        verify(embeddingStore).addAll(embeddings, transformedSegments);
        verifyNoMoreInteractions(embeddingStore);
    }
}
```

langchain4j\src\test\java\dev\langchain4j\store\embedding\inmemory\InMemoryEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.inmemory;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.io.TempDir;
import java.nio.file.Path;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
class InMemoryEmbeddingStoreTest {
    @TempDir
    Path temporaryDirectory;
    private final EmbeddingStore<TextSegment> embeddingStore = new
InMemoryEmbeddingStore<>();
    private final EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    @Test
    void should_add_embedding() {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
```

```
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_embedding_with_segment_with_metadata() {
        TextSegment segment = TextSegment.from(randomUUID(),
Metadata.from("test-key", "test-value"));
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
```

```
        );
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
```

```
    }
    @Test
    void should_return_correct_score() {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
referenceEmbedding)),
                withPercentage(1)
        );
    }
    @Test
    void should_serialize_to_and_deserialize_from_json() {
        InMemoryEmbeddingStore<TextSegment> originalEmbeddingStore =
createEmbeddingStore();
        String json = originalEmbeddingStore.serializeToJson();
        InMemoryEmbeddingStore<TextSegment> deserializedEmbeddingStore =
InMemoryEmbeddingStore.fromJson(json);

assertThat(deserializedEmbeddingStore).isEqualTo(originalEmbeddingStore);
    }
    @Test
    void should_serialize_to_and_deserialize_from_file() {
        InMemoryEmbeddingStore<TextSegment> originalEmbeddingStore =
createEmbeddingStore();
        Path filePath = temporaryDirectory.resolve("embedding-store.json");
        originalEmbeddingStore.serializeToFile(filePath);
        InMemoryEmbeddingStore<TextSegment> deserializedEmbeddingStore =
InMemoryEmbeddingStore.fromFile(filePath);

assertThat(deserializedEmbeddingStore).isEqualTo(originalEmbeddingStore);
    }
    private InMemoryEmbeddingStore<TextSegment> createEmbeddingStore() {
        InMemoryEmbeddingStore<TextSegment> embeddingStore = new
InMemoryEmbeddingStore<>();
        TextSegment segment = TextSegment.from("first");
        Embedding embedding = embeddingModel.embed(segment).content();
        embeddingStore.add(embedding, segment);
        TextSegment segmentWithMetadata = TextSegment.from("second",
Metadata.from("key", "value"));
        Embedding embedding2 =
embeddingModel.embed(segmentWithMetadata).content();
        embeddingStore.add(embedding2, segmentWithMetadata);
        return embeddingStore;
    }
}
```

```
 langchain4j\src\test\resources\mockito-
extensions\org.mockito.plugins.MockMaker

mock-maker-inline
```

langchain4j\src\test\resources\test-file-iso-8859-1.txt

test
content

langchain4j\src\test\resources\test-file-utf8.txt

```
test
content
```

langchain4j\src\test\resources\test-file.banana

this document type is unknown

langchain4j\src\test\resources\test-file.doc

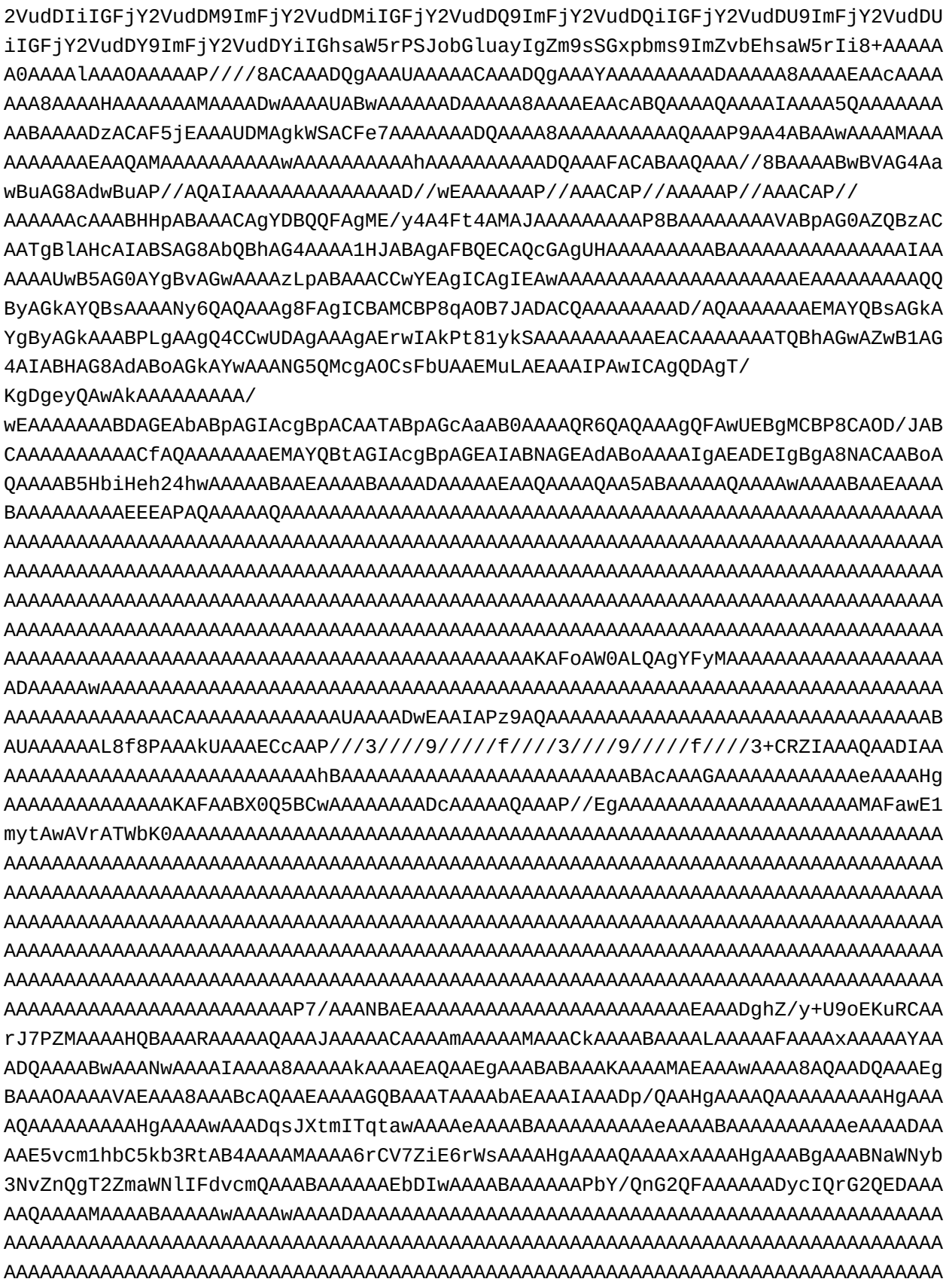
BASE64:

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/
CQAGAAAAAAAAAAAAAAAABAAAAJwAAAAAAAAAAAEAAAKQAAAAEAAAD+////AAAAACYAAAD//////////
//
//
//
//
//
//
//
///////////////////spcEAiwQJBAAA8BK/AAAAAAABEAAAAAAACAAADQgAAA4AYmpiakIUQhQAAA
AAAAAAAAAAAAAAAAAAAAJBBYALg4AACB+rF8gfqxfDQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAD//w8AAAAAAAAAAAAD//w8AAAAAAAAAAAAD//w8AAAAAAAAAAAAAAAAAAAAAAALcAAAAAAEQHAA
AAAAAARAcAANYUAAAAAAAAA1hQAAAAAAADWFAAAAAAAANYUAAAAAAAA1hQAABQAAAAAAAAAAAAAAP//
//8AAAAA6hQAAAAAAAADqFAAAAAAAAOoUAAAAAAAAA6hQAAAwAAAD2FAAADAAAAOoUAAAAAAAAA/RUAAM
QBAAACFQAAAAAAAAIVAAAAAAAAAhUAAAAAAAACFQAAAAAAAAIVAAAAAAAAAhUAAAAAAAACFQAAAAAA
AAIVAAAAAAAAfBUAAAIAAAB+FQAAAAAAAH4VAAAAAAAAfhUAAAAAAAB+FQAAAAAAAH4VAAAAAAAAfh
UAACQAAADBFwAAtgIAAHcaAAA2AAAAohUAABUAAAAAAAAAAAAAAAAAAAAAAAAAAAA1hQAAAAAAAACFQAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAACFQAAAAAAAAIVAAAAAAAAAhUAAAAAAAACFQAAAAAAAKIVAAAAAA
AAAAAAAAAAAADWFAAAAAAAANYUAAAAAAAAAhUAAAAAAAAAAAAAAAAAAAAAAIVAAAAAAAAAtxUAABYAAAA8
FQAAAAAAADwVAAAAAAAAPBUAAAAAAAACFQAAEAAAANYUAAAAAAAAAhUAAAAAAAADWFAAAAAAAAAIVAA
AAAAAAfBUAAAAAAAAAAAAAAAAAAADwVAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAhUAAAAAAAB8FQAAAAAAAAAAAAAAAAAAAAAPBUAAAAAAAAAAAAAAAAAAAD
wVAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAPBUAAAAAAAAAAAAAAAAAAP////8AAAAAQFVzKArG2QEAAAAAAA
AAAP////8AAAAAEhUAAAoAAAA8FQAAAAAAAAAAAAAAAAAAAAAAaBUAABQAAADNFQAAMAAAAAP0VAAAAAAAA
PBUAAAAAAACtGgAAAAAAABwVAAAQAAAArRoAAAAAAAAA8FQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAK0aAAAA
AAAAAAAAAAAAAADWFAAAAAAAADwVAAAsAAAAAhUAAAAAAAACFQAAAAAAAADwVAAAAAAAAAhUAAAAAAAA
ACFQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAhUAAAAAAAACFQAAAAAAAAIV
AAAAAAAAohUAAAAAAACiFQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALBUAAB
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIVAAAAAAAAAhUAAAAAAAACFQAAAAAA
AP0VAAAAAAAAAhUAAAAAAAACFQAAAAAAAAIVAAAAAAAAAhUAAAAAAAAAAAAAAAAAAP////8AAAAA//
///wAAAAD/////AAAAAAAAAAAAAAAAA/////wAAAAD/////AAAAAP////8AAAAA/////wAAAAD/////
AAAAAP////8AAAAA/////wAAAAD/////AAAAAP////8AAAAA/////wAAAAD/////
AAAAAP////8AAAAA/////wAAAAD/////AAAAAK0aAAAAAAAAAhUAAAAAAAACFQAAAAAAAAIVAAAAAA
AAAhUAAAAAAAACFQAAAAAAAAIVAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACFQAAAAAAAAIVAAAAAAAAAhUAAAAAAABEBwAAWAwAAJwTAA
A6AQAABQASAQAAAAwSBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAHRlc3QgY29udGVudA0AAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAgAAAEIAAAFCAAADAgAAA0IAAD4+PjtAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAFBVogkWSABZoAFAzAG1ICQRzSAkEAA4WaIJFkgBtSAkEc0gJBAQACAAA
DQgAAP0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAASwAMZBoAR+w0C8gsOA9IbCgBS
KwoAUjkKAFJJCgBSWwAAAXsMQCGLDEAgyQxAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAHgYPABIAAQB4AQ8ABwADAAQAAwAAAAQACAAAAJgAAACe
AAAAngAAAJ4AAACeAAAAngAAAJ4AAACeAAAAngAAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAA
A2BgAANgYAAHYCAAB2AgAAdgIAAHYCAAB2AgAAdgIAAHYCAAB2AgAAdgIAADYGAAA2BgAANgYAADYG
AAA2BgAANgYAAD4CAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAAD
YGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYA

```
ADYGAACoAAAANgYAADYGAAAWAAAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAC4AAAANg
YAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAAaAEAAEgBAAA2BgAA
NgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2Bg
AANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2
BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAA
A2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYGAAA2BgAANgYAADYG
AAA2BgAANgYAADYGAAA2BgAANgYAAHACAAA2BgAAMgYAABgAAADGAwAA1gMAAOYDAAD2AwAABgQAAB
YEAAAmBAAANgQAAEYEAABWBAAAZgQAAHYEAACGBAAAlgQAAMYDAADWAwAA5gMAAPYDAAAGBAAAFgQA
ADIGAAAoAgAA2AEAAOgBAAAmBAAANgQAAEYEAABWBAAAZgQAAHYEAACGBAAAlgQAAMYDAADWAwAA5g
MAAPYDAAAGBAAAFgQAACYEAAA2BAAARgQAAFYEAABmBAAAdgQAAIYEAACWBAAAxgMAANYDAADmAwAA
9gMAAAYEAAAWBAAAJgQAADYEAABGBAAAVgQAAGYEAAB2BAAAhgQAAJYEAADGAwAA1gMAAOYDAAD2Aw
AABgQAABYEAAAmBAAANgQAAEYEAABWBAAAZgQAAHYEAACGBAAAlgQAAMYDAADWAwAA5gMAAPYDAAAG
BAAAFgQAACYEAAA2BAAARgQAAFYEAABmBAAAdgQAAIYEAACWBAAAxgMAANYDAADmAwAA9gMAAAYEAA
AWBAAAJgQAADYEAABGBAAAVgQAAGYEAAB2BAAAhgQAAJYEAAA4AQAAWAEAAPgBAAAIAgAAGAIAAFYC
AAB+AgAAkAIAAKACAACwAgAAwAIAANACAACAAgAA4AIAAPACAAAAAwAAEAMAACADAAAwAwAAQAMAAO
ACAADwAgAAAAMAABADAAAgAwAAMAMAAEADAADgAgAA8AIAAAADAAAQAwAAIAMAADADAABAAwAA4AIA
APACAAAAAwAAEAMAACADAAAwAwAAQAMAAOACAADwAgAAAAMAABADAAAgAwAAMAMAAEADAADgAgAA8A
IAAAADAAAQAwAAIAMAADADAABAAwAA4AIAAPACAAAAAwAAEAMAACADAAAwAwAAQAMAAOACAADwAgAA
AAMAABADAAAgAwAAMAMAAEADAADgAgAA8AIAAAADAAAQAwAAIAMAADADAABAAwAA4AIAAPACAAAAAw
AAEAMAACADAAAwAwAAQAMAAOACAADwAgAAAAMAABADAAAgAwAAMAMAAEADAADgAgAA8AIAAAADAAAQ
AwAAIAMAADADAABAAwAA4AIAAPACAAAAAwAAEAMAACADAAAwAwAAQAMAAOACAADwAgAAAAMAABADAA
AgAwAAMAMAAEADAAA2BgAANgYAADYGAAA2BgAANgYAACAAAABPSgMAUEoEAFFKAwBfSAEEbUgADG5I
EgRzSAAMdEgSBAAAAABIAABg8f8CAEgADBAAAAAAAAAAAAYATgBvAHIAbQBhAGwAAAACAAAAIABDSh
gAS0gCAF9IAQRhShgAbUgADG5IEgRzSAAMdEgSBAAAAAAAAAAAAAAAAAAAAAAAAAAAAEQAQSDy/6EARAAM
DQAAAAAAABAAFgBEAGUAZgBhAHUAbAB0ACAAUABhAHIAYQBnAHIAYQBwAGgAIABGAG8AbgB0AAAAAA
BSAGkA8/+zAFIADA0AAAAAAAAwBgwAVABhAGIAbABlACAATgBvAHIAbQBhAGwAAAAcABf2AwAANNYG
AAEKA2wANNYGAAEFAwAAYfYDAAACAAsAAAAoAGsg9P/
BACgAAA0AAAAAAAAwBgcATgBvACAATABpAHMAdAAAAAIADAAAAAAAAUEsDBBQABgAIAAAAIQDp3g+//
wAAABwCAAATAAAAW0NvbnRlbnRfVHlwZXNdLnhtbKyRy07DMBBF90j8g+UtSpyygAgl6YLHjseifMD
ImSQWydiyp1X790zSVEKoIBZsLNkz954743K9Hwe1w5icp0qv8kIrJOsbR12l3zdP2a1WiYEaGDxhp
Q+Y9Lq+vCg3h4BJiZpSpXvmcGdMsj2OkHIfkKTS+jgCyzV2JoD9gA7NdVHcGOuJkTjjyUPX5QO2sB1
YPe7l+Zgk4pC0uj82TqxKQwiDs8CS1Oyo+UbJFkIuyrkn9S6kK4mhzVnCVPkZsOheZTXRNajeIPILj
BLDsAyJX89nIBkt5r87nons29ZZbLzdjrKOfDZezE7B/xRg9T/oE9PMf1t/
AgAA//8DAFBLAwQUAAYACAAAACEApdan58AAAAA2AQAACwAAAF9yZWxzLy5yZWxzhI/PasMwDIfvhb
2D0X1R0sMYJXYvpZBDL6N9AOEof2giG9sb69tPxwYKuwiEpO/3qT3+rov54ZTnIBaaqgbD4kM/
y2jhdj2/f4LJhaSnJQhbeHCGo3vbtV+8UNGjPM0xG6VItjCVEg+I2U+8Uq5CZNHJENJKRds0YiR/
p5FxX9cfmJ4Z4DZM0/UWUtc3YK6PqMn/s8MwzJ5PwX+vLOVFBG43lExp5GKhqC/
jU72QqGWq1B7Qtbj51v0BAAD//wMAUEsDBBQABgAIAAAAIQBreZYWgwAAAIoAAAAcAAAAdGhlbWUvd
GhlbWUvdGhlbWVNYW5hZ2VyLnhtbAzMTQrDIBBA4X2hd5DZN2O7KEVissuuu/YAQ5waQceg7l5dZN2O7
jgzfO3xTVm0sNWSycBw2KZc0uiLfwfCynG6jaSBzFLGzhxxXm6XgYybSNE99JyHNRfSPVkIWttd0g1
rUr1SHvLN1euSRqPYtHV+jT9yniResrJgoCOP0BAAD//wMAUEsDBBQABgAIAAAAIQDCOo714AYAALM
fAAAWAAAAdGhlbWUvdGhlbWUvdGhlbWUxLnhtbOxZS28bNxC+F+h/2F+F+h/wOy0gRuA4aRMkaYI4
0pL7dZ28bNxC+F+h/2F+F+h/wOy0gRuA4aRMkaYI4ZRIiR
0pL7dLiLhckZUe3Ijn1UqBAWvTQAL31UBQN0AANeumPMeCgTX9Eh1xpl5So+IEUCApbgKGlvhl+nJm
dmZ29c/dpTL0TzAVhScMv3yr5Hk6GLCBJ2PAf9buf3fY9IVESIMoS3PCnWPh3tz/95A7akhGOsQfyi
```

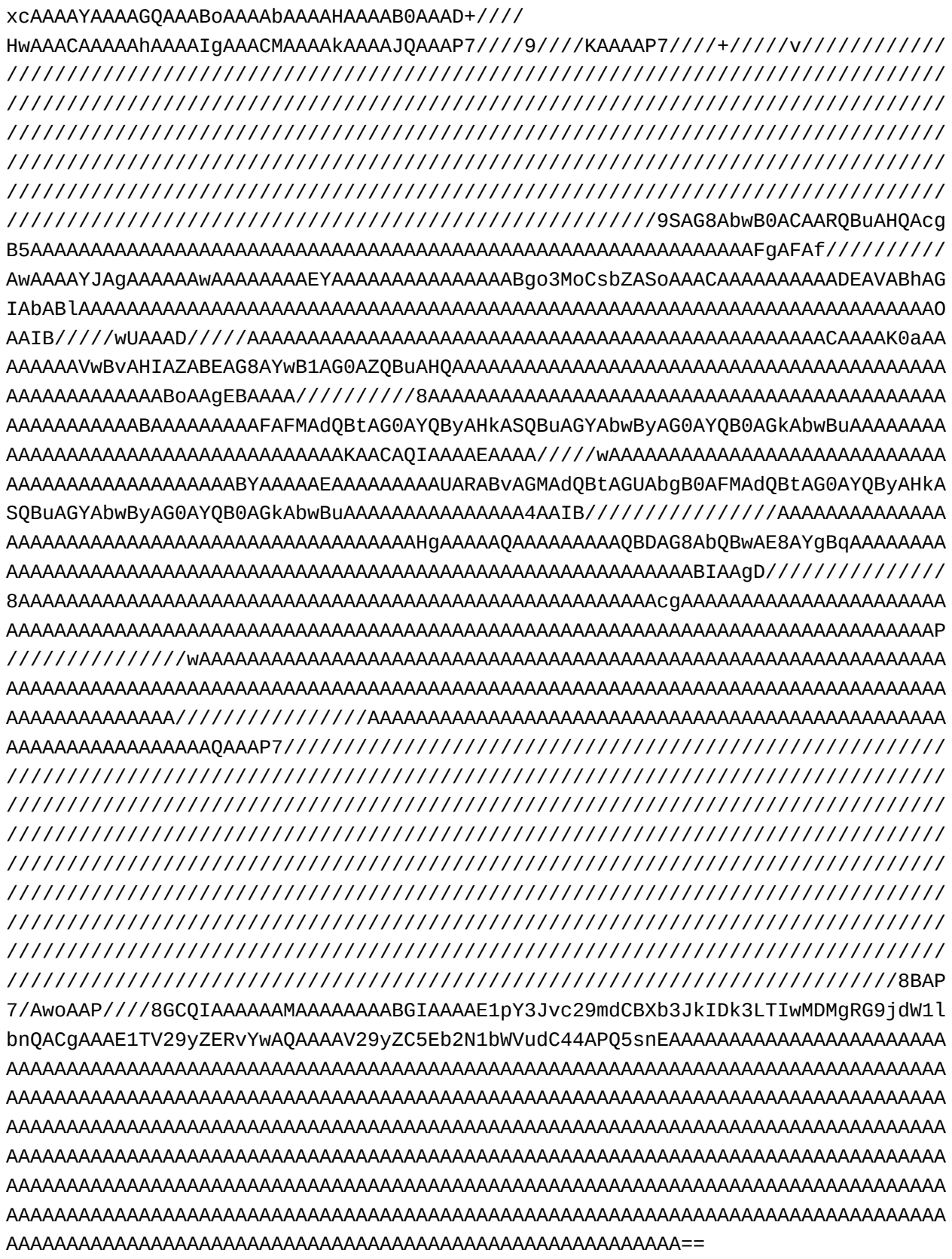
dhCDT+SMt1aWxNDWEbiFktxAr+NGI+RhEsergUcnYLemK6tl0q1tRiRxPcSFIPaw9GIDLHXVyr97bn
yDoXLRAq1MKS8p1RjS0Jjg3FZIcRUtCj3ThBt+LBPwE77+Kn0PYqEhB8afkn/+Wvbd9bQ1kyIyhWyh
lxX/83kZgLBeF3vycNBvmmlUq3UdnL9GkDlMq5T79Q6tVyfBqDhEE6acbF11tdblRnWAGVfHbrb9fZ
G2cIb+jeWOO9U1cfCa1Cmv7KE73ZbYEULr0EZvrqErzY3m21bvwZl+NoSvl7aaVfqln4NiihJxkvoU
rW20ZqfNoeMGN11wjerlW59faa8QEE05NGlthixRK6KtRgdM94FgAJSJEniyWmKR2gIUdxClAw48fZ
IGEHgpShhApZL66VuaQP+q09Ff9MeRVsYGdKKFzARS0uKjyeGnKSy4d8Hrb4BOX/
z5uzZ67Nnv589f3727NfZ3lqVJbeLktCUe/fTN/+8/NL7+7cf3734Ntt6ES9M/Ntfvnr7x5/
vUw8nLkxx/t2rt69fnX//9V8/v3Bo3+FoYML7JMbCO8Cn3kMWwwEd/
PGAX02iHyFiSuwkoUAJUrs49HdkZKEPpogiB66JbTs+5pBqXMB7k2OLcC/iE0kcGh9EsQXcZ4w2GXd
a4YHayzBzf5KE7s35xMQ9ROjEtXcLJZaXO5MUcixxqWxF2KJ5RFEiUYgTLD31Gxtj7DjdE0Isu+6TI
WeCjaT3hHhNRJwm6ZOBFU2F0C6JwS9TF0Hwt2Wb/
cdek1HXqdv4xEbCvYGog3wfU8uM99BEotilso9iahp8D8nIRbI35UMT1xESPB1iyrxOgIVwyRxyOK/
h9AeQZtxu36fT2EZyScYunXuIMRPZZuNWhOLUhe2RJDKxn4sxhCjyjph0wfeZfYeoa/ADSla6+zHBl
rsvzgaPIMOalIoAUb9MuMOX9zCz4rc3pSOEXalmh8dWit3hxBkdzUlohfYexhSdogBj79HnDgZNllo
2L0jfjyCr7GJXYN1Hdqyq6wQL6JVUc7OcJ/eIsEK2h0O2gs/+dCHxTFESI75K8wF43bR5B0pd7AqAQ
zocm8ADAj0gxIvTKIcCdBjBvVLrUYSsAqauhTtep9zy32XuMbgvjy0al7gvQQZfWQYSuynzXtv0EbU
2KAKmj6DLcKVbELHcX4io4qrFJk65kX3TFm6A7shqemKSXNgBLfQ+1f+u94EO4/
yHl4774MP0O27FVrK6YqezKpnsLvQ3q3CLXU2L8YB8/E1NG02SIwx1ZDlj3fQ0Nz2N/7/vaVbdzzed
zKp+46aT8aHDuOlkZsOVD9PJFM0L9DVq4JENevTYJ1459RkRSntySvGe0IMfAc8zQRcWlZyeeOJ8Cp
hG8FWVOdjAwoUcaRmPM/
kFkVEvQilMh8q+UhKKmepQeCkTMDTSy07dCk8n8T4LsmFnuawGm1llFUgW66Vqvg6DKpmha/
VigJer12xDPWidE1CyVyFhbGaT2HCQqM8XlZH0WBeM5iChT/ZBWGw6WNxW6ueuWmIB1HKvwAO3B4/
pDb9aAREQgnkcNOeB8lPm6rl3tTM/pKdXGdOKAGiw5xFQeHpTcV15PHW6LNQu4WmLhBFuNgltGd3gi
Qgeg2fRqVYvQ+Oqvt4sXGrRU6bQ+0FoFTTqt9/H4rq+BrnF3EATM1PQxDtt+LWNKoTMEKUNfwRDY/
gapxA7Qj1zIRrCm5eh5NkNf53MknIh20hEmcF10smyQUwk5h4lccNXx8/
dQBOdQzS38jokhI+W3CaklY+NHDjddjIejfBQmm43VpSls0vI8FmucP6qxa8PVpJsAu7uRcGpN6AT/
hBBiFXrZWXAgAh4d1DOrBkQeBmWJ7Ii/hYK0yztmm+jdAxl64imEZpVFDOZZ3CdynM6+iq3gXE1OzM
Y1DDJrBAOQlVgTaNa1TSvGhmHlVX3YiFlOSNpFjXTyiqqarqzmLXDvAws2PJ6Rd5gNTcx5DSzwmepe
zHlbs5z3UKfkFcJMHhuP0fVvURBMKgVm1nUFOPlNKxy9mzVrh3zA15A7TJFwsj6tbnaBbvlNcK5HSx
eq/KD3GLUwtJo3ldqS+u35uaLbTY4huTRhi53QqXQroTJLkfQEPV0T5KnDS26/S8AAAD//wMAUEsDB
BQABgAIAAAAIQAN0ZCftgAAABsBAAAnAAAAdGhlbWUvdGhlbWUvX3JlbHMvdGhlbWVNYW5hZ2VyLnh
tbC5yZWxzhI9NCsIwFIT3gncIb2/TuhCRJt2I0K3UA4TkNQ02PyRR7O0NriwILodhvplpu5edyRNjM
t4xaKoaCDrplXGawW247I5AUhZOidk7ZLBggo5vN+0VZ5FLKE0mJFIoLjGYcg4nSpOc0IpU+YCuOKO
PVuQio6ZByLvQSPd1faDxmwF8xSS9YhB71QAZllCa/7P9OBqJZy8fFl3+UUFz2YUFKKLGzOAjm6pMB
MpburrE3wAAAP//AwBQSwECLQAUAAYACAAAACEA6d4Pv/8AAAAcAgAAEwAAAAAAAAAAAAAAAAAAAAAA
AW0NvbnRlbnRfVHlwZXNdLnhtbFBLAQItABQABgAIAAAAIQCl1qfnwAAAADYBAAALAAAAAAAAAAAAAA
AAAADABAABfcmVscy8ucmVsc1BLAQItABQABgAIAAAAIQBreZYWgwAAAIoAAAAcAAAAAAAAAAAAAAAA
AABkCAAB0aGVtZS90aGVtZS90aGVtZU1hbmFnZXIueG1sUEsBAi0AFAAGAAgAAAAhAMI6jvXgBgAAs
x8AABYAAAAAAAAAAAAAAAAA1gIAAHRoZW1lL3RoZW1lL3RoZW1lMS54bWxQSwECLQAUAAYACAAAACE
ADdGQn7YAAAAbAQAAJwAAAAAAAAAAAAAAAADqCQAAdGhlbWUvdGhlbWUvX3JlbHMvdGhlbWVNYW5hZ
2VyLnhtbC5yZWxzUEsFBgAAAAAFAAUAXQEAAOUKAAAAAD/eG1sIHZlcnNpb249IjEuMCIgZW5jb2R
pbmc9IlVURi04IiBzdGFuZGFsb25lPSJ5ZXMiPz4NCjxhOmNsck1hcCB4bWxuczphPSJodHRwOi8vc
2NoZW1hcy5vcGVueG1sZm9ybWF0cy5vcmcvZHJhd2luZ21sLzIwMDYvbWFpbiIgYmcxPSJsdDEiIHR
4MT0iZGsxIiBiZzI9Imx0MiIgdHgyPSJkazIiIGFjY2VudDE9ImFjY2VudDEiIGFjY2VudDI9ImFjY



AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAD+/wAADQQBAAAAAAAAAAAAAAAAAAAAAAAABAAAAAtXN1ZwuGxCTlwg
AKyz5rjAAAADoAAAADAAAAAEAAABoAAAADwAAAHAAAAAFAAAAfAAAAAYAAACEAAAAEQAAAIwAAAAXA
AAAlAAAAAsAAACcAAAAEAAAAKQAAAATAAAArAAAABYAAAC0AAAADQAAALwAAAAMAAAAyQAAAAIAAAA
QJwAAHgAAAAQAAAAAAAAAAAwAAAAEAAAADAAAAAQAAAAMAAAAMAAAAAwAAAAAAAEAALAAAAAAAAAAAsAA
AAAAAAACwAAAAAAAAAAALAAAAAAAAAAB4QAAABAAAAAAQAAAAAMEAAAAgAAAB4AAAAGAAAAVGl0bGUAAwA
AAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAIAAAADAAAABAAAAAUAAAAGAAAABwAAAP7///8JAAAACgA
AAAsAAAAMAAAADQAAAA4AAAAPAAAAEAAAABEAAAASAAAAEwAAABQAAAAVAAAA/v///



langchain4j\src\test\resources\test-file.docx

BASE64:

UEsDBBQABgAIAAAAIQDfpNJsWgEAACAFAAATAAgCW0NvbnRlbnRfVHlwZXNdLnhtbCCiBAIooAACAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC0lMtuwjAQRfeV+g+Rt1Vi
6KKqKgKLPpYtUukHGHsCVv2Sx7z+vhMCUVUBkQpsIiUz994zVsaD0dqabAkRtXcl6oGTXmk3K9
nX5C1/ZBkm4ZQw3kHJNoBsNLy9GUw2ATAjtcOSzVMKT5yjnIMVWPgAjiqVj1Ykeo0zHoT8FjPg973e
A5feJXApT7UHGw5eoBILk7LXNX1uSCIYZNlz01hnlUyEYLQUiep86dSflHyXUJBy24NzHfCOGhg/
mFBXjgfsdB90NFEryMYipndhqYuvfFRcebmwpCxO2xzg9FWlJbT62i1ELwGRztyaoq1Yod2e/
ygHpo0BvDxF49sdDymR4BoAO+dOhBVMP69G8cu8E6Si3ImYGrg8RmvdCZFoA6F59s/
m2NqciqTOcfQBaaPjP8ber2ytzmngADHp039dm0jWZ88H9W2gQB3I5tv7bfgDAAD//wMAUEsDBBQAB
gAIAAAAIQAekRq37wAAAE4CAAALAAgCX3JlbHMvLnJlbHMgogQCKKAAAgAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAArJLBasMwDEDvg/2D0b1R2sEYo04vY9DbGNkHCFtJT
BPb2GrX/
v082NgCXelhR8vS05PQenOcRnXglF3wGpZVDYq9Cdb5XsNb+7x4AJWFvKUxeNZw4gyb5vZm/
cojSSnKg4tZFYrPGgaR+IiYzcAT5SpE9uWnC2kiKc/
UYySzo55xVdf3mH4zoJkx1dZqSFt7B6o9Rb6GHbrOGX4KZj+xlzMtkI/
C3rJdxFTqk7gyjWop9SwabDAvJZyRYqwKGvC80ep6o7+nxYmFLAmhCYkv+3xmXBJa/ueK5hk/
Nu8hWbRf4W8bnF1B8wEAAP//AwBQSwMEFAAGAAgAAAAhANZks1H0AAAAMQMAABwACAF3b3JkL19yZW
xzL2RvY3VtZW50LnhtbC5yZWxzIKIEASigAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAArJLLasMwEEX3
hf6DmH0tO31QQuRsSiHb1v0ARR4/qCwJzfThv69Thv69ISevQYLrwcq6Yc8+ANtvPwYp3jNR7p6DIchDojK
971yp4qR6v7kEQa1dr6x0qGJFgW15ebJ7Qak5L1PWBRKI4UtAxh7WUZDocNGU+oEsvjY+D5jTGVgZt
XnWLcpXndzJOGVCeMMWuvhB39TWIagz4H7Zvmt7ggzdvAzo+UyE/

cP+MzOk4SlgdW2QFkzBLRJDnRVZLitAfi2Myp1AsqsCjxanAYZ6rv12yntMu/
rYfxu+wmHO4WdKh8Y4rvbcTj5/oKCFPPnr5BQAA//8DAFBLAwQUAAYACAAAACEAxuw8OQMDAAClCwA
AEQAAAHdvcmQvZG9jdW1lbnQueG1spJZbb9sgFIDfJ+0/
WH5P8SW3Wk2qaNmqPkyK1vUHEExsVAMWkDjZr9/B12zeKsd9MXAuH4cDHPPweOaZc6JKMylWrn/
nuQ4VRMZMJCv39ee3ydJ1tMEixpkUdOVeqHYf158/PRRRLMmRU2EcQAgdFTlZuakxeYSQJinlWN9xR
pTU8mDuiORIHg6MUFRIFaPA872ylytJqNYw3xcsTli7NY6ch9FihQtwtsApIilWhp47hn8zZIbu0bI
PCkaAYIWB30eFN6PmyEbVA01HgSCqHmk2jvSPxc3HkYI+aTGOFPZJy3Gk3nHi/QMucypAeZCKYwNDl
SCO1dsxnwA4x4btWcbMBZjevMFgJt5GRAReLYGH8c2EBeIyplkYNxS5co9KRLX/
pPW3oUeVf920HjQbNi1Md4/
o2WTaNL5qSO4q921dWMqsIUUzyKMUOmV5Wx34WBoo0wZyei8BJ541dkXuD7xq/ytt22obOuCQ8Ou94
1kV+ftE3xuwmxbRegwJ4c85m0g4nOBu4lGpuUquP7D4NICgB5gTOvBn0TCWNQOR7nZbDht4rRpOtSu
Ww7rE+gNr4N/
BXAF0bOL0JkrQ5BVZX2xwinV70C2R3hbUrMVd+FWO8uRjF+FJyWPe0djHaM9dSSzs4+QGVn2hri+5/
lgwLynOoVJyEj0nQiq8zyAiuB4OnHCn3AH7hYNim7JLz6Xc7rVja4y7hlfVXsYX2+agm0Y5VvgZDqU
ffl1stxv4h1gp/JNMKd1swnCzgSdaEcELLv6xcj0vDGee57WinbLC+2A6WwatcEsP+JiZK0055U7ZR
lVNhkUC9icMNZ+KyeuLi9YPqFajznqoSxGZtaHaOEQKA/XYKkylrnjWRFNidmrocsqgk5dfoIRy5gf
BtDSEm+PPltBHlcF3bIlGQtX1p5WJYklquuFeGiN5N87o4UqbUhxTmHfhlZk+SGmuhsnRlMN6OiIzD
VKdY0Irm1IMj+UnZfc+ypigO2YIRBnOSyfUrLvsVgcAde/r9W8AAAD//wMAUEsDBBQABgAIAAAAIQC
29GeY0gYAAMkgAAAVAAAAd29yZC90aGVtZS90aGVtZTEueG1s7FlLixtHEL4H8h+Guct6zehhrDXSS
PJr1zbetYOPvVJrpq2eadHd2rUwhmCfcgkEnJBDDLnlEEIMMcTkkh9jsEmcH5HqHkkzLfXEj12DCbu
CVT++qv66qrq6NHPh4v2YOkeYC8KSjls9V3EdnIzYmCRhx719MCy1XEdIlIwRZQnuuAss3Is7n392A
Z2XEY6xA/KJOI86biTl7Hy5LEYwjMQ5NsMJzE0Yj5GELg/LY46OQW9My7VKpVGOEUlcJ0ExqL0xmZA
Rdg6USndnpXxA4V8ihRoYUb6vVGNDQmPH06r6EgsRUO4cIdpxYZ0xOz7A96XrUCQkTHTciv5zyzsXy
mshKgtkc3JD/
beUWwqMpzUtx8PDtaDn+V6ju9avAVRu4wbNQWPQWOvTADQawU5TLqbOZi3wltgcKG1adPeb/
XrVwOf017fwXV99DLwGpU1vCz8cBpkNc6C06W/h/V671zf1a1DabGzhm5Vu32saeA2KKEmmW+iK36g
Hq92uIRNGL1vhbd8bNmtLeIYq56IrlU9kUazF6B7jQwBo5yJJEkcuZniCRoALECWHnDi7JIwg8GYoY
QKGK7XKsFKH/+rj6Zb2KDqPUU46HRqJrSHFxxEjTmay414FrW4O8urFi5ePnr989PvLx49fPvp1ufa
23GWUhHm5Nz9988/TL52/f/vxzZNv7XiRx7/+5avXf/z5X+qlQeu7Z6+fP3v1/dd//
fzEAu9ydJiHH5AYC+c6PnZusRg2aFkAH/
L3kziIEMlLdJNQoAQpGQt6ICMDfX2BKLLgeti04x0O6cIGvDS/ZxDej/
hcEgvwWhQbwD3GaI9x656uqbXyVpgnoX1xPs/jbiF0ZFs72PDyYD6DuCc2lUGEDZo3KbgchTjB0lFz
bIqxRewuIYZd98iIM8Em0rlLnB4iVpMckEMjmjKhyyQGvyxsBMHfhm327jg9Rm3q+/
jIRMLZQNSmElPDjJfQXKLYyhjFNI/cRTKykdxf8JFhcCHB0yGmzBmMsRA2mRt8YdC9BmnG7vY9uohN
JJdkakPuIsbyyD6bBhGKZ1bOJIny2CtiCiGKnJtMWkkw84SoPvgBJYXuvkOw4e63n+3bkIbsAaJm5t
x2JDAzz+OCThC2Ke/y2EixXU6s0dGbh0Zo72JM0TEaY+zcvmLDs5lh84z01QiyymVss81VZMaq6idY
QK2kihuLY4kwQnYfh6yAz95iI/
EsUBIjXqT5+tQMmQFcdbE1XuloaqRSwtWhtZO4IWJjf4Vab0bICCvVF/
Z4XXDDf+9yxkDm3gfI4PeWgcT+zrY5QNRYIAuYAwRVhi3dgojh/kxEHSctNrfKTcxDm7mhvFH0xCR5
awW0Ufv4H6/2gQrj1Q9PLdjTqXfswJNUOkXJZLO+KcJtVjUB42Py6Rc1fTRPbmK4RyzQs5rmrKb539
c0Ref5rJI5q2TOKhm7yEeoZLLiRT8CWj3o0Vriwqc+E0LpvlxQvCt02SPg7I+HMKg7Wmj9kGkWQXO5
nIELOdJthzP5BZHRfoRmsExVrxCKpepQODMmoHDSw1bdaoLO4z02Tker1dVzTRBAMhuHwms1DmWaTE
cbzewB3lq97oX6QeuKgJJ9HxK5xUwSdQuJ5mrwLST0zk6FRdvCoqXUF7LQX0uvwOXkIPVI3PdSRhBu
ENJj5adUfuXdU/d0kTHNbdcs22srrqfjaYNELtxMErkwjODy2Bw+ZV+3M5ca9JQptmk0Wx/D1yqJbO

QGmpg95xjOXN0HNSM067gT+MkEzXgG+oTKVIiGSccdyaWhPySzzLiQfSSiFKan0v3HRGLuUBJDrOfd
QJOMW7XWVHv8RMm1K5+e5fRX3sl4MsEjWTCSdWEuVWKdPSFYddgcSO9H42PnkM75LQSG8ptVZcAxEX
JtzTHhueDOrLiRrpZH0Xjfkh1RRGcRWt4o+WSewnV7TSe3D810c1dmf7mZw1A56cS37tuF1EQuaRZc
IOrWtOePj3fJ51hled9glabuzVzXXuW6olvi5BdCjlq2mEFNMbZQy0ZNaqdYEOSWW4dm0R1x2rfBZt
SqC2JVV+re1ottdngPIr8P1eqcSqGpwq8WjoLVK8k0E+jRVXa5L505Jx33QcXvekHND0qVlj8oeXWv
Umr53Xqp6/v16sCvVvq92kMwioziqp+uPYQf+3SxfG+vx7fe3cerUvvciMVlpuvgshbW7+6rteJ39w
4Byzxo1IbtervXKLXr3WHJ6/dapXbQ6JX6jaDZH/
YDv9UePnSdIw32uvXAawxapUY1CEpeo6Lot9qlplerdb1mtzXwug+Xtoadr75X5tW8dv4FAAD//
wMAUEsDBBQABgAIAAAAIQBOdcfYCAQAAKwLAAARAAAAd29yZC9zZXR0aW5ncy54bWy0Vttu4zYQfS/
QfzD0XEd3JxHWWSSO3WQbd4vYRZ8pibIJ8yKQVBzvov/eISVabuMu7F3kJaHmzJwZDg+H/
vDxldHBC5aKCD72wovAG2BeiJLw1dj7czkbXnkDpREvERUcj70dVt7Hm59/+rDNFNYa3NQAKLjKWDH
21lrXme+rYo0ZUheixhzASkiGNHzKlc+Q3DT1sBCsRprkhBK986MgGHkdjRh7jeRZRzFkpJBCiUqbk
ExUFSlw989FyFPytiH3omgY5tpm9CWmUIPgak1q5djY97IBuHYkL9/
axAujzm8bBidsdytkuY84pTwTUEtRYKXggBh1BRLeJ07eEO1zX0DubouWCsLDwK4OK0/PI4jeEIwK/
Hoex1XH4UPkIQ8pz+MZ7XlI39hw9H3FHBCoUpfrs1gi11ffxCKN1kjtVWQY8XlFpXu6Het7pOgpqmm
hJ5JLJNs72UmGFdnjiguJcgrlgHQGcPoDW535C000/+wSv1q76YN3AzPiixBssM1qLAu4KDBg4pHnG
wDkKaqFRhoospVEDAbD2CsoRrx1KHGFGqqXKF9oUYPTC4JdXEZBCxdrJFGhsVzUqACRTwTXUlDnV4r
fhZ7AkJFwB7oIO3LMqlF4Nn1CO9HoA2TRjjNg4IjBPv81ouaihHkDoZKcfiAmwFYTpocl/DeRgPErS
YmXpr8LvaN4BptZkC/4lpefGqUJMNpB9QMVfKsAzE3mz6CI5a7GM4x0A217p2T2ZGaU1HMipZCPvAR
hvFsyUlVYQgICQpuDnIgUW9vnB4xKePXeKS8o7C9whgsZL0GmmzuhtWAPu3oNvf6xk7T69w/
lDG93qdziWQi9dw2uoyS9itpKDdojQRQHwewYEgajOJ0eQ+I4hcBjyFUYzi6PIv9fwXSSTiPbRX+/
B5aZV/QP6VbmIgxYGzFBLJcEDebmnfWNRy43d4Q7PMcwzPAhsmhyBw6HLaAYonQGR+IAWzTLSqLqe1
zZNZ0juep5Ow951ApT6tOey4w4LH+VoqlbdCtR3QrcuYRJ0kUSrp8Ic3bV5AsXxWH8HkANLz+/
SNunvj3bTINg7KB4QlZ41hfz4W/
PRioYKX2rCBp7G2FMrVapXBid4Tmq61au+Soce5Ss1jo0YRq+SviFZj/yVdRhkcWiFrMfqDCbBe9u0
dsiZzvwi50t7m2JsyW9LXW2tLeNnG1kbGsYUJISvoGb45bGXglKxRaXDz3+xuTelYKACBY7lvfPxUW
LUaLgKtfwsmghHfaLxcIkK0XxaF66pLUnYTC9D9PLFk7ti6TtbYfWPuPqDilcdpgLTdvQr8k0md5eB
ukwjq+vh7dBEg+vruN4GN9dxun1fZSGk+Dv7mq4n9k3/wAAAP//AwBQSwMEFAAGAAgAAAAhAD83wgx
6AQAA6QIAABEACAFkb2NQcm9wcy9jb3JlLnhtbCCiBAEooAABAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
JySP07DMBTGdyTuEHlP7aQCVVGSSoA6UQmJIhCbsV9b08SxbLdpd66AxCEYWLhUJe6AkzRpKzqxvT+
/9/n5s+PhOs+8FWgjCpmgoEeQB5IVXMhZgh4mI3+APGOp5DQrJCRoAwYN0/OzmKmIFRrudKFAWwHGc
0rSREwlaG6tijA2bA45NT1HSNecFjqn1qV6hhVlCzoDHBJyiXOwlFNLcSXoq04R7SQ56yTVUme1AGc
YMshBWoODXoD3rAWdm5MDdeeAzIXdKDiJts2OXhvRgWVZ9sp+jbr9A/
w0vr2vr+oLWXnFAKUxZ5EVNoM0xvvQRWb58grMNuUucTHTQG2h0+3X+8/H2/b7sybaauX3AjZloblx
s0eZwzgYpoWy7hUb5aOCozNq7Ng961QAv9ocHvK3WfEaVqL6E2lYE10a7wxuFgPuOWOixsa289i/
vpmMUBqSsO+TgU/6kyCMLgYRIc/Vbkfze8F8t8C/FVuBxp7jz5n+AgAA//8DAFBLAwQUAAYACAAAAC
EA7dspD0kCAACsBwAAEgAAAHdvcmQvZm9udFRhYmxlLnhtbNyTzY7TMBDH70i8g+X7Nk76XW262q8i
JNjDankA13ESC39Ettu0N14BCY5I3HgA3ordh8B20lJRVmwQ4kAOjj3j+Xnm7/HUN14dCaKrOj1U+h1UpnlV
cQBFQSlTFZpPDN3eJkAoGxWGaYK0lTuKUGns2fPzutZ7mS1gAXL81MkBSW1lazKDCKkpAKbnqqodM5c
aYGtW+oiEli/XVUnRIkKW7ZknNltlCA0gi1GP4Wi8pwReqXISlBpQ3ykKXdEJU3JKrOj1U+h1UpnlV

aEGuNqFrzhCczkHhMPjkCCEa2Mym3PFdNmFFAuPEZhJvgPwLAbIDkCjAjddGNMWkbkIg85LOvGGe05
LDvg/FkyBwCT2azsREl2ukY+FltcYlMeEmm3pIZ73FZ4jQSZvSyk0njJHcndOnAXBwLYj65+/wtTug
l2XwKct08B1DOJhYu8xJwtNQuOCktlaOx8a8xT6GpYoCHytSRogPp+hJHfSEqsDfWQZiNqzDkWjG93
VlMzYxpHxSwpd/Y11sxn3bgMK5xjZZYohdcIoeR8sYCNJXbZectgfNFaEn9W+Katpb+3IG8hgROWcc
MhgbPf486MGgWOlHiNebGS4IWyJSNBD8ztjXPtEr//8v7h0zvw7evn+w8f28KOFLtwinml/
OfHXys2if+OYtNwzPmBYsn0ajy+XBwpFie/UWziN3VT7I4JasANrcGtElg+0kMJGjlFhq6DfC/1O/
WQDtzOPXT9cw+NJ8N/0kPtawKvWFHaR9+U74//9E21EzP/DgAA//8DAFBLAwQUAAYACAAAACEA7wop
Tk4BAAB+AwAAFAAAAHdvcmQvd2ViU2V0dGluZ3MueG1snNNfa8IwEADw98G+Q8m7psOUKVZhDMdexm
DbB4jp1YYluZKLq+7T79qpc/hi95L/9+MuIfPlztnkEwIZ9LkYDVORgNdYGL/JxfvbajATCUXlC2XR
Qy72QGK5uL2ZN1kD61eIkU9SwoqnzOlcVDHWmZSkK3CKhliD580Sg1ORp2EjnQof23qg0dUqmrWxJu
7lOE2n4sCEaxQsS6PhAfXWgY9dvAxgWURPlanpqDXXaA2Gog6ogYjrcfbHc8r4EzO6u4Cc0QEJyzjk
Yg4ZdRSHj9Ju5OwvMOkHjC+AqYZdP2N2MCRHnjum6OdMT44pzpz/JXMGUBGLqpcyPt6rbGNVVJWi6l
yEfklNTtzetXfkdPa08RjU2rLEr57wwyUd3LZcf9t1Q9h1620JYsEfAutonPmCFYb7gA1BkO2yshab
l+dHnsg/v2bxDQAA//8DAFBLAwQUAAYACAAAACEAcg8mJnwLAADKcgAADwAAAHdvcmQvc3R5bGVzLn
htbLydTXPbOBKG71u1/4Gl0+4hkb+duMaZcpxk7JrY44mczRkiIQlrENDyI7bn1w8IUhLkJig22OtL
YknshyBevA00SVG//PqUyugnz3Kh1flo/+3eKOIq1olQ8/PR9/svb96NorxgKmFSK34+eub56NcP//
zHL49nefEseR4ZgMrP0vh8tCiK5dl4nMcLnrL8rV5yZT6c6SxlhXmZzccpyx7K5ZtYp0tWiKmQonge
H+ztnYwaTNaHomczEfNPOi5TrgobP864NESt8oVY5ivaYx/ao86SZaZjnufmoFNZ81Im1BqzfwRAqY
gznetZ8dYcTNMiizLh+3v2r1RuAMc4wAEAnMT8Ccd41zDGJtLliATHOVlzROJwwhrjAPKkSBYoysGq
X8dVLCvYguULl8hxjTpe457Tqo/S+Ox6rnTGptKQjOqRES6y4Opfc/zVf/ZP/mTfrw5h9MF4IdHxJz
5jpSzy6mV2lzUvm1f2vy9aFXn0eMbyWIjz0SWTYpqJkXmHs7y4yAU7H90wOS9V9JsuFiKuPlpcqHx7
4zg/H92L1Fjvlj9G33TK1Ghc7UAyNTef/2TyfMTVm9+/baMfdPPWVCSGyLI3k4sqcNy0sP7fafdy/
are6sVBGsMZ+03qLGA+5bOvOn7gyaQwH5yP9qpdmTe/X99lQmfG6eej9+
+bNyc8FVciSbhyNlQLkfAfC66+5zzZvP/nF+vW5o1Yl8r8fXh6Yjte5snnp5gvK++bTxVLza5vqwBZ
bV2Kzc5t+P9WsP2mz9riF5xVCTDaf4mwzUchDqqI3Dnadmb54tjtVqgdHb7Wjo5ea0fHr7Wjk9fa0e
lr7ejda+3IYv6fOxIqMbnWbg93A6i7OB43ojkes6E5Hi+hOR6roDkeJ6A5noGO5njGMZrjGaYITqFj
3yh0BvuhZ7R3c3fPEWHc3VNCGHf3DBDG3Z3ww7i783sYd3c6D+Puzt5h3N3JGs+tl1rRtbGZKga7bK
Z1oXTBo4I/DacxZVi2KqThVZMez0gOkgBTZ7ZmIh5Mi5l9vXuEWJOGz+dFVVxFehbNxLzMeD644Vz9
5NKU9RFLEsMjBGa8KDNPj4SM6YzPeMZVzCkHNh1UCsUjVaZTgrG5ZHMyFlcJcfetiCRJYT2gWVksKp
MIgkGdsjjTw5umGVl++Cry4X1VQaKPpZSciHVLM8Qsa3htYDHDSwOLGV4ZWMzwwsDRjKqLGhpRTzU0
og5raET9Vo9Pqn5raET91tCI+q2hDe+3e1FIm+LdVcd+/3N3l1JX5/
EHt2Mi5oqZBcDw6aY5ZxrdsYzNM7ZcRNWJ4Hase8zY/XzUyXN0TzGnrUlU63o7RC7NUQtVDu/
QLRqVudY8InuteUQGW/OGW+zGLJOrBdoVTT0zKadFq2ktqZdpJ0yW9YJ2uNtYMXyEbQzwRWQ5mQ3as
QQj+LZazlZyUmS+TSuHN2zDGm6rl1mJtHkNkqCVUscPNGn46nnJM1OWPQwmfdFS6kee0BEnRabrseZ
a/sBK0svyn9PlguXC1kpbiP5T/eoOgOiGLQcf0J1kQtHo9vlNyoSM6FYQV/
c3X6N7vazKzKpjaIAfdVHolIzZnAn81w8+/TdNAy9MEayeiY72guj0kIVdCoJJpibphIhklplCCZI5
1PJ+589TzbKEhnaX8fqmm4ITEScsXdaLDgJvmbz4aPIPwWrI8v7DMlGdF6Iy1T0JzDltmJfT//
J4eKq71RHJmaE/
ysKef7RLXRtNhxu+TNjCDV8iWDXN9FCNX4KD3cINP9gtHNXBXkqW58J7CTWYR3W4Kx718Q4v/hqelj
qblZKuA1dAsh5cAcm6UMsyVTnlEVse4QFbHvXxEg4ZyyM4JWd5v2UiIRPDwqiUsDAqGSyMSgMLIxVg
+B06Dmz4bToObPi9OjWMaAngwKjGGen0T3SVx4FRjTMLoxpnFkY1ziyMapwdfor4bGYWwXRTjIOkGn
MOkm6iUQVPlzpj2TMR8rPkc0ZwgrSm3WV6Vn0bQ6v6Jm4CZHWOWhIutmsclcg/
+JSsaRWLsl0EZ0SZlFoTnVvbTDg2cvvetV1h9wueDi+j7ySL+ULLhGeeY/
LHmnp5smRxc5oeXO7rddrzq5gvimiyWJ/tdzEnezsjVwX7VtjuHbb1+clBR9gNT0SZrhoKv0xxctg/

2I7oreCj3cGblcRW5HHPSLjPk92Rm1XyVuRpz0i4z3c9I61PtyK7/PCJZQ+tA+G0a/
ysazzP4DvtGkXr4Nbddg2kdWTbEDztGkVbVoku4ri6WgDV6ecZf3w/8/
jjMS7yUzB28lN6+8qP6DLYN/5TVDM7Jmna/
a3vngB53y6ie2XOP0tdn7ffuuDU/0td12bhpHIetXIO+1+42soy/n7snW78iN55x4/
onYD8iF6ZyBuOSkl+Su/
c5Ef0TlJ+BDpbwRkBl61gPC5bwfiQbAUpIdlqwCrAj+i9HPAj0EaFCLRRB6wU/AiUUUF4kFEhBW1Ui
EAbFSLQRoULMJxRYTzOqDA+xKiQEmJUSEEbFSLQRoUItFEhAm1UiEAbNXBt7w0PMiqkoI0KEWijQgT
aqHa9OMCoMB5nVBgfYlRICTEqpKCNChFoo0IE2qgQgTYqRKCNChEoo4LwIKNCCtqoEIE2KkSgjVp/1
TDcqDAeZ1QYH2JUSAkxKqSgjQoRaKNCBNqoEIE2KkSgjQoRKKOC8CCjQgraqBCBNipEoI1qLxYOMCq
MxxkVxocYFVJCjAopaKNCBNqoEIE2KkSgjQoRaKNCBMqoIDzIqJCCNipEoI0KEV3js7lE6bvNfh9/1
tN7x37/S1dNo765X+V2UYf9UatW+Vn9v4vwUeuHqPWLh4e23ugHEVMptD1F7bms7nLtLRGoC59/
XHZ/w8elD3zoUvNdCHvNFMCP+kaCcypHXUPejQRF3lHXSHcjwarzqCv7upFgGjzqSrrWl6ubUsx0BI
K70owTvO8J78rWTjjs4q4c7QTCHu7KzE4g7OCufOwEHkdVcn4Zfdyzn07W95cCQtdwdAinfkLXsIRa
rdIxNEZf0fyEvur5CX1l9BNQenoxeGH9KLTCflSY1NBmWKnDjeonYKWGhCCpASZcaogKlhqiwqSGiR
ErNSRgpQ5Pzn5CkNQAEy41RAVLDVFhUsOpDCs1JGClhgSs1AMnZC8mXGqICpYaosKkhos7rNSQgJUa
ErBSQ0KQ1AATLjVEBUsNUWFSgyoZLTUkYKWGBKzUkBAkNcCESw1RwVJDVJfU9izKltQohZ1w3CLMCc
RNyE4gLjk7gQHVkhMdWC05hMBqCWq10hxXLbmi+Ql91fMT+sroJ6D09GLwwvpRaIX9qDCpcdVSm9Th
RvUTsFLjqiWv1LhqqVNqXLXUKTWuWvJLjauW2qTGVUttUocnZz8hSGpctdQpNa5a6pQaVy35pcZVS2
1S46qlNqlx1VKb1AMnZC8mXGpctdQpNa5a8kuNq5bapMZVS21S46qlNqlx1ZJXaly11Ck1rlrqlBpX
LfmlxlVLbVLjqqU2qXHVUpvUuGrJKzWuWuqUGlctdUqNq5ZuTIggeATUJGVZEdE9L+6K5YuCDX844X
eV8VzLnzyJaA/1K+oox49bP39Vse3P4ZntC9Nn1RPQna8rJfUTYBug3fA6Wf9MVRVctSRqfrqreds2
uLlca/9ufkbsgWdV2+2mzUjI/1q/0VzuzP+6rH6Ty3nP+ZUv2wDY5Hhh2hw3z8DyNLl5lu36y1j2Sb
YvD8DzwFvbsM1AXm3dSLPp93q7rV6v2+9pd1EZp6PN1lidfV17z9fA900y2dVC056prDUyf1yrxAAe
m58dq1uaPLEaZT6/5FLesHprvfRvKvmsqD/d37OPPnjx+bR+ip83PrPp3gsYbzemftk9Turn+jf3IX
iHdpXTWrrb3hQztKc3bVv9lX/4GwAA//8DAFBLAwQUAAYACAAAACEAO+eBcs8BAADYAwAAEAAIAWRv
Y1Byb3BzL2FwcC54bWwgogQBKKAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACcU8Fu2zAMvQ/
YPxi6N0qCoRgCRUWRYuhhWwPEbc+sTCfCZEmQ2KDZ14+yG8/ZdppP75HU0yMpq5u3zlVHTNkGvxaL2
VxU6E1orN+vxWP95eqzqDKBb8AFj2txwixu9McPaptCxEQWc8USPq/
FgSiupMzmgB3kGac9Z9qQOiCmaS9D21qDd8G8duhJLufza4lvhL7B5iqOgmJQXB3pf0WbYIq//FSfI
utpVWMXHRDq7+WkmzWBOiXHqKoDgatth3rO4ZGoLewx64WSA1DPITU9H4DaHCCBIZ6fXiyVnFB1G6O
zBogHq79Zk0IOLVUPvduqHFdyWqK4gx2a12TpVExMqfpq/WBjAGwrwT5BPLx7G5naGXC44d51Cy6jk
r8D6h6h7HULtvg70uqIhkKqsv3Jm12K6gUylomtxRGSBU9iKBtIj13MlHRtybH2yHs4LZti+6mYHMB
lYU96D4wv3fU35IeWe6N/
mF1MzfYeBqsTO1Nn5zv+UN2ELoLn+coR8YB/5MdYh7vyLt5neBmcLP3Z0mEXwZTlXKx/
klA7jmLD+xxXMgbUPTeQXJHns36Pzbnm70R5UE/Dj6oX17M5f/0LOsf4HYx/kP4FAAD//wMAUEsBAi
0AFAAGAAgAAAAhAN+k0mxaAQAAIAUAABMAAAAAAAAAAAAAAAAAAAAAAAFtDb250ZW50X1R5cGVzXS54
bWxQSwECLQAUAAYACAAAACEAHpEat+8AAABOAgAACwAAAAAAAAAAAAAAAAAACTAwAAX3JlbHMvLnJlbH
NQSwECLQAUAAYACAAAACEA1mSzUfQAAAAxAwAAHAAAAAAAAAAAAAAAAAAAACzBgAAd29yZC9fcmVscy9k
b2N1bWVudC54bWwucmVsc1BLAQItABQABgAIAAAAIQDG7Dw5AwMAAKULAAARAAAAAAAAAAAAAAAAAO
kIAAB3b3JkL2RvY3VtZW50LnhtbFBLAQItABQABgAIAAAAIQC29GeY0gYAAMkgAAAVAAAAAAAAAAAA

AAAAABsMAAB3b3JkL3RoZW1lL3RoZW1lMS54bWxQSwECLQAUAAYACAAAACEATnXH2AgEAACsCwAAEQ
AAAAAAAAAAAAAAAAAgEwAAd29yZC9zZXR0aW5ncy54bWxQSwECLQAUAAYACAAAACEAPzfCDHoBAADp
AgAAEQAAAAAAAAAAAAAAAABXFwAAZG9jUHJvcHMvY29yZS54bWxQSwECLQAUAAYACAAAACEA7dspD0
kCAACsBwAAEgAAAAAAAAAAAAAAAAAIGgAAd29yZC9mb250VGFibGUueG1sUEsBAi0AFAAGAAgAAAAh
AO8KKU5OAQAAfgMAABQAAAAAAAAAAAAAAAAAAAgRwAAHdvcmQvd2ViU2V0dGluZ3MueG1sUEsBAi0AFA
AGAAgAAAAhAHIPJiZ8CwAAynIAAA8AAAAAAAAAAAAAAAAAAAAR4AAHdvcmQvc3R5bGVzLnhtbFBLAQIt
ABQABgAIAAAAIQA754FyzwEAANgDAAAQAAAAAAAAAAAAAAAAAAAKopAABkb2NQcm9wcy9hcHAueG1sUE
sFBgAAAAALAAsAwQIAAK8sAAAAAA==

langchain4j\src\test\resources\test-file.ppt

BASE64:

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAPgADAP7/
CQAGAAAAAAAAAAAAAAADAAAAAQAAAAAAAAAAEAAAAgAAAAEAAAD+////AAAAAAAAAAAGAAAABwAAAP
//
//
//
//
//
//
//
////////////////////9////TwAAAP7///8EAAAABQAAABYAAAD9/////f///wkAAAAKAAAACwAAAA
wAAAANAAAADgAAAA8AAAAQAAAAEQAAABIAAAATAAAAFAAAABUAAAADAAAAFwAAABgAAAAZAAAAGgAA
ABsAAAAcAAAAHQAAAB4AAAAfAAAAIAAAACEAAAAiAAAAIwAAACQAAAAlAAAAJgAAACcAAAAoAAAAKQ
AAACoAAAArAAAALAAAAC0AAAAuAAAALwAAADAAAAAxAAAAMgAAADMAAAA0AAAANQAAADYAAAA3AAAA
OAAAADkAAAA6AAAAOwAAADwAAAA9AAAAPgAAAD8AAABAAAAAQQAAAEIAAABDAAAARAAAAEUAAABGAA
AARwAAAEgAAABJAAAASgAAAEsAAABMAAAA/v///04AAAD+/////v////////////////////////////
//
//
//1
IAbwBvAHQAIABFAG4AdAByAHkAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAWAAUA/////////8BAAAAEI2BZJtPzxGG6gCqALkp6AAAAAAAAAAAAAAAAPDdzmIJxtkBTQ
AAAIADAAAAAAAAUABvAHcAZQByAFAAbwBpAG4AdAAgAEQAbwBjAHUAbQBlAG4AdAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAACgAAgECAAAAAwAAAP////8AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAIAAAAeI4AAAAAAAAFAFMAdQBtAG0AYQByAHkASQBuAGYAbwByAG0AYQB0AGkAbwBu
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAKAACAQQAAAD/////////wAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAQAAAAAAAAUARABvAGMAdQBtAGUAbgB0AFMAdQBtAG0A
YQByAHkASQBuAGYAbwByAG0AYQB0AGkAbwBuAAAAAAAAAAAAAAA4AAIB//////////////AAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AP7///8GAAAABwAAAAgAAAAJAAAACgAAAAsAAAAMAAAA/v////7/////////////////////////////
//
//
//
//
//
//
////////////////////////////////////VOBuOI8YEhCyt8+GDNK8GxeigkjspAmYUCjgsfS
AWLy4ipYkIW4TjralOZLnBL6Y8ldLXpw5rGP1vjXviTP2mfclnWZ4jNyEu8vFkiAv3UrP5TBPUnVEl
RyPQAs8puQgu0al8WIeTkq8FSeGqg8qwGBioR2w4FcmqZH2Pv6icMkoX2XmjmmrRKBUX1FfyTjl2AM
QYjlm0CY0O6c9kPMB9ie/YhGQc2X9xDuYieYqKDkiSuwDJ17mrvzoV8RAMpwW3jHyicB8FdBm/ja5w
FCSuOxoYInuAzRopiHmjXIVkLWGpb92O2pLVRoebuAX2wWCmcZ7oTo1btqAfsalXey3w+ESS0msrIo

lWrETHV4Z0PRP20uJW1mfb4BO3z7xacdHe1OFf7DrtZo9ZeDjGltzs10/
P4gLDeF58xMEaTQJsUtOc89T1D00eKGpVxbn412gyng2H06x3u8jz3myRv+/dzGZZb5INp1eLfPD+Z
jz6G+s+jqltTePzoWllrBiHednt26Y1n5qUEmSs5FL3fr9P8ywWINukPMX3vuQaIdI94Jodjj5t1lH
ibCcd3RppxABOzs7QirhT0/
xXzZP8NS434KVq6BbBVGmzcsbUUfZtuFMS8KhU+0pTwNpQ+ScO0p8XtYwN8k16AmdlXePQOhO/X+ou
MXvy3smxzCKjNV5DJf+51T0VulkLXygkJIXwXyiE7zoUs0AB0hj4v0m+aZOEOaMrB3sT39gxRLPU1Q
oc0IT97kqf8P3o1X7hP2bF4vty/6Po7fwfAAAA//8DAFBLAwQUAAYACAAAACEAjBg1ZNwAAAADAQAA
DwAAAGRycy9kb3ducmV2LnhtbFSPTUvDQBCG74L/YRnBm91EStG02yJqUQQpjaXpccxOPjQ7G3a3af
rvXbyIl4GXmXlensVqNJ0YyPnWsoJ0koaAvVqNNmkagLq1uuVaw+1jf3IHwAVljZ5kUnMnDanl5scBM2xNvachD
LSKEfYYKmhD6TEpfNmTQT2xPHHeVdQZDjK6W2uEpwk0nb5NkJg22HBsa7OmxofI7PxoFRbH5Ivt2LJ
j379Nhnev00N0rdX01Ps3jeJiDCDSGv4/
nIT3Mqv8Xv9BXrWAKono5f7pWb9EHcgqiZXSOviCXPwAAAP//AwBQSwECLQAUAAYACAAAACEA8PeKu
/0AAADiAQAAEwAAAAAAAAAAAAAAAAAAAAAAW0NvbnRlbnRfVHlwZXNdLnhtbFBLAQItABQABgAIAAA
AIQAx3V9h0gAAAI8BAAALAAAAAAAAAAAAAAAAAC4BAABfcmVscy8ucmVsc1BLAQItABQABgAIAAAAI
QAuDE6wfAMAAIcKAAAQAAAAAAAAAAAAAAAAACkCAABkcnMvc2hhcGV4bWwueG1sUEsBAi0AFAAGAAg
AAAAhAIwYNWTcAAAAAwEAAA8AAAAAAAAAAAAAAAAA0wUAAGRycy9kb3ducmV2LnhtbFBLBQYAAAAAB
AAEAPUAAADcBgAAAAAAABDwCAAAAKQPEALQCIoQDwAR8BAAAAAAAMMLCAAAAAIAAAAHAQAADwAN8FY
AAAAAAJ8PBAAAAAQAAAAAAKAPAgAAACoAAAChDxgAAAACAAAAAAAAAAAAAAgAAAAAABgAMAImJif4AA
PgPBAAAAAAAAAAAAKYPDAAAAPAAAADUAdAC8AMQBQ8ABPB7CAAAEgAK8AgAAAAFBAAAAAoAAJMAC/
BgAAAAfwABAO8BgAAA+PMBhwABAAAAvwAAAAYAvwEBABEA/wEBAAkAPwMAAAgAgMMqAAAAvwMAAAIA
RgBvAG8AdABlAHIAIABQAGwAYQBjAGUAaABvAGwAZABlAHIAIAA0AAAAEwAi8YkHAACpw4MHAABQSw
MEFAAGAAgAAAAhAPD3irv9AAAA4gEAABMAAABbQ29udGVudF9UeXBlc10ueG1slJHNSsQwEMfvgu8Q
5iptqgcRaboHq0cVXR9gSKZt2DYJmVh33950Py7iCh5n5n5v/xI6lX22kUM0W23im4LisQ5LQ31vUKPt
ZPxR0ITugMjt6Rgh0xrJrLi3r///
//
//
//
//
//
//
//
//
//
//
//
//
//
//
//
//
//DwDoA9ILAAABAO
kDKAAAAAAeAADgEAAA4BAAAIAWAAAFAAAACgAAAAAAAAAAAAAAAQAGAAAAAAAEPAPIDrAEAAC8AyA8M
AAAAMADSDwQAAAABAAAADwDVB+QAAAAAALcPRAAAAEMAYQBsAGkAYgByAGkAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQiEAC3D0QAAABBAHIAaQBhAGwA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAC
AAtw9EAAAAQwBhAGwAaQBiAHIAaQAgAEwAaQBnAGgAdAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAABCIAAKQPCAAAAIAAQAAAAAAAAAClDwwAAAAAAAAILgAAAAIAAAAAAKkPCg
AAAAcAAAAAAAAMAABAAKMPbgAAAAUA//0/AAAAIiABAGQAAAAA/
wAAZAAAAAAAAAAAAEACAAAAAAcAAAD//+8AAAAAAP//AAD//xIAAAAAAQAAAAUAACABIAEAAAAAAAU
AAEACQAIAAAAAAAUAAGADYAMAAAAAAAUAAIAEgAQAAAAADwALBHQAAAAPAADwbAAAAAAABvAgAAAAA
wgAAAMAAAAJAAAAAgAAAAEAAAAHAAAAAgAAAAMAAABjAAvwJAAAAIEBBAAACIMBAAAACL8BEAAQAMA
BAQAACP8BCAAIAAECAgAACEAAHvEQAAAA//8AAAAA/wCAgIAA9wAAEB8A8A8cAAAAAADzAxQAAAACA
AAAAAAAAAAAAAAAACAAAAAAA8A0AcoAQAAHwAUBBwAAAAAABUEFAAAAC/osfYAypo7f9yOxwDKmjs
BAQAAHwATBDwAAAAAAP0DNAAAAAEAAAABAAAAAQAAAAEAAABYKgdrAQAAAAEAAAAAAAAAICcHawEAA
AAAAAAAAAAAAAAB2sPAPoDRwAAAAAA/gMDAAAAAAAAAAD9AzQAAAB5AAAAZAAAAHkAAABkAAAAWCo
HawEAAAAgJwdrAQAAAIB3SAEAYAAA6AIAALAAAAABAAdrDwCIE2kAAAAPAIoTKQAAAAAAug8QAAAAX
wBfAF8AUABQAFQAMQAyAAAAixMJAAAAAAAlBAEAAAAADwCKEzAAAAAALoPEAAAAF8AXwBfAFAAUAB
UADEAMAAAAIsTEAAAAAAADQQIAAAAcLUAAHC1AAA/ANkPDAAAAAAA2g8EAAAADQACAE8A2Q8MAAAAA
ADaDwQAAAANAAIADwDwDxwAAAAAAPMDFAAAAAMAAAAEAAAAAAAAAAAABAAAAAAAAAAAoBMIHAABQSwM
EFAAGAAgAAAAhAFauB8P3AAAAqQEAABMACAJbQ29udGVudF9UeXBlc10ueG1sIKIEAiigAAIAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAHyQzU7DMBCE70i8g+Urih04IIS
a9MDPEZAoD7A4m8Sq/+TdVs3b46RFQqhwsta7M/NpVuuzIdCUkUdaBGjszpXmsyI3ogFROGsulj9sBlzINOYLYwoL6p61ttYmAMXPHsIdv
VI/awcyyeDuX7SJLRkRQPx8M5q5GQkrMGuJDqfeh+pVSnBFWUyw2NNtFVwZD6bMK8+TvgpHst1WTbo
XiDzC/gC4Zm+HT4zpNDUv+bnKGMfW8NdtHsfGlApYxU3gXYO/XD+ptcL0W3XwAAAP//AwBQSwMEFAA
GAAgAAAAhAO3kDEu7AAAAJgEAAAsACAJfcmVscy8ucmVscyCiBAIooAACAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACEj80KwjAQhO+C7xD2blM9iEjT1TXkTwqvUB1nT7g2k
SklXs25tjC4LH2WG+2Smqz2jEm0IcnFWwzXIQZLVrBtspuNfnzQFEZLQNGmdJwUQRqnK9Kq5kkFMo9
oOPIlFsVNAz+6OUUfc0YsycJ5uc1oUROcnQSY/6iR3JXZ7vZZgzoFwwxaVREC7NFkQ9+dT8n+3adtB
0cvo1kuUfFZLxYejGk0krRI2hI1YwO2bpW5BlIRfryi8AAAD//wMAUEsDBBQABgAIAAAAIQDY/Y2Pr
AAAALYAAAAPAAAAdGFibGVTdHlsZXMueG1sDMxJDoIwGEDhvYl3aP59LUNRJBTCICt36gEqlCHpQGi
jEuPdZfnyki/P4n13+ABEro13aQHBo97g2NA1cWm0YLAKC3m95c+6sUV+vQp

```
mibcAajc3NCiG1Hobg9mFno7fVmUdxtuQykW/
h705UkgecdieKTBtSJnsE3qoIgorTAp8vliGlIA1x6NMZxVNbVuan9Kix+QLI/AAAA//8DAFBLAQIt
ABQABgAIAAAAIQBWrgfD9wAAAKkBAAATAAAAAAAAAAAAAAAAAAAAAABbQ29udGVudF9UeXBlc10ueG
1sUEsBAi0AFAAGAAgAAAAhAO3kDEu7AAAAJgEAAAsAAAAAAAAAAAAAAAAAMAMAAF9yZWxzLy5yZWxz
UEsBAi0AFAAGAAgAAAAhANj9jY+sAAAAtgAAAA8AAAAAAAAAAAAAAAAAHAYAAHRhYmxlU3R5bGVzLn
htbFBLBQYAAAAAAwADALcAAAD1BgAAAAAAAOoDAAAAAA8A+ANPfwAAAgDvAxgAAAABAAAAAQIHCQgA
AAAAAAAAAAAAAAIAAABgAPAHIAAAAP///wAAAAA5+bmAERUagBEcsQA7X0xAAVjwQCVT3IAAACjDz
4AAAABAP/9PwAAACIgAQBkAAAAAP8AAFoAAAAAAAAAABAAgAAAAAHAAAA///vAAAAAgD//
wIA//8sAAAAAAEAABAAow92AAAABQD//T8AAwAiIAEAZAAAAAD/AABaALD/
AACQAAAAQAIAAAAABwAAAP//7wAAAAAA//8AAP//HAAAAAABAAAAJQAA2P+wASABAAACABgAAAUAAN
ACQAIAAAIAFAAABQAA8ANgAwAAAgASAAAFAAAQBYAEAAAAACAAow9uAAAABQD//
T8AAAAiIAEAZAAAAAD/AABkAB4AAAAAAAAAQAIAAAAABwAAAP//7wAAAAAA//8AAP//DAAAAAABAAA
ABQAAIAEgAQAAAAAABQAAQAJAAgAAAAAABQAAYANgAwAAAAAABQAAgASABAAAAABAAKMPbgAAAAUA/
/0/AAAAIiABAGQAAAAA/wAAZAAAAAAAAAAAAEACAAAAAAcAAAD//+8AAAAAAP//AAD//xIAAAAAAQA
AAAUAACABIAEAAAAAAAUAAEACQAIAAAAAAAUAAGADYAMAAAAAAAUAAIAEgAQAAAAAUACjD1IAAAAFA
AAAAQkAAAIAAQAAAAAAAAABAAEJAAACAAEAIAEAAAAAAgABCQAAAgABAEACAAAAAAMAAQkAAAIAAQB
gAwAAAAAEAAEJAAACAAEAgAQAAAAAYACjDwwAAAABAAAAAAAAAAAAAABwAKMPPgAAAAUAAAAAAAAAA
AACABgAAQAAAAAAAAACABQAAgAAAAAAAAACABIAAwAAAAAAAAACABAABAAAAAAAAAACABAAgACjDz4
AAAAFAAAAAAAAAAAAAgAUAAEAAAAAAAAAAgASAAIAAAAAAAAAAgAQAAMAAAAAAAAAAgAOAAQAAAAAA
AAAAgAOAAAAIwRjBgAAUEsDBBQABgAIAAAAIQCR783y+wAAALsBAAATAAAAW0NvbnRlbnRfVHlwZXN
dLnhtbHyQyU7DMBCG70i8g+Urip1yQAgl6YHlxnIoDzByJomFN3ncqn17JmmRKlQ4jWb9/vmb9d47s
cNMNoZWrlQtBQYTexvGVn5uXqp7KahA6MHFgK08IMl1d33VbA4JSfB2oFZOpaQHrclM6IFUTBi4M8T
soXCaR53AfMGI+rau77SJoWAoVZlvyK55wgG2rojnPZePSjI6kuLxODizWgkpOWugsFK9C/0vSnUiK
N5cZmiyiW5YhtQXCXPnb8Bp752tybZH8QG5vIFnGbrPpMlx8RWosHPnyUr9f/
aC7jgM1mAfzdazJyplJI7LC96pM9DPL3qxvvsGAAD//
wMAUEsDBBQABgAIAAAAIQCO6ir6vgAAADgBAAALAAAAX3JlbHMvLnJlbHOEj8EKwjAQRO+C/xD2btN
6EJGmvYjQgxfRD1iSbRtsk5CNon9vjhYEj8Mwb2bq9jVP4kmRrXcKqqIEQU57Y92g4HY9bfYgOKEzO
HlHCt7E0DbrVX2hCVMO8WgDi0xxrGBMKRykZD3SjFz4QC47vY8zpizjIAPqOw4kt2W5k/GbAc2CKTq
jIHamAnF9h9z8n+373mo6ev2YyaUfFZIna+iMnChmLMaBkgIT+dtYiKrI+0E2tVz8bT4AAAD//wMAU
EsDBBQABgAIAAAAIQCgLd67MgMAAMEeAAAhAAAAZHJzL3NsaWRlTWFzdGVycy9zbGlkZU1hc3RlcjE
ueG1s7Jlfb5swEMDfJ07IgLQau2Ul1VT1HYfwAEnsBqDbKdr9rTPso+2T7TzYAAAD//wMAUEsDBBQA
BgAIAAAAIQCgLd67MgMAAMEeAAAhAAAAZHJzL3NsaWRlTWFzdGVycy9zbGlkZU1hc3RlcjE
ueG1s7Jlfb5swEMDfJ07IgLQau2Ul1VT1HYfwAEnsBqDbKdr9rTPso+2T7TzYKAAD//wMAU
EsDBBQABgAIAAAAIQCgLd67MgMAAMEeAAAhAAAAZHJzL3NsaWRlTWFzdGVycy9zbGlkZU1hc3RlcjE
ueG1s7Jlfb5swEMDfJ07IgLQau2Ul1VT1HYfwAEnsBqDbKdr9rTPso+2T7TzYKAAD//wMAU
UWmutldISvQO5zMc03vq1ktuYboBO1FkZWR9i1c33xI0dX3fAbtxSROk7Iev2AW/A2siS2nGzFcYyj
HbwPaXW5ZKNa4ls5s6VReiTpeptO4xiJk68EL2fGbDT7cmXJD186n7abDcjK62n4AVLMYxaHkNWgtQ
s1HyjnDFGayvf1xUtMgWBaVaquJGLilvdJAPrtHgySy9JUvuarLGKXjEu/
LLiEo1E8cEPxsguBlIxbOBVBjZjYbNRhsbw/VT69fxqsp2v6EoMb9KkOdFE2X9gmWAMkGj/Y3TISVF
Y1D3wLQNa1Ex2bLOe15g8K8as0qAeR3PuXAmjg+f+7cPo4VM8wUuC7pL0BhupDnmgsD+tTvgeLW9hD
v6doJ+fv/RYGi5gxcpq/0Ld2Bd7sBGHe7ARn90Bx19noq+BvkkCrTyPUAePEbxCRDXWerEiSvMKpNA
vh0fiLuuP1bB04coPynk2mgnjlxxNsj9FnLI6uopNSA/
Lq+7fcjrirNBHhyQe04Q6hw1JPajHuW9QK44G+STFvLA9ftyfDulxN4L5IqzQR62kE/DRvshyv+/
KFecDfLogHzse6qgHZ7lR9ZovYhyxdkgn7aQR9FkOL4dX5b3ArnirLs0rb5MHVcyJ/yxSwOthmXjGK
bpQKH3lSDCRh+vD60fM2XfXWvqe0gUf79502p99MLGqttowqrV+vCDUEfVYJ+XGwX7Xuxgn46qehy6
```

+rw9GOjlGtSNvEhX0IOBOio23W8fUjQ8sjrqm9AfDzlad3A7qgGwjm7zDCHWcXaeBOGQpLUHPZ4024
dL9feg+e95/gsAAP//AwBQSwECLQAUAAYACAAAACEAke/N8vsAAAC7AQAAEwAAAAAAAAAAAAAAAAA
AAAAW0NvbnRlbnRfVHlwZXNdLnhtbFBLAQItABQABgAIAAAAIQCO6ir6vgAAADgBAAALAAAAAAAAAAA
AAAAAAACwBAABfcmVscy8ucmVsc1BLAQItABQABgAIAAAAIQCgLd67MgMAAMEeAAAhAAAAAAAAAAA
AAAAABMCAABkcnMvc2xpZGVNYXN0ZXJzL3NsaWRlTWFzdGVyMS54bWxQSwUGAAAAAAMAAwDJAAAAhA
UAAAAADwAMBK8iAAAPAALwpyIAABAACPAIAAAABgAAAAYEAAAPAAPwFx0AAA8ABPAoAAAAAQAJ8BAA
AAAAAAAAAAAAAAAAAAAAAAAAAgAK8AgAAAAABAAABQAAAA8ABPAEAQAAEgAK8AgAAAACBAAAAAoAAJ
MAC/BeAAAAfwABAO8BgAAA1NEBhwABAAAAvwAAAAYAvwEBABEA/wEBAAkAPwMAAAgAgMMoAAAAvwMA
AAIAVABpAHQAbABlACAAUABsAGEAYwBlAGgAbwBsAGQAZQByACAAMQAAAAAAEPAIAAAA5gAQAvAbKQ
QPABHwEAAAAAAAwwsIAAAAAAAAAAAEAAAAPAA3wXgAAAAAAnw8EAAAAAAAAAAAAAqA8gAAAAQ2xpY2sg
dG8gZWRpdCBNYXN0ZXIgdGl0bGUgc3R5bGUAAKIPBgAAACEAAAAAAAAAqg8UAAAAIAAAAAYAAAAJBA
AAAQAAAAAAAAAPAATwRgEAABIACvAIAAAAAwQAAAAKAACDAAvwVgAAAH8AAQDvAYAAAM3RAb8AAAAG
AL8BAQARAP8BAQAJAD8DAAAIAIDDJgAAAL8DAAACAFQAZQB4AHQAIABQAGwAYQBjAGUAaABvAGwAZA
BlAHIAIAAyAAAAAAAQ8AgAAAB+BBAC8BszDw8AEfAQAAAAAADDCwgAAAABAAAAAgAAAA8ADfCoAAAA
AACfDwQAAAABAAAAAACoD1IAAABDbGljayB0byBlZGl0IE1hc3RlciB0ZXh0IHN0eWxlcw1TZWNvbm
QgbGV2ZWwNVGhpcmQgbGV2ZWwNRm91cnRoIGxldmVsDUZpZnRoIGxldmVsAACiDx4AAAAhAAAAAAAN
AAAAAQAMAAAAAgANAAAAAwAMAAAABAAAAKoPFAAAAFIAAAAGAAAACQQAAAEAAAAAAAAADwAE8O8IAA
ASAArwCAAAAAQEAAAACgAAkwAL8FwAAAB/AAEA7wGAAIDy8wGHAAEAAAC/AAAABgC/AQEAEQD/
AQEACQA/AwAACACAwyYAAAC/AwAAAgBEAGEAdABlACAAUABsAGEAYwBlAGgAbwBsAGQAZQByACAAMw
AAABMAIvHtBwAAqcPnBwAAUEsDBBQABgAIAAAAIQDw94q7/QAAAOIBAAATAAAAW0NvbnRlbnRfVHlw
ZXNdLnhtbJSRzUrEMBDH74LvEOYqbaoHEWm6B6tHFV0fYEimbdg2CZlYd9/edD8u4goeZ+b/8SOpV9
tpFDNFtt4puC4rEOS0N9b1Cj7WT8UdCE7oDI7ekYIdMayay4t6vQvEIrsdKxhSCvdSsh5oQi59IJcv
nY8TpjzGXgbUG+xJ3lTVrdTeJXKpSEsGNHVLHX6OSTxu8/
pAEmlkEA8H4dKlAEMYrcaUSeXszI+W4thQZudew4MNfJUxQP7asFzOFxx9L/
lpojUkXjGmZ5wyhjSRJQ8YKGvKv1MWzIkL33VWU9lGfl98J6hz4cZ/uUjzf7PbbHuj+ZQu9z/
UfAMAAP//AwBQSwMEFAAGAAgAAAAhADHdX2HSAAAAjwEAAAsAAABfcmVscy8ucmVsc6SQwWrDMAyG7
4O9g9G9cdpDGaNOb4VeSwe7CltJTGPLWCZt376mMFhGbzvqF/o+8e/2tzCpmbJ4jgbWTQuKomXn42D
g63xYfYCSgtHhxJEM3Elg372/7U40YalHMvokqlKiGBhLSZ9aix0poDScKNZNzzlgqWMedEJ7wYH0p
m23Ov9mQLdgqqMzkI9uA+p8T9X8hx28zSzcl8Zy0Nz33r6iasfXeKK5UjAPVAy4LM8w09zU50C/9q7
/6ZURE31X/kL8TKv1x6wXNXYPAAAA//8DAFBLAwQUAAYACAAAACEALgxOsHwDAACHCgAAEAAAAGRyc
y9zaGFwZXhtbC54bWzsVV9v2zYQfx/
Q70DwdXBt2XJsC5GLJKu7YUZhxNkHOFNUrJoiCZL27Az77rsj5TjtwzA0fSiKAYJ01B15v/vdH16/O
7aKHaTzjdElz94OOJNamKrRjyX/42HRm3LmA+gKlNGy5Cfp+bv5m5+ubeEtw83aF7bk2xBs0e97sZU
t+LfGSo262rgWAi7dY9866aUOENBRq/rDweCq30Kj+RyP0oe1XTmSxMfDyrGmKnnOmYYWXf4CQbKVA
iG3RlXSsRHvd6ZpFyCUpRE73+GB/4KncvAnBvkZFKbNB4fRZOSgH8GccWmERU7tloWTRVRVQGKe0BO
omiPgY8mH3bZki/svYfkYHhTH2rWvRTm/hsLUNUOP09EUieTsVPKr0RifAUGAQh4DE4Roko+igUCL0
b0LxCK7HSsYUgr3UrIeaEIufSCXL52PE6Y8xl4G1BvsSd5U1a3U3iVyqUhLBjR1Sx1+jkk8bvP6QBJ
pZBAPB+HSpQBDGK3GlEnl7MyPluLYUGbnXsODDXyVMUD+2rBczhccfS/5aaI1JF4xpmecMoY0kSUPG
Chryr9TFsyJC991VlPZRn5ffCeoc+HGf7lI83+z22x7o/mULvc/1HwDAAD//wMAUEsDBBQABgAIAAA
AIQAx3V9h0gAAAI8BAAALAAAAX3JlbHMvLnJlbHOkkMFqwzAMhu+DvYPRvXHaQxmjTm+FXksHuwpbS
Uxjy1gmbd++pjBYRm876hf6PvHv9rcwqZmyeI4G1k0LiqJl5+Ng4Ot8WH2AkoLR4cSRDNxJYN+9v+1
ONGGpRzL6JKpSohgYS0mfWosdKaA0nCjWTc85YKljHnRCe8GB9KZttzr/ZkC3YKqjM5CPbgPqfE/V/
IcdvM0s3JfGctDc996+omrH13iiuVIwD1QMuCzPMNPc1OdAv/au/+mVERN9V/5C/
Eyr9cesFzV2DwAAAP//AwBQSwMEFAAGAAgAAAAhAFbznF4YAwAAWQcAABAAAABkcnMvc2hhcGV4bWw

```
ueG1srFVtb9owEP4+af/B8tepgxRoO9RQtZXopqEKle4HHI4DGc7Zsx0G/
Pqd7fDSfpimdhIK59xj33PPnS/XN5tasbW0rtKY8+xzlzOJQhcVLnL+43l8dsWZ84AFKI0y51vp+M3
o44drM3SG0WZ0Q5Pzpfdm2Ok4sZQ1uM/aSCRfqW0NnpZ20TFWOokePAWqVee8273o1FAhH9FRuJ6Zq
Q2WeFxPLauKnA84Q6gp5FhrLy2bKhByqVVBdp93WnDaB0RmosXKtYzgXxgVFn5Tmi/IMNQPlvLJQoB
OpLNnhkQsBDVL5reGeJXekja7nP9qwBJDTrQ3Oe+1WxOezjgm52KSMNyUtn4v09E1DHVZMorY7/auL
rpUuW3OL3oD+nUDBxjKjWciALKsfxUAghC9i0F2PogkE5OANNb5B6nfzYqFg3JupfBUWBjCeuJ8kPI
YIuqalDBDv7nTxTYg5/RPlU8d9fb6UStT/KW2O87UN3Q5/5L1+5S6j4v+4PKcFvbUM3/h8epeq5wTC
FDQOTkXVOcop3J+5rdKvpdkSFetVUbdwEAt6N6FEOFtIcsnehmaKqMbEt85rapiXCkVAPGCyXtl2Rq
Ipd9kEeMr9OnN5aBL+5LiB3DU/+QcqkaKFB0tlWS3KYZY+
+v95mLEQyidGuwkKkrGUzQoZPyvsKCZkNQ+CMGI2zPMZyRCrF2ol094CRO8s6twP1mp0d/
GTdB4TdWm0YKtm7YsARd0u6cNCgqQZFI4MyLKaMRUtIplJNhBslPEnSz3WO+SugdljTh6b0v/
F1zrnTdUtOdN7KN5M9sdzDGlcVg80oyNEA/zdHH2lUoVbeeHxGIKFkKrrJq6qvXPKslKOedc4tn3pz
SZYhOxedI6PpucIwUJQ91WK5piqGfR4mwlbfgEhL5jAmgItkAj4k4Mw1xVO/
k1LufgpKrCJ4HgqKdW6zLYQQqF4Yk6dG0int6ctOD/amUacmVJ02avVjPBVs0mRG/
t2BtxbJf0Ecn5pxrPlG+HJLxySEgO4V45hGsv1lH/eGkMPY+jjExnRn8AAAD//
wMAUEsDBBQABgAIAAAAIQBl+zXt3AAAAAMBAAAPAAAAZHJzL2Rvd25yZXYueG1sVI/BasJAEIbvBd9
hGcFb3VhssdFVSkUsBRFTaXqcZsckbXY27K4xvn2XXkovAz8z8/18i1VvGtGR87VlBZNxAoK4sLrmU
sHxbXM7A+EDssbGMim4kofVcnCzwFTbCx+oy0IpIoR9igqqENpUSl9UZNCPbUscdyfrDIYYXSm1w0u
Em0beJcmDNFhzbKiwpeeKiu/sbBTk+f6L7Os5Z37fTbtNpicfzaNSo2G/nsfxNAcRqA9/H8dit57O/
l/8Ql+0gnsQp+3109X6gD6QUxAto3P0Bbn8AQAA//8DAFBLAQItABQABgAIAAAAIQDw94q7/QAAAOI
BAAATAAAAAAAAAAAAAAAAAAAAAABbQ29udGVudF9UeXBlc10ueG1sUEsBAi0AFAAGAAgAAAAhADHdX
2HSAAAAjwEAAAsAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
YAwAAWQcAABAAAAAAAAAAAAAAAAAAAAAAKQIAAGRycy9zaGFwZXhtbC54bWxQSwECLQAUAAYACAAAACEAZ
fs17dwAAAADAQAADwAAAAAAAAAAAAAAAAAAAAAAAAAvBQAAZHJzL2Rvd25yZXYueG1sUEsFBgAAAAAEAAQA9QA
AAHgGAAAAAAAAEPAIAAAApA/wCRAUihAPABHwEAAAAAAAwwsIAAAAAwAAAAkCAAAPAA3wQgAAAAAAn
w8EAAAABAAAAAAAoQ8aAAAAAQAAAAAAAAgAAAEAAQAAAAAABgAMAImJif4AAKYPDAAAAPAAAADUAdA
C8AMQBQ8ABPALCQAAEgAK8AgAAAAGBAAAAAoAAJMAC/
BsAAAAfwABAO8BgAAA8PMBhwABAAAAAvwAAAAYAvwEBABEA/wEBAAkAPwMAAAgAgMM2AAAAAvwMAAAIA
UwBsAGkAZABlACAATgB1AG0AYgBlAHIAIABQAGwAYQBjAGUAaABvAGwAZABlAHIAIAA1AAAAEwAi8f
cHAACpw/EHAABQSwMEFAAGAAgAAAAhAPD3irv9AAAA4gEAABMAAABbQ29udGVudF9UeXBlc10ueG1s
lJHNSsQwEMfvgu8Q5iptqgcRaboHq0cVXR9gSKZt2DYJmVh33395OPy7iCh5n5v/xI6lX22kUM0W23i
m4LisQ5LQ31vUKPtZPxR0ITugMjt6Rgh0xrJrLi3q9C8Qiux0rGFIK91KyHmhCLn0gly+djxOmPMZe
BtQb7EneVNWt1N4lcqlISwY0dUsdfo5JPG7z+kASaWQQDwfh0qUAQxitxpRJ5ezMj5bi2FBm517Dgw
18lTFA/tqwXM4XHH0v+WmiNSReMaZnnDKGNJElDxgoa8q/UxbMiQvfdVZT2UZ+X3wnqHPhxn+5SPN/
s9tse6P5lC73P9R8AwAA//8DAFBLAwQUAAYACAAAACEAMd1fYdIAAACPAQAACwAAAF9yZWxzLy5yZW
xzpJDBasMwDIbvg72D0b1x2kMZo05vhV5LB7sKW0lMY8tYJm3fvqYwWEZvO+oX+j7x7/
a3MKmZsniOBtZNC4qiZefjYODrfFh9gJKC0eHEkQzcSWDfvb/tTjRhqUcy+iSqUqIYGEtJn1qLHSmg
NJwo1k3POWCpYx50QnvBgfSmbbc6/2ZAt2CqozOQj24D6nxP1fyHHbzNLNyXxnLQ3PfevqJqx9d4or
lSMA9UDLgszzDT3NTnQL/2rv/plRETfVf+QvxMq/XHrBc1dg8AAAD//wMAUEsDBBQABgAIAAAAIQCy
xTf9hgMAAJUKAAAQAAAAZHJzL3NoYXBleG1sLnhtbOxV227jNhCG7wu8g6B7RZIlWbIRZ2HLdrBA
CI0dF70OKGpsEZFIlaQcZ4u+e4eSfEmbRYvuVWwDIjnDGX7zz5DS7ZdjXTkHkIoJPnPDG991gFNR
Mr6buV+3a2/iOkoTXpJKcJi5r6DcL3effrptpnsiNTwKqQ1GEtJn1qLHSmg
```

vdcQuOuURpQ7QsO1ameSvYxTXLdFUReJxPk9F0hBSeQN/VBM/IY2CZODrCYTCepVdgnhIOi6vpJBkH
DocRibdsjXUfhX4zKuIPyqkR3CHDLGOHtXWezYuLQG1kos3ccanLk7cs8EUJxNL6+hSipuF/p80zJf
IXZXP6PklThO7CIp3MxliY15riE42TKy1zCiOmOM7JKXcm0imt27qTFG8F6cOVB5mgGgiTT2hAE8gq
RfWAX76kEp8ub2c12uCuljIsfJ+JlTTkwIDRHZNg42rl4p/ZZIR9ke/QlN44sP/qHOQiegqKHkiU+w
C9r3OXf3UqwiEIp2FmHfiE8BAEuAzfWpUYDZHrngYCZI+s2IKCkDefKxetBVurpdn79iSVVu4mbGGd
08g05ovq1diyY+oJzb3pFMfxkSSpti0PJLZ8w3u+EtD1Qthri6WozrbORm5feG35RXtTuX+x67VFhy
w8HkNLFt32+UW8Qxgvi3sM2mDiWBGb5pynvn/8BGJZJcswJ/+6ubudL6+m88HqfTodTGfpajBfztLB
LFneLm9Gt+Pxbfo36r4fVximCgPLH2GQlX3X1I3+vY4JAV85FWrw60OcaKH8SBGzFN5dThUA+lvB1H
sMQKW3QaJkL4y/
Q+KAYZifvWHLw07lbwNZP4ufw7JgVsja3ylIlNIbo3UVZNu4lRQMR8XKl8pjVdoXf2Qg/
nlVyWiPb9IRmJRVhZF1pr1bqz4tnffey6HIAp8VLqOc/tSogXT9pGWfKQSLCm4/
U3Db9yey4AP0Q+D/FvmmLeIWxF846Ey80S+eZqHKDTPMz9fvrvQ9vv96tV/4D1lp8b7c/hBtu/
gHAAD//wMAUEsDBBQABgAIAAAAIQAxKWUA3AAAAAMBAAAPAAAAZHJzL2Rvd25yZXYueG1sVI/BasJA
EIbvBd9hGaG3urEU0egq0iIthWKNxfQ4ZsckbXY27K4xvn2XXkovAz8z8/18i1VvGtGR87VlBeNRAo
K4sLrmUsHHfnM3BeEDssbGMim4kofVcnCzwFTbC++oy0IpIoR9igqqENpUSl9UZNCPbEscdyfrDIYY
XSm1w0uEm0beJ8lEGqw5NlTY0mNFxXd2NgryfPtF9vWcMx/eHrpNpsefzUyp22H/NI9jPQcRqA9/
H3v3fpi5/xe/0BetYALi9Hw9ulrv0AdyCqJldI6+IJc/
AAAA//8DAFBLAQItABQABgAIAAAAIQDw94q7/QAAAOIBAAATAAAAAAAAAAAAAAAAAAAABbQ29udG
VudF9UeXBlc10ueG1sUEsBAi0AFAAGAAgAAAAhADHdX2HSAAAAjwEAAAsAAAAAAAAAAAAAAAAALgEA
AF9yZWxzLy5yZWxzUEsBAi0AFAAGAAgAAAAhALLFN/2GAwAAlQoAABAAAAAAAAAAAAAAAAAAKQIAAG
Rycy9zaGFwZXhtbC54bWxQSwECLQAUAAYACAAAACEAMSllANwAAAADAQAADwAAAAAAAAAAAAAAADd
BQAAZHJzL2Rvd25yZXYueG1sUEsFBgAAAAAEAAQA9QAAAOYGAAAAAAAEPAIAAAApA8wFfAbihAPAB
HwEAAAAAAwwsIAAAABAAAAAgCAAAPAA3wWAAAAAAAnw8EAAAABAAAAAAAoA8CAAAAKgAAAKEPGgAA
AAIAAAAAAAAIAAACAAIAAAAAAAYADACJiYn+AADYDwQAAAAAAAAAACmDwwAAADwAAAA1AHQAvADEA
UPAATwcAUAABIACvAIAAAAAQQAAAAMAABjAAvwJAAAAIEBAAAACIMBBQAACL8BEAAQAP8BAAAIAAQD
CQAAAD8DAQABABMAIvEsBQAAqcMmBQAAUEsDBBQABgAIAAAAIQDw94q7/QAAAOIBAAATAAAAW0Nvbn
RlbnRfVHlwZXNdLnhtbJSRzUrEMBDH74LvEOYqbaoHEWm6B6tHV0fYEimbdg2CZlYd9/edD8u4goe
Z+b/8SOpV9tpFDNFtt4puC4rEOS0N9b1Cj7WT8UdCE7oDI7ekYIdMayay4t6vQvEIrsdKxhSCvdSsh
5oQi59IJcvnY8TpjzGXgbUG+xJ3lTVrdTeJXKpSEsGNHVLHX6OSTxu8/
pAEmlkEA8H4dKlAEMYrcaUSeXszI+W4thQZudew4MNfJUxQP7asFzOFxx9L/
lpojUkXjGmZ5wyhjSRJQ8YKGvKv1MWzIkL33VWU9lGfl98J6hz4cZ/uUjzf7PbbHuj+ZQu9z/
UfAMAAP//AwBQSwMEFAAGAAgAAAAhADHdX2HSAAAAjwEAAAsAAABfcmVscy8ucmVsc6SQwWrDMAyG7
4O9g9G9cdpDGaNOb4VeSwe7CltJTGPLWCZt376mMFhGbzvqF/o+8e/2tzCpmbJ4jgbWTQuKomXn42D
g63xYfYCSgtHhxJEM3Elg372/7U40YalHMvokqlKiGBhLSZ9aix0poDScKNZnzzlgqWMedEJ7wYH0p
m23Ov9mQLdgqqMzkI9uA+p8T9X8hx28zSzcl8Zy0Nz33r6iasfXeKK5UjAPVAy4LM8w09zU50C/9q7
/6ZURE31X/kL8TKv1x6wXNXYPAAAA//8DAFBLAwQUAAYACAAAACEAa0u0o74AAAARAQAAEAAAAGRyc
y9zaGFwZXhtbC54bWyMj01uAjEMRvdIvUfIpCeIJs0AUmDf6eG66xKRTij0KU2ANzVjJHbuyMibKxAVLFXonv5RaSY8TQ39F2m/
Se/66zeydOFPKU2ANzVKBIB6Dmdhq+P3Zfa5B5IJs0AUmDf+UYdN/
LLrYDjgebQonNqJCOLdRw6GU2EqZxwN5zMsQiWu2D8ljqWuyMibKxAVLFXonv5RaSY8TQ39F2m/
ai8nM9RWlmnrD9sairUsPC75jMQn/aoUXgTij0zDYBmTfyYfsPj2b9BcAAAD//wMAUEsDBBQABgAIA
AAAIQDB3HzC2QAAAAQBAAAPAAAAZHJzL2Rvd25yZXYueG1sTI9BS8NAEIXvgv9hGGGb3ShYJM2mFIs
IHhQbQY/T7LiJZmdDdpvG/nqnXurlMcNj3nyvWE6+UyMNsQ3nyvWE6+UyMNsQ1s4HqWgSKug23ZGXirHq7uQMWEbLELT
AZ+KMKyPD8rMLdhz680bpJTEsIxRwNNSn2udawb8hhnoScW7zMMHpOsg9N2wL2E+07fZNlce2xZPjT
Y031D9fdm5w08r+fjwd8eXPXx9bLSU/WePblHYy4vpvVCZLUAlWhKpws4dhB+KAVsG3Zso+oMSJv0p

+LJvD2qLgv9H778BQAA//8DAFBLAQItABQABgAIAAAAIQDw94q7/QAAAOIBAAATAAAAAAAAAAAAAAA
AAAAAAABbQ29udGVudF9UeXBlc10ueG1sUEsBAi0AFAAGAAgAAAAhADHdX2HSAAAAjwEAAAsAAAAAA
AAAAAAAAAALgEAAF9yZWxzLy5yZWxzUEsBAi0AFAAGAAgAAAAhAGtLtKO+AAAAEQEAABAAAAAAAAAA
AAAAAAAAKQIAAGRycy9zaGFwZXhtbC54bWxQSwECLQAUAAYACAAAACEAwdx8wtkAAAAEAQAADwAAA
AAAAAAAAAAAAAAVAwAAZHJzL2Rvd25yZXYueG1sUEsFBgAAAAAEAAQA9QAAABsEAAAAABAA8AcgAAA
A////AAAAAADn5uYARFRqAERyxADtfTEABWPBAJVPcgAPAIgTOAAAAA8AihMwAAAAAAC6DxAAAABfA
F8AXwBQAFAAVAAxADAAAACLExAAAAAAAOsuCAAAAAnG2QEwfNZFAAAOBAUNAABQSwMEFAAGAAgAAAA
hAOneD7//AAAAHAIAABMAAABbQ29udGVudF9UeXBlc10ueG1srJHLTsMwEEX3SPyD5S1KnLJACCXpg
seOx6J8wMiZJBbJ2LKnVfv3TNJUQqggFmws2TP3njvjcr0fB7XDmJynSq/yQisk6xtHXaXfN0/ZrVa
JgRoYPGGlD5j0ur68KDeHgEmJmlKle+ZwZ0yyPY6Qch+QpNL6OALLNXYmgP2ADs11UdwY64mROOPJQ
9flA7awHVg97uX5mCTikLS6PzZOrEpDCIOzwJLU7Kj5RskWQi7KuSf1LqQriaHNWcJU+Rmw6F5lNdE
1qN4g8guMEsOwDIlfz2cgGS3mvzueiezb1llsvN2Oso58Nl7MTsH/FGD1P+gT08x/W38CAAD//
wMAUEsDBBQABgAIAAAAIQCl1qfnwAAAADYBAAALAAAAX3JlbHMvLnJlbHOEj89qwzAMh+
+FvYPRfVHSwxgldi+lkEMvo30A4Sh/aCIb2xvr20/HBgq7CISk7/epPf6ui/nhlOcgFpqqBsPiQz/
LaOF2Pb9/gsmFpKclCFt4cIaje9u1X7xQ0aM8zTEbpUi2MJUSD4jZT7xSrkJk0ckQ0kpF2zRiJH+nk
XFf1x+YnhngNkzT9RZS1zdgro+oyf+zwzDMnk/
Bf68s5UUEbjeUTGnkYqGoL+NTvZCoZarUHtC1uPnW/QEAAP//AwBQSwMEFAAGAAgAAAAhAGt5lhaDA
AAigAAABwAAAB0aGVtZS90aGVtZS90aGVtZU1hbmFnZXIueG1sDMxNCsMgEEDhfaF3kNk3Y7soRWK
yy6679gBDnBpBx6DSn9vX5eODN87fFNWbSw1ZLJwHDYplzS6It/B8LKcbqNpIHMUsbOHHFebpeBjJt
I0T30nIc1F9I9WQha213SDWtSvVIe8s3V65JGo9i0dX6NP3KeJF6ysmCgI4/QEAAP//AwBQSwMEFAA
GAAgAAAAhAHtDvF2NBwAAzyAAABYAAAB0aGVtZS90aGVtZS90aGVtZTEueG1s7FlfixvJEX8P5DsM8
y7r34z+LJYPaSR5z961jSU73GOv1Jppb8+06G7tWhyG4HvKSyBwd+QlkLc8HMcd3EGOvOTDGGySy4d
Idc9o1C217N3FBBN2BctM61fVv66qrip13/3sZUq9C8wFYVnPr9+p+R7OZmxOsrjnP5uOKx3fExJlc
0RZhnv+Ggv/s3u//c1ddCQTnGIP5DNxhHp+IuXyqFoVMxhG4g5b4gy+WzCeIgmvPK7OOboEvSmtNmq
1VjVFJPO9DKWg9vFiQWbYmyqV/
r2N8hGF10wKNTCjfKJUY0tCY+fndYUQaxFR7l0g2vNhnjm7nOKX0vcoEhK+6Pk1/
edX792toqNCiMoDsobcWP8VcoXA/Lyh5+TxWTlpEIRBq1/
q1wAq93Gj9qg1apX6NADNZrDSnIuts92IggJrgPJHh+5he9isW3hDf3OPcz9UHwuvQbn+YA8/HkdgR
QuvQTk+3MOHg+5gaOvXoBzf2sO3a/1h0Lb0a1BCSXa+h66FrWa0WW0JWTB67IR3w2DcbhTKtyiIhjK
61BQLlslDsZaiF4yPAaCAFEmSeXK9xAs0gyiOECVnnHgnJE4g8JYoYwKGa43auNaE/
+oT6CftUXSEkSGteAETsTek+HhixslS9vwHoNU3IG9/+eXN65/fvP77m6+
+evP6h2JurcqSO0ZZbMr9+rc//ecvv/f+/dNff/36m3zqXbww8e++/8O7f/
zzfephxVtTvP32x3c///j2z3/813dfO7T3OToz4VOSYuE9wpfeU5bCAh388Rm/nsQ0QcSU6GexQBlS
szj0j2RioR+tEUUO3ADbdnzOIdW4gPdXLyzCk4SvJHFofJikFvCUMTpg3GmFh2ouw8zTVRa7J+crE/
cUoQvX3BHKLC+PVkvIscSlMkqwRfMJRZlEMc6w9NR37Bxjx+q+IMSy6ymZcSbYQnpfEG+AiNMkU3Jm
RdNW6Jik4Je1iyD427LN6XNvwKhr1UN8YSNhbyDqID/F1DLjfbSSKHWpnKKUmgY/
QTJxkZys+czEjYQET8eYMm80x0K4ZB5zWK/
h9IeQZtxuP6Xr1EZySc5dOk8QYyZyyM6jBKVLF3ZCssTEfi7OIUSR94RJF/yU2TtEvYMfUHbQ3c8Jt
tz94WzwDDKsSWkbIOqbFXf48j5mVvxO1nSBsCvV9Hlqpdg+J87oGKxiK7RPMKboEs0x9p597mAwYEv
L5lvSDxLIKsfYFVgPkB2r6j3DAnol1dzs58kTIqyQneCYHeBzut5JPGuUpYgf0vwIvG7afASlLnUFw
GM6OzeBjwj0gBAvTqM8FqDDCO6DWp8kyCpg6l2443XNLf9dZY/Bvnxh0bjCvgQZfG0ZSOymzHttM0X
UmmAbMFMEXYYr3YKI5f6tiCquWmzllFvYm3brBuiOrKYnJdkHO6Cd3if83/
Q+jt3wcboet2IrZV2z3zmUUo53upxDuN3eJmJ8Tj791maIVtkTDNVkP2/ddja3nY3/f9/
ZHNrPt/3Moa7jtp/xoc+47WeKI5aP089sWxjobtSxR37cow9/0oNnPwtC6USuKT4R+vhHwK+a+RgGl

```
Zw+98TlWeAygUdV5mACCxdzpGU8zuTviEwmCVrCGVHdV0piUaiOhbdkAo6O9LBTt8LTVXrK5vmRZ72
ujjfzyiqQ3I7XwnIcjqtkjm61t8d4pXrNNtbHrRsCSvY6JIzJbBJNB4n2ZlAZSR/
ugtEcJPTKPgqLroNFR6nfuGqPBVArvQI/uz34sd7zwwBEQAhO5aBFnys/5a7eeFc782N6+pAxrQiAN
nsTAVtPdxXXg8tTq8tD7QqetkgY4WaT0JbRDZ5I4MdwEZ1q9Co0ruvr7talFj1lCj0fhNaWRrvzPhY
39TXI7eYGmpmZgmbeZc9vNUMImRla9vwFHB3DY7qE2BHqlxeiMdy/zCTPN/xNMsuSCzlEIskNrpNOn
g1SIjH3KEl7vlp+6Qaa6RyiudUbkBA+WXJdSCufGjlwuu1kvFjgmTTdbowoS+evkOHzXOH8VovfHKw
k2QrcPUnml94ZXfGnCEIsbNeVAedEwA1CPbfmnMCVWJnItvG3U5iKtGveSekYyscRXSaoqChmMs/hO
pWXdPRbaQPjrVgzGNQwSVEIz2JVYE2jWtW0rBo5h4NV98NCynJG0tzWTCurqKrpzmLWDJsysGPLmxV
5g9XGxJDTzAqfp+7dlNvd5LqdPqGsEmDw0n6OqnuFgmBQ205mUVOM99OwytnFqF07Ngv8ALWrFAkj6
7c2anfsVtYI53QweKPKD3K7UQtDi01fqS2t787N62129gKSxxC63BWVQrsSznc5goZoonuSPG3AFnk
pi60BT96Kk57/ZS3sB1EjjCq1TjiqBM2gVumE/
WalH4bN+iis14aDxisoLDJJ62F+bz+Gawy6Lm7v9fjeDX66uam5M2Nplekb+qomrm/w643DN/
gegaTzZasx7ja7g1al2+yPK8Fw0Kl0o9agMmxF7eF4GIWd7viV711ocNBvRkFr1Km06lFUCVo1Rb/
TrbSDRqMftPudUdB/VbQxsPI8fRS2APNqXvf+CwAA//8DAFBLAwQUAAYACAAAACEADdGQn7YAAAAbA
QAAJwAAAHRoZW1lL3RoZW1lL19yZWxzL3RoZW1lTWFuYWdlci54bWwucmVsc4SPTQrCMBSE94J3CG9
v07oQkSbdiNCt1AOE5DUNNj8kUeztDa4sCC6HYb6ZabuXnckTYzLeMWiqGgg66ZVxmsFtuOyOQFIWT
onZO2SwYIKObzftFWeRSyhNJiRSKC4xmHIOJ0qTnNCKVPmArjijj1bkIqOmQci70Ej3dX2g8ZsBfMU
kvWIQe9UAGZZQmv+z/TgaiWcvHxZd/lFBc9mFBSiixszgI5uqTATKW7q6xN8AAAD//
wMAUEsBAi0AFAAGAAgAAAAhAOneD7//AAAAHAIAABMAAAAAAAAAAAAAAAAAAAAAAFtDb250ZW50X1R
5cGVzXS54bWxQSwECLQAUAAYACAAAACEApdan58AAAAA2AQAACwAAAAAAAAAAAAAAAAAwAQAAX3Jlb
HMvLnJlbHNQSwECLQAUAAYACAAAACEAa3mWFoMAAACKAAAAHAAAAAAAAAAAAAAAAAAZAgAAdGhlbWU
vdGhlbWUvdGhlbWVNYW5hZ2VyLnhtbFBLAQItABQABgAIAAAAIQB7Q7xdjQcAAM8gAAAWAAAAAAAAA
AAAAAAAAANYCAAB0aGVtZS90aGVtZS90aGVtZTEueG1sUEsBAi0AFAAGAAgAAAAhAA3RkJ+2AAAAGwE
AACcAAAAAAAAAAAAAAAAAlwoAAHRoZW1lL3RoZW1lL19yZWxzL3RoZW1lTWFuYWdlci54bWwucmVsc
1BLBQYAAAAABQAFAF0BAACSCwAAAAAAA8EOgEAADw/eG1sIHZlcnNpb249IjEuMCIgZW5jb2Rpbmc
9IlVURi04IiBzdGFuZGFsb25lPSJ5ZXMiPz4NCjxhOmNsck1hcCB4bWxuczphPSJodHRwOi8vc2NoZ
W1hcy5vcGVueG1sZm9ybWF0cy5vcmcvZHJhd2luZ21sLzIwMDYvbWFpbiIgYmcxPSJsdDEiIHR4MT0
iZGsxIiBiZzI9Imx0MiIgdHgyPSJkazIiIGFjY2VudDE9ImFjY2VudDEiIGFjY2VudDI9ImFjY2Vud
DIiIGFjY2VudDM9ImFjY2VudDMiIGFjY2VudDQ9ImFjY2VudDQiIGFjY2VudDU9ImFjY2VudDUiIGF
jY2VudDY9ImFjY2VudDYiIGhsaW5rPSJobGluayIgZm9sSGxpbms9ImZvbEhsaW5rIi8+sAAeBNkFA
ABQSwMEFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMAAABbQ29udGVudF9UeXBlc10ueG1sfJDLTsQwDEX
3SPxDlC1q0mGBEGo7Cx4rBCyGD7ASt43IS3E6mv49aWeQAA2sLD+uz7Wb7cFZtsdEJviWb0TNGXoVt
PFDy993T9UtZ5TBa7DBY8tnJL7tLi+a3RyRWFF7avmYc7yTktSIDkiEiL50+pAc5JKmQUZQHzCgvK7
rG6mCz+hzlZcdvGsesIfJZvZ4KOWjk4SWOLs/Di6slkOM1ijIxance/2LUp0IoijXGRpNpKtig8uzh
KXzN+Ckey2vSUYje4OUX8AVG1InkmRL8RnmMOUfyUb8v/aM79D3RqEOanLlJyImpBLXE5wV30Bft8j
19d0nAAAA//8DAFBLAwQUAAYACAAAACEAcPA43L4AAAA4AQAACwAAAF9yZWxzLy5yZWxzhI/
BCsIwEETvgv8Q9m7TehCRpr2IIHgS/tZAl2bbBNgnZKPbvzdGC4HEY5s1M3b6nUbwosvVOQVWUIMhpb
6zrFdxvp80eBCd0BkfvSMFMDG2zXtVXGjHlEA82sMgUxwqGlMJBStYDTciFD+Sy0/
k4Ycoy9jKgfmBPcluWOxm/GdAsmOJsFMSzqUDc5pCb/7N911lNR6+fE7n0o0LyaA1dcPbPlLEYe0oK
TORvYyGqIu8H2dRy8bf5AAAA//8DAFBLAwQUAAYACAAAACEAuYX+U6YCAABoBwAAIQAAAGRycy9zbG
lkZUxheW91dHMvc2xpZGVMYXlvdXQxLnhtbLRVTW8bIRC9V+p/QNxrr9dOHMvKRqqiRj20UqQ0
QNyyMLtLwwIF7Mb/vrPrr7RRD+0FMcPMm8ebGbi+7RpNWvBBWZPTfDSmBIy0QpkqpyvPHy7mlIQo
jBDaGsjpFgK6vbwcsfBcMBSE94J3CG9
v07oQkSbdiNCt1AOE5DUNNj8kUeztDa4sCC6HYb6ZabuXnckTYzLeMWiqGgg66ZVxmsFtuOyOQFIWT
onZO2SwYIKObzftFWeRSyhNJiRSKC4xmHIOJ0qTnNCKVPmArjijj1bkIqOmQci70Ej3dX2g8ZsBfMU
kvWIQe9UAGZZQmv+z/TgaiWcvHxZd/lFBc9mFBSiixszgI5uqTATKW7q6xN8AAAD//
```

```
5Oya01VAvEuAV3gn2W9WKFUbA3HewM8RQooHNRI0laWPgJppwSgYI9AsbQgq1cMLN6YRjzDrL7bvRc
z0zwvulmBvHao/UoOOs3erPwK8EMPrId94eERKrV0rTTMamAHbSaYBBx7Z/
gRCpIAtG4SF9WaXO7x5Y23/ZYZykAZLAJCvrrWNHrcvJUzg4pw0150YcAxpWijxZJBQV7HmKd9KZLq
L54H0c3KGrivB4YKcNBuShR7xVNA03J2waqU/4bgsqzfPRpEGkqTk+G+ck2V/lpXoZ9z9hJORyWRRm
CJCQIEqF15VZfVL32TP+CNwjqm2aCGfHFR1hh3dytBQt6AGukgpLgAcaC+EFj8uh+Hm3d9EJw+oicQ
qzmDl0T65hBoWqYREAZgx4O2qFHYbKeEUPu/
gT7cddnC8Eg1ZRiyNqT+XfpitfS+QaaCUJZo0QNqeQeG3o/afRPKnqudkSESYA2TS3wDjGLEs6S0PL
7tDw7K0aF3/9fWkKLIdGJzcC9X1vPcJDWbmkLCodmiY8UJRD0dgfNGVVw/gjWMXEAYhD0bcRFw83hg
EVs5beIuFRPxjUHpzg6AJEv9wLCCfjuCQmDEs9s+PSnfOh4Ya6Jvu1CXXC/wVxehCUNN1p/
cL2YeIx0Q06fAQAA//8DAFBLAQItABQABgAIAAAAIQBNjvP8/QAAALsBAAATAAAAAAAAAAAAAAAAAAA
AAAABbQ29udGVudF9UeXBlc10ueG1sUEsBAi0AFAAGAAgAAAAhAHDwONy+AAAAOAEAAAsAAAAAAAAA
AAAAAAAALgEAAF9yZWxzLy5yZWxzUEsBAi0AFAAGAAgAAAAhALmF/lOmAgAAaAcAACEAAAAAAAAAAA
AAAAAAFQIAAGRycy9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQxLnhtbFBLBQYAAAAAAwADAMkAAAD6
BAAAAACgAB4EnAUAAFBLAwQUAAYACAAAACEATY7z/
P0AAAC7AQAAEwAAAFtDb250ZW50X1R5cGVzXS54bWx8kMtOxDAMRfdI/
EOULWrSYYEQajsLHisELIYPsBK3jchLcTqa/j1pZ5AADawsP67PtZvtwVm2x0Qm+JZvRM0ZehW08UP
L33dP1S1nlMFrsMFjy2ckvu0uL5rdHJFYUXtq+ZhzvJOS1IgOSISIvnT6kBzkkqZBRlAfMKC8rusbq
YLP6HOVlx28ax6wh8lm9ngo5aOThJY4uz8OLqyWQ4zWKMjFqdx7/
YtSnQiiKNcZGk2kq2KDy7OEpfM34KR7La9JRiN7g5RfwBUbUieSZEvxGeYw5R/
JRvy/9ozv0PdGoQ5qcuUnIiakEtcTnBXfQF+3yPX13ScAAAD//
wMAUEsDBBQABgAIAAAAIQBw8DjcvgAAADgBAAALAAAAX3JlbHMvLnJlbHOEj8EKwjAQRO+C/
xD2btN6EJGmvYggeBL9gCXZtsE2Cdko9u/N0YLgcRjmzUzdvqdRvCiy9U5BVZQgyGlvrOsV3G+nzR4
EJ3QGR+9IwUwMbbNe1VcaMeUQDzawyBTHCoaUwkFK1gNNyIUP5LLT+ThhyjL2MqB+YE9yW5Y7Gb8Z0
CyY4mwUxLOpQNzmkJv/s33XWU1Hr58TufSjQvJoDV1w9s+UsRh7SgpM5G9jIaoi7wfZ1HLxt/
kAAAD//wMAUEsDBBQABgAIAAAAIQAZzPVNaQIAAIgGAAAhAAAAZHJzL3NsaWRlTGF5b3V0cy9zbGlk
ZUxheW91dDEueG1snFVNbxshFLxX6n9A3JN1kqqqVrYjNW16aD6s2On9lcVZFBYQkK397zvArtM0ae
T0gll4b3gzb8DT002nWS99UNbM+NHhhDNphG2UuZvx29X5wSfOQiTTkLZGzvhWBn46f/9u6uqgmwva
2ofIgGFCTTPexujqqgqilR2FQ+ukwd7a+o4iPv1d1Xj6BexOV8eTyceqI2X4kO/3ybfrtRLyixUPnT
SxgHipKaL+0CoXRjS3D5rzMgAmZz8tKW4d2EKYuNpwluN8j5UjPgd1sdQNM9RhYaWilgwCsR8IVoI0
W8lNzGHBrbyUKcH037xbuoXP2Vf9wjPVJLQBhVfDxhCWPw3CMKn+Sr8bkajerH03n1INVdhmxtG8bR
qRRDWKYKIsisdV0V6/ECvary9EV+MBqGB3KPruCqPndI5HOkWUox2rEkpIvbDiPjBjwTPRL/TEVT+C
Jc4J3rWstCAmfYe4spn1GOMDNM1ixc1n22wT8Z/4zYtU6xCXcatlFgRlUw1wDJBfU3K4NAe3y4ROdZ
yfaSXuWbRMNiqySwpRepbPxxUAyhSCRPRjQJGmWZCnmz/Bvt/kUqnGYahzLArTotq/tTsZtXtiI7bQ
JGRrdYNSjhM2zDeq9V96JnU4s17B98XgHFaET8ZmvEXk9HIARVIqusj4XHJ0iOle74z59hYkK+cOhC
ctQCNyT8swnpJ5vN7opRQWF1bLXuo9ELPuryOuWuX3BzwpUr0mxLl98LHdu8QPeyCq9YuAeCnebOTs
5/K2YZpew2xM7S/JXfeZF95/XJ+zvOTw4sO6KfQxJGGM/yDz3wAAAP//
AwBQSwECLQAUAAYACAAAACEATY7z/P0AAAC7AQAAEwAAAAAAAAAAAAAAAAAAAAAAW0NvbnRlbnRfVH
lwZXNdLnhtbFBLAQItABQABgAIAAAAIQBw8DjcvgAAADgBAAALAAAAAAAAAAAAAAAAAAAAAAC4BAABfcmVs
cy8ucmVsc1BLAQItABQABgAIAAAAIQAZzPVNaQIAAIgGAAAhAAAAAAAAAAAAAAAAABUCAABkcnMvc2
xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MS54bWxQSwUGAAAAAAMAAwDJAAAAvQQAAAAkAAeBHoIAABQ
SwMEFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMAAABbQ29udGVudF9UeXBlc10ueG1sfJDLTsQwDEX3SP
xDlC1q0mGBEGo7Cx4rBCyGD7ASt43IS3E6mv49aWeQAA2sLD+uz7Wb7cFZtsdEJviWb0TNGXoVtPFD
y993T9UtZ5TBa7DBY8tnJL7tLi+a3RyRWFF7avmYc7yTktSIDkiEiL50+pAc5JKmQUZQHzCgvK7rG6
mCz+hzlZcdvGsesIfJZvZ4KOWjk4SWOLs/Di6slkOM1ijIxance/2LUp0IoijXGRpNpKtig8uzhKXz
```

N+Ckey2vSUYje4OUX8AVG1InkmRL8RnmMOUfyUb8v/aM79D3RqEOanLlJyImpBLXE5wV30Bft8j19d
0nAAAA//8DAFBLAwQUAAYACAAAACEAcPA43L4AAAA4AQAACwAAAF9yZWxzLy5yZWxzhI/
BCsIwEETvgv8Q9m7TehCRpr2IIHgS/YAl2bbBNgnZKPbvzdGC4HEY5s1M3b6nUbwosvVOQVWUIMhpb
6zrFdxvp80eBCd0BkfvSMFMDG2zXtVXGjHlEA82sMgUxwqGlMJBStYDTciFD+Sy0/
k4Ycoy9jKgfmBPcluWOxm/GdAsmOJsFMSzqUDc5pCb/7N911lNR6+fE7n0o0LyaA1dcPbPlLEYe0oK
TORvYyGqIu8H2dRy8bf5AAAA//8DAFBLAwQUAAYACAAAACEAHddHlkcFAABqEAAAIQAAAGRycy9zbG
lkZUxheW91dHMvc2xpZGVMYXlvdXQxLnhtbKxY7W7bNhT9P2PEKZJs2YalrVG7D0BL
VKSFpgSSdp08/c4lKUtOss1NgwAKRV4e3u9D+fzDfiPZTmhTVmoexe/7ERMqrbJS3c2jr6ubd9OIGc
tVxmWlxDx6ECb6cPHzT+f1zMjslj9UW8uAocyMz6PC2nrW65m0EBtu3le1UFjLK73hFq/6rpdp/g3Y
G9lL+v1xb8NLFYX9+pT9VZ6Xqfi1SrcboawH0UJyC/1NUdamQatPQau1MIBxu49Vsg81rK3LdLWPmB
PTO0zE0QUsT5cyY4pvMLEoU7vVgn0rbcGueU16OBlTr7QQJK12v+t6WS+02/
ppt9CszAgqQES9sBDE3KuCGAa9J9vvGiQ+2+d6c3HOZ/AI288jBO6BntjEZ2JvWeon03Y2LT6/IJsW
v70g3WsOgAaHQxHz2lv03JykMWdVWilYfLDKi3Jsva3Se8NUBTvJfG9e+mnXgJHNBF8XzLvfElSQ84
vOH428cT5tFD14Yjo4m0yRtjB8OJogzY59MjgbJMlgEjHyTDzu94NE12KPXM/s/qrKHsija/
xH4LhKiwqJuvaY0tilfZAIM5/JnYyDQpnIv0DYPM6jQYt+EMBJ3Y1wKZ/
BcDywSXKqPKHefV36M+zFtSzTe2YrJrLSso/cWKGZ8w1KE6efA9AiVwKKUNmCa04aHMD+
+uLcyGc4DD5s7MLQR/Tf4zpo4tpk+kLyVBSVzKBEQqiohyaGr4oyiixCRSBdm5w4PdajeDqIQ7DPpp
NhMvJOawpgHE8SioAL9nA6GYy9xEnBRlu06CKVfoyY/FOZeXQWD4cAs+7FJVfEdHdlfbRi5XUlqSgp
uApd8HJrq7y0lPlNSrnUeZ5GbMP1ravqUmVoUQ3KevsJfdgV+f9kmcvIBBkZoEItsFPwkqmvGmiJrC
UQp+ZODlo85ws44yS8YRePQALesMWLBxOqxhMB+11AQgmAow7gNJmSHadpeARIKAFw3AImyRQKvgqQ
UALgpAM4Gbr+8AoNCSUATltAQiNXvwKQUALgWQdwPJq8MiiE4vL8Wa9r4aEp8vyLy3Mk2pN8Z1ze4V
IiI4ZUX/H1Ev20yTptvbTgt+pK3zs6zStlL90WjjpDzYLXVVjGSQU6K64ei61KUU7E5MhutaxTGpg6
XaSW7ThgzxBkRxqU/h2BK5E3otZ40biVBEIrcJmjIXvURrRBhFxYXW+vpV7tXTGvt8vHw/AGhjgKzN
Fq59GlLjmcUHNVGbyCsK76Y8Q5OfwNsVratLjhm1KC1gaYSAuujXCNI5xw6ByWr28NdSE+800EQ9ga
MqDDH/fbTbmp/i59gBpmApm03MbWPmruuZ1HCu2Jroy6vIeuqlq6UcTuhaYLpm/
GHBeAIFinbid1Ry7LR/GHe11zI2RJF04ETlULXVU5jUllqeipqptSSq+4nzGVLDOadJ6nK6iAf32c7
N5zC0LSlRJ5LlLb+GJ7q0IItgQTxi6LOtH4ZaPeSRtohj9ZENwvpObJQmpoAU5uvet9jud38TEi62+
eK+K4LhkP6IQfJWO65/joFlzmaCTEy47moebpvNy5gyX90SS0//
ZienQJG0zjeAQS9w7y91p3M6EG0jqnYUyIIQl+lDId1/gTDxczwHq2ewVlxkcU9+OU6SrloN8bUGan
VcHKN6DMY8A3oMxjwDegzGPAN6DMY8A3oMxjwP+mzECQLv1RGN/5uUDdwn0tGOpCh88FavsvtSBXf/
4LFkP64HW9ReqPvP68c8fjCx8fImixmKpBrJSsEG1FCKP5jeDiHwAAAP//
AwBQSwECLQAUAAYACAAAACEATY7z/P0AAAC7AQAAEwAAAAAAAAAAAAAAAAAAAAAAW0NvbnRlbnRfVH
lwZXNdLnhtbFBLAQItABQABgAIAAAAIQBw8DjcvgAAADgBAAALAAAAAAAAAAAAAAAAAAC4BAABfcmVs
cy8ucmVsc1BLAQItABQABgAIAAAAIQAd10eWRwUAAGoQAAAhAAAAAAAAAAAAAAAAABUCAABkcnMvc2
xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MS54bWxQSwUGAAAAAAMAAwDJAAAAmwcAAAAgAAeBBUHAABQ
SwMEFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMAAABbQ29udGVudF9UeXBlc10ueG1sfJDLTsQwDEX3SP
xDlC1q0mGBEGo7Cx4rBCyGD7ASt43IS3E6mv49aWeQAA2sLD+uz7Wb7cFZtsdEJviWb0TNGXoVtPFD
y993T9UtZ5TBa7DBY8tnJL7tLi+a3RyRWFF7avmYc7yTktSIDkiEiL50+pAc5JKmQUZQHzCgvK7rG6
mCz+hzlZcdvGsesIfJZvZ4KOWjk4SWOLs/Di6slkOM1ijIxance/2LUp0IoijXGRpNpKtig8uzhKXz
N+Ckey2vSUYje4OUX8AVG1InkmRL8RnmMOUfyUb8v/aM79D3RqEOanLlJyImpBLXE5wV30Bft8j19d
0nAAAA//8DAFBLAwQUAAYACAAAACEAcPA43L4AAAA4AQAACwAAAF9yZWxzLy5yZWxzhI/
BCsIwEETvgv8Q9m7TehCRpr2IIHgS/YAl2bbBNgnZKPbvzdGC4HEY5s1M3b6nUbwosvVOQVWUIMhpb
6zrFdxvp80eBCd0BkfvSMFMDG2zXtVXGjHlEA82sMgUxwqGlMJBStYDTciFD+Sy0/
k4Ycoy9jKgfmBPcluWOxm/GdAsmOJsFMSzqUDc5pCb/7N911lNR6+fE7n0o0LyaA1dcPbPlLEYe0oK

TORvYyGqIu8H2dRy8bf5AAAA//8DAFBLAwQUAAYACAAAACEAiBC3oeIDAABnDQAAIQAAAGRycy9zbG
lkZUxheW91dHMvc2xpZGVMYXlvdXQxLnhtbMRX3ZLaNhi970zfQeP7BPwDNp6FzHTb9KKbzU4gD6C1
xdqNLHlkLYE+fY8kiwVCFi/Tmd6AEec70vd3Pvnmw7bhZMNUV0sxD8L344AwUciyFk/
z4Ovq47ssIJ2moqRcCjYPdqwLPix+/eWmzTte3tGdfNYEHKLL6TyotG7z0agrKtbQ7r1smcB/a6kaq
vFTPY1KRb+Du+GjaDyejhpai6C3V0Ps5XpdF+x3WTw3TGhHohinGufvqrrtPFs7hK1VrAONtT4+kt6
18FY+/r3aBsTC1AYLYbCA58WSl0TQBgu3UmgwkO+1rsgtbc05LKZrV4oxgxabP1W7bB+UNb3fPChSl
4aqpwhG/R89zP4UgOFhdGL+5Jlovl2rZnFDc0SEbOcBErcznzCiOdtqUrjF4mW1qD6fwRbVH2fQI78
BTrDfFDlvnUc/uhN5d1a15oyEe68clML0ThbfOiIk/DTuO/eK+40nMz4b+rYiLvzaUPU496eNh8d3N
qb+oPtIZPEszVC2cDyZpCiz45jEsziK4jQgJjLhdDzuEYceO+Y219vfZLkzEX3ENxJHRVFJFOqj4+S
dXuodR5ppzjc87A9UsvUXgLt/5kH8wr4HYKdDQ4SU5nAcHzDi1HQeE++
+Lt0eenHL6+Ib0ZKwstbkE+00U8TGBq2J3W9AqFErPQsT5QNV1JxgT/bXFxtGmmMzxND7hUeX0Z/nN
fZ59ZX+wGnBKslLHCIyrOgHn8M3ZrkuUaO+EIYneBJmcdhneJalSTRxkfJVPw3TyITdZjjJ0njqEEM
ybNvnMDv7rNH8QlptCUQ/lkCUuRI0ad9wD8BjfAabHGI9ANjkDHZ8iPUAYCeXsB4A7PQS1gOATS9hP
QDY7BLWA4CdXcI6gInfQb/ZtoElAcNe+N7eRqZobBd1R22EzdwGpi/3u9hafb1Zl6yQoiScbRgfwGg
76HXGVVWr4YSx64XXAvFRPitMq6FHTAYw1uuzhJhE/6kYJV6MViZth0pk3b5eidy8MSKP6w5ku6J8H
WBMQ59shuzcMdPpjQMoGk/SvqVfpvLRBIqzMJxAzMw09KPMyrKbhV6p/QT6mT6Rhqo7ewuoRYkLiXk
0pfv4fI97m7U6kC8z9vod0T1+bOHRqFNP1c9OMojvSLZOJK7nm4WJQQ3jO5qaJzLY84Vxat0YRviaV
nrCLMqMVF9BeCKoPWEUZYjzVYQnqusJ08TeJ6444Yk094SGbXhSjmJ4ot+ecDpJTW1dccL/
TeTRbb7LbOPZJvfXd6yY27695XD1ibafN1Za8XqDW9itXWrxQmPaF9AXiKHyL0iLfwEAAP//
AwBQSwECLQAUAAYACAAAACEATY7z/P0AAAC7AQAAEwAAAAAAAAAAAAAAAAAAAAAAW0NvbnRlbnRfVH
lwZXNdLnhtbFBLAQItABQABgAIAAAAIQBw8DjcvgAAADgBAAALAAAAAAAAAAAAAAAAAAAAAAC4BAABfcmVs
cy8ucmVsc1BLAQItABQABgAIAAAAIQCIELeh4gMAAGcNAAAhAAAAAAAAAAAAAAAAAAABUCAABkcnMvc2
xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MS54bWxQSwUGAAAAAAMAAwDJAAAANgYAAAAcAAeBHUEAABQ
SwMEFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMAAABbQ29udGVudF9UeXBlc10ueG1sfJDLTsQwDEX3SP
xDlC1q0mGBEGo7Cx4rBCyGD7ASt43IS3E6mv49aWeQAA2sLD+uz7Wb7cFZtsdEJviWb0TNGXoVtPFD
y993T9UtZ5TBa7DBY8tnJL7tLi+a3RyRWFF7avmYc7yTktSIDkiEiL50+pAc5JKmQUZQHzCgvK7rG6
mCz+hzlZcdvGsesIfJZvZ4KOWjk4SWOLs/Di6slkOM1ijIxance/2LUp0IoijXGRpNpKtig8uzhKXz
N+Ckey2vSUYje4OUX8AVG1InkmRL8RnmMOUfyUb8v/aM79D3RqEOanLlJyImpBLXE5wV30Bft8j19d
0nAAAA//8DAFBLAwQUAAYACAAAACEAcPA43L4AAAA4AQAACwAAAF9yZWxzLy5yZWxzhI/
BCsIwEETvgv8Q9m7TehCRpr2IIHgS/YAl2bbBNgnZKPbvzdGC4HEY5s1M3b6nUbwosvVOQVWUIMhpb
6zrFdxvp80eBCd0BkfvSMFMDG2zXtVXGjHlEA82sMgUxwqGlMJBStYDTciFD+Sy0/
k4Ycoy9jKgfmBPcluWOxm/GdAsmOJsFMSzqUDc5pCb/7N911lNR6+fE7n0o0LyaA1dcPbPlLEYe0oK
TORvYyGqIu8H2dRy8bf5AAAA//8DAFBLAwQUAAYACAAAACEA/
+70YUIBAABwAgAAIQAAAGRycy9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQxLnhtbIxSy07DMBC8I/
EPlu/ULQeEMBC8I/
tHZD8vds3AQE6qEXaz2eGe94vVx1RrNWYWiczfliNudMWenKxu5z/r57ufurrlESwJWhnVc57FfiquL
xY+izloSwJWhnVc57FfiquL
mNMNBYPurxHL2rqkaqBycPRtl4NEGlIVL/oW58mNz8OW4eVSCbpP7bUuw9pf3QYD85SzRsCVjwgpLL
rS6ZBUPAXWIMYPA7VGqobPuMfus3mLhv7QZZUw7aUcPFeDDS0tYSjQrxT76fnCDrKjTFEjJ6AtblnC
bVDyuJIFNdZPIIyl9U1usTXFk/
nmCL6QLq4OdSqqdYVA6xU+caX8GvW8oHGc05KrxPkKfJHjPIX8rgMf2U4hsAAP//
AwBQSwECLQAUAAYACAAAACEATY7z/P0AAAC7AQAAEwAAAAAAAAAAAAAAAAAAAAAAW0NvbnRlbnRfVH
lwZXNdLnhtbFBLAQItABQABgAIAAAAIQBw8DjcvgAAADgBAAALAAAAAAAAAAAAAAAAAAAAAAC4BAABfcmVs
cy8ucmVsc1BLAQItABQABgAIAAAAIQD/7vRhQgEAAHACAAAhAAAAAAAAAAAAAAAAAAABUCAABkcnMvc2

xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MS54bWxQSwUGAAAAAAMAAwDJAAAAlgMAAAAAYAAeBAoFAABQ
SwMEFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMAAABbQ29udGVudF9UeXBlc10ueG1sfJDLTsQwDEX3SP
xDlC1q0mGBEGo7Cx4rBCyGD7ASt43IS3E6mv49aWeQAA2sLD+uz7Wb7cFZtsdEJviWb0TNGXoVtPFD
y993T9UtZ5TBa7DBY8tnJL7tLi+a3RyRWFF7avmYc7yTktSIDkiEiL50+pAc5JKmQUZQHzCgvK7rG6
mCz+hzlZcdvGsesIfJZvZ4KOWjk4SWOLs/Di6slkOM1ijIxance/2LUp0IoijXGRpNpKtig8uzhKXz
N+Ckey2vSUYje4OUX8AVG1InkmRL8RnmMOUfyUb8v/aM79D3RqEOanLlJyImpBLXE5wV30Bft8j19d
0nAAAA//8DAFBLAwQUAAYACAAAACEAcPA43L4AAAA4AQAACwAAAF9yZWxzLy5yZWxzhI/
BCsIwEETvgv8Q9m7TehCRpr2IIHgS/YAl2bbBNgnZKPbvzdGC4HEY5s1M3b6nUbwosvVOQVWUIMhpb
6zrFdxvp80eBCd0BkfvSMFMDG2zXtVXGjHlEA82sMgUxwqGlMJBStYDTciFD+Sy0/
k4Ycoy9jKgfmBPcluWOxm/GdAsmOJsFMSzqUDc5pCb/7N911lNR6+fE7n0o0LyaA1dcPbPlLEYe0oK
TORvYyGqIu8H2dRy8bf5AAAA//8DAFBLAwQUAAYACAAAACEA32GgxNcBAAC1AwAAIQAAAGRycy9zbG
lkZUxheW91dHMvc2xpZGVMYXlvdXQxLnhtbIxTTU/cMBC9V+p/
sHyH7V+p/
xmzfznu3N5dAZ0WOIrbOlXJ+vpECrXd3aUykfD9/PvkoRCWwNxlks5YhRXlafP228iqa+hdE9k2AOG
xWUsiHyqiiibrCDeO48Wj47utAB8TacijrAb+buTHGxWn0pOmitnPPDR/
Ld8dhqvHH6uUNLE0lAA8T9x6b1cWHzH2HzASPT5Oy/
W6LRs1pqyeDWmlGKDA09B9eyYvV6b2phoePAIaFEhqWT6A8BMa1s/
yP4vd+FnHDf74Jo60QwJ8piPphheWsZxovin/
TTwgRqOIau2oBiL8RQSr6yMY2cBAoHEnoK6teobrbvYHXz7R10sRTgDl6KJlWTov/lXCxyJh/WL6om
KHDqrdNPUVjHOpP8SZ6+7xeypDnR+0a8MX7GTYfZjwUf2dNsFg1Xrh6T8F885yAoE2lPo8FsCLcNis
l5YPsNpHeN9uxxn9hBUXVtWv0kyAmsWxJ3EAmDyBfPD59ZNmwI8X3MLGjrHQR4eEv28yG3CoqLcZ9L
U7xMruVpehK8TO8m37oJd+C3fW6NPwtXvc4hz99jNugVkjiW71b9AQAA//8DAFBLAQItABQABgAIAA
AAIQBNjvP8/QAAALsBAAATAAAAAAAAAAAAAAAAAAAAAABbQ29udGVudF9UeXBlc10ueG1sUEsBAi0A
FAAGAAgAAAAhAHDwONy+AAAAOAEAAAsAAAAAAAAAAAAAAAAALgEAAF9yZWxzLy5yZWxzUEsBAi0AFA
AGAAgAAAAhAN9hoMTXAQAAtQMAACEAAAAAAAAAAAAAAAAFQIAAGRycy9zbGlkZUxheW91dHMvc2xp
ZGVMYXlvdXQxLnhtbFBLBQYAAAAAAwADAMkAAADrBAAAAABQAB4ETAcAAFBLAwQUAAYACAAAACEATY
7z/P0AAAC7AQAAEwAAAFtDb250ZW50X1R5cGVzXS54bWx8kMtOxDAMRfdI/
EOULWrSYYEQajsLHisELIYPsBK3jchLcTqa/j1pZ5AADawsP67PtZvtwVm2x0Qm+JZvRM0ZehW08UP
L33dP1S1nlMFrsMFjy2ckvu0uL5rdHJFYUXtq+ZhzvJOS1IgOSISIvnT6kBzkkqZBRlAfMKC8rusbq
YLP6HOVlx28ax6wh8lm9ngo5aOThJY4uz8OLqyWQ4zWKMjFqdx7/
YtSnQiiKNcZGk2kq2KDy7OEpfM34KR7La9JRiN7g5RfwBUbUieSZEvxGeYw5R/
JRvy/9ozv0PdGoQ5qcuUnIiakEtcTnBXfQF+3yPX13ScAAAD//
wMAUEsDBBQABgAIAAAAIQBw8DjcvgAAADgBAAALAAAAX3JlbHMvLnJlbHOEj8EKwjAQRO+C/
xD2btN6EJGmvYggeBL9gCXZtsE2Cdko9u/N0YLgcRjmzUzdvqdRvCiy9U5BVZQgyGlvrOsV3G+nzR4
EJ3QGR+9IwUwMbbNe1VcaMeUQDzawyBTHCoaUwkFK1gNNyIUP5LLT+ThhyjL2MqB+YE9yW5Y7Gb8Z0
CyY4mwUxLOpQNzmkJv/s33XWU1Hr58TufSjQvJoDV1w9s+UsRh7SgpM5G9jIaoi7wfZ1HLxt/
kAAAD//wMAUEsDBBQABgAIAAAAIQBKfWAaGQQAAJMTAAAhAAAAZHJzL3NsaWRlTGF5b3V0cy9zbGlk
ZUxheW91dDEueG1s7Jhbc6M2FMfO9PvwHcPuPC2PC2PC2EO9PvwPC2FMffO9PwPC+sewBmUJbHOEj8EKwjAQRO+C/
dkiLv70IoijXGRpNpKtig8uzhKXv+UsRh7SgpM5G9jIaoi7wfZ1HLxt/
dkiLv70Bm/2Bikn85N5y9z/WlVV86SK11KMXa9j0PX4SKTeSmex+7X2e2HxHW0YSJnlRR87K65dj9N
fv3lukl1ld+xtVwYBxhCp2zsFsY06WCgs4LXTH+UDRfwbC5VzQz8VM+DXLEXYNfVwB8Oo0HNSuF281
Wf+XI+LzP+u8wWNRemhSheMQP266JstKU1fWiN4howNHvfJLNuwFvzImer2Yt8ePrbdWiwWsJtz52A
/9m0yh3BarhxI+uGqVJLQU90M1Oc4xix/FM10+ZR0YT75aNyyhwB3UR30D3ohtFPAcPgYnAw/
dmSWLqaq3pyzVKIhrMau5C0NX7CJJbylXGy9ma2vZsVDyfGZsUfJ0YP7AJgwWZRyHfTenTsjm/dmZW
m4o638aodymDqncy+aUdI8BPdb93L7pcWhj4jvimcLvSI6sa1DykedrymmFpDN5FIgqs4gZIFx4Mo9

PxwPybeMPTCaAjxwtB4gR+GUUCLWBIs0qKb1Kx+k/kaQ/
oE35QSllbaTM264hRqCAhLwWz4gMRWDPcMFx++TttVzeSmKrNvjpEOz0vjfGbacOUYCpJGyjWE2kCm
OwoX+SNT7Msu7K8vnX2wGBhnjSI7MWCvZyXYZAVL4rFiGS9klYMFPiKhgG34z0oQBsWFaoZSs/k8K0
9elHhem4Vt8UKa4jiJ2zwlfnDlkc27lflamhwmskJCM3lqs2AzhmmqlpUHhePUTN3RtilFDvsfLynP
i3tocpTanM8xDfr72PVHWDBP1s0NpQP6W+AojKGrQVD6UIfHVER11GBLvfJGZEEfqpccUxHVUUdbqh
fEHu2EXlgauR8CZHXYcAeb+AnZcC4WWR022mJ9PwETeoa29WvfWmR12HgHG4+C3hk7hUVWh022WGT2
T9mJ2CKrw17tYKMw/k8pQxZ1m909QU0MF4HNsRERWv1dTQ17DPU0vdfUYMe+u3GNbOO6kcLA9tzrXd
Suz+9duKELVs27ztV2FRRaigxeTClIVgasLpxSGD8chsP4QGL2OlcQJaMQ9AjlzpLerzA/
OTmUIfiwJUDd/G1dm/JMityp+JJXJ9J9SKQov02cFaXqD6QieBt4KxfKFL1NHLVi8Vbd35bzk0BQrR
+q26Et/9mhbpOR59d+e7BqdRv3wT8LpuBA0m2F9hz0nq0QebFP3RPPUqdVPAk8PI7haeui4v0l4aLi
WFHHR5mLioNI/09UPLJt7JSKk2ie38mOuxe1RpDW/kK+271eUfKd7nVRcjoeXpT88B84Hefa1z1wie
+E6K91pT6z5mFJ5wl4FQYye0O3Gnj5hadBGLodggz7Mm3yLwAAAP//
AwBQSwECLQAUAAYACAAAACEATY7z/P0AAAC7AQAAEwAAAAAAAAAAAAAAAAAAAAAAW0NvbnRlbnRfVH
lwZXNdLnhtbFBLAQItABQABgAIAAAAIQBw8DjcvgAAADgBAAALAAAAAAAAAAAAAAAAAC4BAABfcmVs
cy8ucmVsc1BLAQItABQABgAIAAAAIQBKfWAaGQQAAJMTAAAhAAAAAAAAAAAAAAAAABUCAABkcnMvc2
xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MS54bWxQSwUGAAAAAAMAAwDJAAAAbQYAAAAAQAAeBOEFAABQ
SwMEFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMAAABbQ29udGVudF9UeXBlc10ueG1sfJDLTsQwDEX3SP
xDlC1q0mGBEGo7Cx4rBCyGD7ASt43IS3E6mv49aWeQAA2sLD+uz7Wb7cFZtsdEJviWb0TNGXoVtPFD
y993T9UtZ5TBa7DBY8tnJL7tLi+a3RyRWFF7avmYc7yTktSIDkiEiL50+pAc5JKmQUZQHzCgvK7rG6
mCz+hzlZcdvGsesIfJZvZ4KOWjk4SWOLs/Di6slkOM1ijIxance/2LUp0IoijXGRpNpKtig8uzhKXz
N+Ckey2vSUYje4OUX8AVG1InkmRL8RnmMOUfyUb8v/aM79D3RqEOanLlJyImpBLXE5wV30Bft8j19d
0nAAAA//8DAFBLAwQUAAYACAAAACEAcPA43L4AAAA4AQAACwAAAF9yZWxzLy5yZWxzhI/
BCsIwEETvgv8Q9m7TehCRpr2IIHgS/YAl2bbBNgnZKPbvzdGC4HEY5s1M3b6nUbwosvVOQVWUIMhpb
6zrFdxvp80eBCd0BkfvSMFMDG2zXtVXGjHlEA82sMgUxwqGlMJBStYDTciFD+Sy0/
k4Ycoy9jKgfmBPcluWOxm/GdAsmOJsFMSzqUDc5pCb/7N911lNR6+fE7n0o0LyaA1dcPbPlLEYe0oK
TORvYyGqIu8H2dRy8bf5AAAA//8DAFBLAwQUAAYACAAAACEAjlP7IK4CAACyCQAAIQAAAGRycy9zbG
lkZUxheW91dHMvc2xpZGVMYXlvdXQxLnhtbOxW23LbIBB970z/geHdkWU1hcKEELasXhnNvJNsfTtU
4UgNAgaIYvfru4DIrY7B79z5sbWSE5rvH1DCJVNlLW8m9GpxsjeixDqQJQ
gl+YSuuaVH08+fxrqwojyFtbpzBDGkLWBCK+d0kWWWVbwBu680l7i3VKYBh5/
mJisN3CN2I7LewcEwa6CWtPM3u/ir5bJm/Jtidw2XLoIYLsBh/LaqtU1oehc0bbhFmOD9PCS31pitu
1cX178oCXamxZWcTjF1NhclkdDgwuJekWMlHcKELasXhnNvJNsfRs/1zASP83ZmSF16hM6TZt1GZxY
+JZrhJHvhfpOQoFgtTTMdQ4FMkNWEYsHWfkQnKPjKERYX2eMqqy422LLq+wbrLB2AETwcirXWMaO/0
+mldBa1E5zkD1lFU0DXU8VuLZEK8/Tpx/TYeZvAfM4eXleko91DdXZxM/CR7C1yGshyq6+qXPvEr/
E3LEIhrJu7teCBEAwbCgTHAekX4FXN5d7V3KND4abHoma3xCnCy9qRM7COG+JCKtajjJEQh/XoULgs
Z2Dg8inYz8sQKhR4GMaZgsJpZO117vqJu05AZCaA8UqJEoPoeVSUWuLpjUza33gBQCwpig4VkWh/
hU5PxgthjfojvKJBXfmoNxj2BpGzpLFBPsqH3sAr7bA/yPv9UUdERAoExMomTv5RKKwrEa14kPPbC+
eDC3WzzwqH5QtKiEM6JXCyXR5zzpQsieAtFzsghpptR1xUtdkdsB8530bEibozrto5xMMdEOvlRkDs
L+8q/8Nt8g+Zv5v8Q2FCE/GCDJOn3WSD/If5l96H/j/0/5/
tPzTB+CLAqX83hKYuzBnoizbca3wp4Z/OcVjS+DbCtu9NH008RnprTf8AAAD//
wMAUEsBAi0AFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMAAAAAAAAAAAAAAAAAAAAAAFtDb250ZW50X1R
5cGVzXS54bWxQSwECLQAUAAYACAAAACEAcPA43L4AAAA4AQAACwAAAAAAAAAAAAAAAAAuAQAAX3Jlb
HMvLnJlbHNQSwECLQAUAAYACAAAACEAjlP7IK4CAACyCQAAIQAAAAAAAAAAAAAAAAAVAgAAZHJzL3N
saWRlTGF5b3V0cy9zbGlkZUxheW91dDEueG1sUEsFBgAAAAADAAMAyQAAAAIFAAAADAAHgSUBgAAU
EsDBBQABgAIAAAAIQBNjvP8/QAAALsBAAATAAAAW0NvbnRlbnRfVHlwZXNdLnhtbHyQy07EMAxF90j

8Q5QtatJhgRBqOwseKwQshg+wEreNyEtxOpr+PWlnkAANrCw/
rs+1m+3BWbbHRCb4lm9EzRl6FbTxQ8vfd0/
VLWeUwWuwwWPLZyS+7S4vmt0ckVhRe2r5mHO8k5LUiA5IhIi+dPqQHOSSpkFGUB8woLyu6xupgs/oc
5WXHbxrHrCHyWb2eCjlo5OElji7Pw4urJZDjNYoyMWp3Hv9i1KdCKIo1xkaTaSrYoPLs4Sl8zfgpHs
tr0lGI3uDlF/AFRtSJ5JkS/EZ5jDlH8lG/L/2jO/
Q90ahDmpy5SciJqQS1xOcFd9AX7fI9fXdJwAAAP//
AwBQSwMEFAAGAAgAAAAhAHDwONy+AAAAOAEAAAsAAABfcmVscy8ucmVsc4SPwQrCMBBE74L/EPZu03
oQkaa9iCB4Ev2AJdm2wTYJ2Sj2783RguBxGObNTN2+p1G8KLL1TkFVlCDIaW+s6xXcb6fNHgQndAZH
70jBTAxts17VVxox5RAPNrDIFMcKhpTCQUrWA03IhQ/
kstP5OGHKMvYyoH5gT3JbljsZvxnQLJjibBTEs6lA3OaQm/
+zfddZTUevnxO59KNC8mgNXXD2z5SxGHtKCkzkb2MhqiLvB9nUcvG3+QAAAP//AwBQSwMEFAAGAAgA
AAAhAGsGxvhhAwAAJgwAACEAAABkcnMvc2xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MS54bWzMVttu2z
AMfR+wfxD83voWx47RpMC6dQPWS9C0H6DaSm1Ulg1J9ZJ9/
UhZdpKuA7w+BH1JZImXw0OK1Nn5puKkZVKVtZg7/qnnECayOi/
F09x5uL88SRyiNBU55bVgc2fLlHO++PzprEkVz6/
otn7RBGwIldK5U2jdpK6rsoJVVJ3WDRNwtq5lRTV8yic3l/
QX2K64G3je1K1oKRyrL8fo1+t1mbGvdfZSMaE7I5JxqgG/KspG9daaMdYayRSYMdqHkPS2gWgVy34w
mjvECMoWtnxnAbFnK54TQSvYWLEMnRMUZNKcquZeMoZyov0um1WzlEbppl1KUuZoxCo7rj2wYuZTgB
gs3FfqT70lmm7Wslqc0RTYIJu5A0nb4i8o0ZRtNMm6zWy3mxW3b8hmxbc3pN3eASAYnEK+my6iv8MJ
+nDuS80Z8YeoOlEKqld19qyIqCFODL8LL7tpe2MYM5pvCtJRr9GUlesODR+9vDKc9kAHJpLQT6KODj
/2ZnGYHJLie5EfTT2QQG6CJAriMDZeelPgpbPdpHrzpc63yOkj/EPqqMiKGsr0sTPKlV7pLYdE05S3
3LeQcra+A2H1e+6AJ5MVYHQQwPWeIpBKUwgdfkCJU7x3TJw8rDofenHBy+yZ6JqwvNTkmirNJDHswM
UE72dgUEO1WCtM5EsqKSIYjP28syGCM4ivj8uEiqT/O7PhkFksqyWnGStqDnVOAjQJF6FP4buSjLw6
cCOgXPuaeFeuJ1Eym0zDjrL+Ahzm2o88z09G59pcpf082fyRisorc+NKkUPrwKWpkJcb6I9Gay//wQ
Tyj8eq5mV+WXJuPrA9sgsuSUv53NEb7CmQxFLobicGrH3dmF6KwibFe3Yg4Z0nczCUl6m0AErRIp1E
MXRaIHkMXHR7LLiIEX0B8nAHd+ZPkLNRcP3kiHARo4U72cH1w9g37WQMvUbyWPQiSIs32sObBAmyNo
5fjOxYeBGkxTvd4Q2CxPTQj4cXQVq88R7eeBKOvm5HrQcEafEmO7wIdvx9O2Y9IEiLd7aHdxrFH/
O+IciuE78e7ogemtzwQDNh/dewx4lmZr06GPYwAd4c6Gaudy9QWOJT1UxqLq9pc9sa9/
A6h2cETBXYauA9bt9jOxG00b/vF38AAAD//wMAUEsBAi0AFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMA
AAAAAAAAAAAAAAAAAAAAAFtDb250ZW50X1R5cGVzXS54bWxQSwECLQAUAAYACAAAACEAcPA43L4AAA
A4AQAACwAAAAAAAAAAAAAAAAAAAAuAQAAX3JlbHMvLnJlbHNQSwECLQAUAAYACAAAACEAawbG+GEDAAAm
DAAAIQAAAAAAAAAAAAAAAAAAAVAgAAZHJzL3NsaWRlTGF5b3V0cy9zbGlkZUxheW91dDEueG1sUEsFBg
AAAAADAAMAyQAAALUFAAAAACAAHgSCBQAAUEsDBBQABgAIAAAAIQBNjvP8/QAAALsBAAATAAAAW0Nv
bnRlbnRfVHlwZXNdLnhtbHyQy07EMAxF90j8Q5Q5QtatJhgRBqOwseKwQshg+wEreNyEtxOpr+PWlnkA
ANrCw/rs+1m+3BWbbHRCb4lm9EzRl6FbTxQ8vfd0/
VLWeUwWuwwWPLZyS+7S4vmt0ckVhRe2r5mHO8k5LUiA5IhIi+dPqQHOSSpkFGUB8woLyu6xupgs/oc
5WXHbxrHrCHyWb2eCjlo5OElji7Pw4urJZDjNYoyMWp3Hv9i1KdCKIo1xkaTaSrYoPLs4Sl8zfgpHs
tr0lGI3uDlF/AFRtSJ5JkS/EZ5jDlH8lG/L/2jO/
Q90ahDmpy5SciJqQS1xOcFd9AX7fI9fXdJwAAAP//
AwBQSwMEFAAGAAgAAAAhAHDwONy+AAAAOAEAAAsAAABfcmVscy8ucmVsc4SPwQrCMBBE74L/EPZu03
oQkaa9iCB4Ev2AJdm2wTYJ2Sj2783RguBxGObNTN2+p1G8KLL1TkFVlCDIaW+s6xXcb6fNHgQndAZH
70jBTAxts17VVxox5RAPNrDIFMcKhpTCQUrWA03IhQ/
kstP5OGHKMvYyoH5gT3JbljsZvxnQLJjibBTEs6lA3OaQm/
+zfddZTUevnxO59KNC8mgNXXD2z5SxGHtKCkzkb2MhqiLvB9nUcvG3+QAAAP//AwBQSwMEFAAGAAgA

```
AAAhAE0UC8pPAgAAUQYAACEAAABkcnMvc2xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MS54bWysld9u2y
AUxu8n7R0Q962TdJomK0mlZesu1j9Rkz7AKSYxKwYExIvffgcw6bplkSPththwzsc5Pz6T6fW+kaTl
1gmtZnR8OaKEK6YrobYz+rS+ufhEifOgKpBa8RntuKPX8/
fvpqZ0srqFTu88QQ3lSpjR2ntTFoVjNW/AXWrDFa5ttG3A46vdFpWFn6jdyGIyGn0sGhCK9vl2SL7e
bATjXzTbNVz5JGK5BI/1u1oYl9XMEDVjuUOZmP22JN8Z7FY//6AkBtkWX8d0jn2zlayIggYn1sJLTp
AOWWjlUSkGOLO2nIdQ1X6zZmWWNubdt0tLRBV0+nxa9At9WHxVGIYPxR/p26wE5X5jm/kUSoRB9jOK
Z9aFEZOg5HtPWJpkr7OsfjgSy+qvR6KLvAFWcNgUj9ukjv5uZ5LbSTjGh65SKGDqrWYvjiiNfYb2U3
vsvs1ioecgb2qSyPtAto9Li5FHjnfINMLy+8+66kLjz/gbJ6GUzq98J3kEgmVDieI4IH4JwdhcXTyt
gjqUfr6Qgr0QrwmvhCd34Dy3JO6PzkeVKQLxeB69ClfVEiw8/i72/TGWCiVuhnXmovAxUfs3u6vMrj
cQWUpgvNaywiImQRUNlzmdSVJU6IMM+z9AROZEtvJgtfOhBnNGpu4NVEQbTykNeZdY+emjW3Gm8eOT
vOVygGLkeVpxXQs7XPAqeegUiBu9s74eXOKHAYpic1QQv/2zrRkdmm4rfAw3WzSctHdgHtrYF17k+E
Es4pTBqxstGUJfQ4JG/iuY/wIAAP//AwBQSwECLQAUAAYACAAAACEATY7z/P0AAAC7AQAAEwAAAAAA
AAAAAAAAAAAAAAAAW0NvbnRlbnRfVHlwZXNdLnhtbFBLAQItABQABgAIAAAAIQBw8DjcvgAAADgBAA
ALAAAAAAAAAAAAAAAAAAC4BAABfcmVscy8ucmVsc1BLAQItABQABgAIAAAAIQBNFAvKTwIAAFEGAAAh
AAAAAAAAAAAAAAAABUCAABkcnMvc2xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MS54bWxQSwUGAAAAAA
MAAwDJAAAAowQAAAAEAAeBDoGAABQSwMEFAAGAAgAAAAhAE2O8/z9AAAAuwEAABMAAABbQ29udGVu
dF9UeXBlc10ueG1sfJDLTsQwDEX3SPxDlC1q0mGBEGo7Cx4rBCyGD7ASt43IS3E6mv49aWeQAA2sLD
+uz7Wb7cFZtsdEJviWb0TNGXoVtPFDy993T9UtZ5TBa7DBY8tnJL7tLi+a3RyRWFF7avmYc7yTktSI
DkiEiL50+pAc5JKmQUZQHzCgvK7rG6mCz+hzlZcdvGsesIfJZvZ4KOWjk4SWOLs/Di6slkOM1ijIxa
nce/2LUp0IoijXGRpNpKtig8uzhKXzN+Ckey2vSUYje4OUX8AVG1InkmRL8RnmMOUfyUb8v/aM79D3
RqEOanLlJyImpBLXE5wV30Bft8j19d0nAAAA//8DAFBLAwQUAAYACAAAACEAcPA43L4AAAA4AQAACw
AAAF9yZWxzLy5yZWxzhI/BCsIwEETvgv8Q9m7TehCRpr2IIHgS/YAl2bbBNgnZKPbvzdGC4HEY5s1M
3b6nUbwosvVOQVWUIMhpb6zrFdxvp80eBCd0BkfvSMFMDG2zXtVXGjHlEA82sMgUxwqGlMJBStYDTc
iFD+Sy0/k4Ycoy9jKgfmBPcluWOxm/GdAsmOJsFMSzqUDc5pCb/7N911lNR6+fE7n0o0LyaA1dcPbP
lLEYe0oKTORvYyGqIu8H2dRy8bf5AAAA//8DAFBLAwQUAAYACAAAACEAogD1XAcDAABGCQAAIQAAAG
Rycy9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQxLnhtbKyWW2/
aMBSA3yftP1h5b0MSCGlUxqWW2/
eTXF5tWk7WTOmmE6MoOe9FhImqqxuxGEVPj9/PiohoQ0VNeSfYKNoyHV2NP3+6lKXm9Q3ddDgCF0
SUfR0hhZxrGulqyl+ryTTMCzeadaauBWLeJa0V/Abnmc9np53NJGRF5fvUa/m8+bin3tqlXLhHEQxT
g14L9eNlIHmnwNTSqmAWO1j10yWwnRmsZwFhErptZwkERjiLya8poI2sLBI0qQKW9qZh9p+agYQyGx
/qHkVN4rqzFZ3yvS1EjwmlHsH3gxeytADC7iv9QXgUTLzVy140taQiLIZhRBvbb4C0q0ZBtDKndY7U
+r5d0LstXy2wvScTAAHuyMQqmli+jfcNIQjktEsovKiVJQvemqZ01EB3Fi+C68arIOMIwZ8XJJXNYr
oyzNi7rnNiVBRdu0Bl93yUgGab/XcylJkjTN8uw4MRdJ3wlgetKsGOYgjQ4FFFhxbFmazZeu3mJaZ/
AP1aOiWnbQpDPH5NpMzZZDrWnJ1zwBlwjlC5gi8B96gZY1mz/
Aof49isBOMORlrdFDBqSYlpAI+AElTnEAmTh7mjpzZnzNm+qZmI6wujHklmrDFDG2/
zQ6glEYi7UUJup7qih6sIP9fPDRgjEINYRoo8YS/L/
OWajzdDVzNlNEwSyEQr6r1Ho1c6WG2YDGDd3xvpJneS/tZYXLV5iFo5In+WAwzK3nrym5HarDGoU6t
1Td2NlrRA37w14e1362msC+tICDNsD29BXYtYHtnhS7x1H7gyGsxoi8FX3UYcjzjZnt0S4Xb0YnxaH
XyPPo/h6dZMMEh+mtblslN4GQEQR69uCAXaQFunAaG4Gene/ZaVrY2TyNjUDPHh6wh/3sPaU8zgkCP
bvYsxGMzXSa3wj07IsDdg5DcjIbgR+05GBLfOSes+vOvabhEl/
mdpFxdUvl3domBL5eYLte2yMJ3yvYoSC6F0FG+P4Z/wEAAP//AwBQSwECLQAUAAYACAAAACEATY7z/
P0AAAC7AQAAEwAAAAAAAAAAAAAAAAAAAAAAW0NvbnRlbnRfVHlwZXNdLnhtbFBLAQItABQABgAIAAA
AIQBw8DjcvgAAADgBAAALAAAAAAAAAAAAAAAAAAC4BAABfcmVscy8ucmVsc1BLAQItABQABgAIAAAAI
QCiAPVcBwMAAEYJAAAhAAAAAAAAAAAAAAAAABUCAABkcnMvc2xpZGVMYXlvdXRzL3NsaWRlTGF5b3V
```

0MS54bWxQSwUGAAAAAAMAAwDJAAAAWwUAAAAAAAAcBAQAAAABAAAAIAC6DxgAAABPAGYAZgBpAGMAZ
QAgAFQAaABlAG0AZQAPAO4DAwMAAAIA7wMYAAAAEAAAAAAAAAAAAAAAAAAAgAAAAAAHAAAADwAMBGM
CAAAPAALwWwIAACAACPAIAAAAAwAAAAIIAAAPAAPwQwIAAA8ABPAoAAAAAQAJ8BAAAAAAAAAAAAAAAA
AAAAAAAAAAAgAK8AgAAAAACAAABQAAAA8ABPD0AAAAEgAK8AgAAAABCAAAIAIAAPMAC/BqAAAABAA
AAAAAfwABAO8BgAAA8vMBgQAwZQEAggCYsgAAgwAwZQEAhACYsgAAhQAAAAAAhwACAAAAiAAAAAAAv
wAAAAYA/wEAAAEAAQMCBAAAgMMQAAAAvwMAAAIAVABpAHQAbABlACAAMQAAAAAAEPAIAAAAwwLAA0A
aowgPABHwHAAAAAAAwwsIAAAA/////w8AAAAAAB8EBAAAAAIAAAAPAA3wNgAAAAAAnw8EAAAABgAAA
AAAqA8EAAAAdGVzdAAAoQ8WAAAABQAAAAAAAAgAAAEABQAAAAAAgA8AA8ABPAPAQAAEgAK8AgAAAA
CCAAAIAIAAPMAC/BwAAAABAAAAAAAfwABAO8BgACA3vMBgQAwZQEAggCYsgAAgwAwZQEAhACYsgAAh
QAAAAAAhwAAAAAAiAAAAAAAvwAAAAYA/wEAAAEAAQMDBAAAgMMWAAAAvwMAAAIAUwB1AGIAdABpAHQ
AbABlACAAMgAAAAAAEPAIAAAA3QjAA0Aa8AwPABHwHAAAAAAAwwsIAAAA/////xAAAAAAAB8EBAAAA
AMAAAAPAA3wSwAAAAAAnw8EAAAABQAAAAAAqA8HAAAAY29udGVudAAAoQ8UAAAACAAAAAAAAAAAAAAg
AAAAAAAIAGAAAAKYPDAAAAPAAAACwAdAC8AMQBRAA8AcgAAAA////AAAAAADn5uYARFRqAERyxADtf
TEABWPBAJVPcgAPAIgTOAAAAA8AihMwAAAAAAC6DxAAAABfAF8AXwBQAFAAVAAxADAAAACLExAAAAA
AAOsuCAAAAAnG2QEgqN1FAAAiBAgAAAABAAAAAQAAAAAAchcQAAAAAQAwAAAAAADaCwAAMYsAAAAA9
Q8cAAAAAAAAAAICAAMAAAAAPI4AAAEAAAADAAAAAQAcAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAA/v8AAA0EAQAAAAAAAAAAAAAAAAAAAAAAAQAAAOCFn/L5T2gQq5EIACsns
9kwAAAAEAEAAAsAAAABAAAAYAAAAAIAAABoAAAABAAAAHgAAAAIAAAAjAAAAAkAAACgAAAAEgAAAKw
AAAAKAAAA1AAAAAwAAADgAAAADQAAAOwAAAAPAAAA+AAAABEAAAAAAQAAAgAAAOn9AAAeAAAACAAAA
HRlc3QAAAAAHgAAAAwAAADqsJXtmITqtawAAAAeAAAADAAAAOqwle2YhOq1rAAAAB4AAAAEAAAAMQA
AAB4AAAAgAAAATWljcm9zb2Z0IE1hY2ludG9zaCBQb3dlclBvaW50AABAAAAAENLxHAAAAABAAAAAw
L3cRQnG2QFAAAAA4LbOYgnG2QEDAAAAAgAAAEcAAAAIAAAA/////wMAAAD+/wAADQQBAAAAAAAAAAAA
AAAAAAAAAAAABAAAAAtXN1ZwuGxCTlwgAKyz5rjAAAACoAQAAEAAAAAEAAACIAAAAAwAAAJAAAAAPA
AAApAAAAAQAAACwAAAABgAAALgAAAAHAAAAwAAAAAgAAADIAAAACQAAANAAAAAKAAAA2AAAABcAAAD
gAAAACwAAAOgAAAAQAAAA8AAAABMAAAD4AAAAFgAAAABAAANAAAACAEAAAwAAABSAQAAAgAAABAnA
AAeAAAADAAAAFdpZGVzY3JlZW4AAB4AAAAEAAAAAAAAAAAAMAAAB4jgAAAwAAAAIAAAADAAAAAQAAAAM
AAAAAAAAwAAAAAAAAAADAAAAAAAAAMAAAAAABAACwAAAAAAAALAAAAAAAAAsAAAAAAAAACwAAA
AAAAAAeEAAABQAAAAgAAABDYWxpYnJpAAYAAABBcmlhbAAOAAAAQ2FsaWJyaSBMaWdodAANAAAAT2Z
maWNlIFRoZW1lAAUAAAB0ZXN0AAwQAAAGAAAAHgAAAAsAAABGb250cyBVc2VkAAMAAAADAAAAHgAAA
AYAAABUaGVtZQADAAAAAQAAAB4AAAANAAAAU2xpZGUgVGl0bGVzAAMAAAABAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA9g8cAAAAFAAAAF/AkeNUjgAAAAD0AwMAAAAIA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEMAdQByAHIAZQBuAHQ
AIABVAHMAZQByAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAaAAIA/////
///////////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADQAAACAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAD//////////////8AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAP//////////////wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA//////////////
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

langchain4j\src\test\resources\test-file.pptx

BASE64:

UEsDBBQABgAIAAAAIQDfzBj1rQEAAEYMAAATAAgCW0NvbnRlbnRfVHlwZXNdLnhtbCCiBAIooACAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADMl1tPwjAUx99N/
A5LXw0roCIaBg9+gbgeodm3TFoRv79kFgoSbArEvS87O+f/dq3OJBXBGIzlSkakFlZ
JADJWCZeDiLz1HipNEljHZMKEkhCRKVjSaZ+etHpTDTZAtbQRGTqnbyi18RBSZkOlQWKmr0zKHIZmQ
DWLP9kAaL1abdBYSQfSVVzmQdqtO+izkXDB/QRvFyQfGgYkuC0Ks14R4WlmkCfoSo0BYZc0TGvBY+Y
wT8cyWSKrlFQhKvMaO+TanmHBmg5ZZn2DUveCj9PwBIIuM+6ZpVhFtXZUG7Coy2vDzU4rUFW/z2NIV
DxKURIumqXiRximjMvZINbBWIE3n5h1OPWLQe3QZAveOzGVNMfh2EaQabpGaXuM+cmNtxGMOXwdhWB
uvI3A4TsMxXX/SchttnZk7wJe3VTAwUe9YL3T6ntkUzVy5RosguOsxML7r0x1D5nOPWS68JDp0kOmh
odMVx4yNT1kuvaQqVb1Eeq/vuQoz3dePPca+D3D7JCaqSsajcA4vnk/m3dE670HDdn5N4FkRW+a/
wW0vwEAAP//AwBQSwMEFAAGAAgAAAAhAGj4dKEDAQAA4gIAAAsACAJfcmVscy8ucmVscyCiBAIooAA
CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACskttKAzEQhu8F3yHMf
TfbKiLSbG9E6J3I+gBjMrsb3RxIptK+vaHgYWEtgr3MzD8f3yRZb/ZuFO+Usg1ewbKqjBCKjNzgGTwoOlGHTXF6sn2hELkN5sDGLQvFZwcAc76TMeiCHuQqRfOl0ITnkcky9jKjfsCe5q
usbmX4yoJkwxdYoSFtzBaI9RPofWzpiNMgodUi0iKlMJ7ZlF9Fi6okVmKAfSzkfE1Uhg5wXWp1XiIe
de/FoxxmVr171Gqn/
TWj5d6HQdVbTfdA7R57nvKaJb6cYWcZEuRSP6VM3dH1OIdozeUPm9KNhjJ9GcvIzmw8AAAD//wMAUE
sDBBQABgAIAAAAIQBjXCO0wAAAADcBAAAgAAAAcHB0L3NsaWRlcy9fcmVscy9zbGlkZTEueG1sLnJl
bHOMz71qwzAQB/A90HcQt1eyO4QLGUpBUOnkD7AIZ1tUVsSOrnEbx+NMXTIeF+/P9dd7sss/iizj0
FDKxsQFGx0Powafm5f7ycQXDA4nGMgDRsxXMzbobvSjKUe8eQTi6oE1jCVks5KsZ1oQZYxUaiTIeYF
Sy3zqBLaXxxJfTTNUeVnA8zOFL3TkHvXgrhtiV6x4zB4S5/
RrguF8k+E4tk7+sYtrqWymEcqGqR87u+WWlkjQJlO7d41DwAAAP//AwBQSwMEFAAGAAgAAAAhAH6d4
suxAgAADAcAABUAAABwcHQvc2xpZGVzL3NsaWRlMS54bWzMVMtu2zAQvBfoPwi6M5Koh2UjTmC9iqJ
pE9TJBygUHQuVSIKkHRtB/r0kRcVN7QK+BOhFu1zukrszI15e7/
rO2WIIuWkrmbnDhuw4miDYteZq7D/

```
cVSF1HyJo0dUcJnrt7LNzrq8+fLtlMdI2jqomY1XN3LSWbeZ5Aa9zX4oIyTNTeivK+lmrJn7yG18/
q1L7zoO8nXl+3xLX1/
Jx6ulq1CBcUbXpM5HAIx10tVedi3TIxnsbOOY1xLNQxpvpdS1dqMrTsGm0Fu+cYa49sv3C2ZHfcbP/
Y3nGnbRRerkPqXsHienbDppkl2RrH+6v8aXTr2W7Fe23VbM5u7irw9/rr6RjeSQcNQXSIovXtiVy0L
k9ke+MF3h+X6qmG5o7HgeM4963ssBO4to8bIceONryduy9VBbO4rCJQKQ9EfhaBrIymoIJhWsJJlcM
wedXVQTJDHBuUv76pJUiOGOpbxKmgK3mBaG+pHhWjyAkiqxfd5UuxgJGfhwUIimkK0jIqwaSMQxBPi
zLwg7T0J8WrBUD1PFozhWfntYOPRAh2Q9Ev4RCqiNK8Dry9ZQxkasvWjtwzhRGS3MBkU4d94xyAtkK
Qu4w2e33Po7ImWM86IZdy32GzYPpjOuGKi67Wfx8m4GHpOk3L5YFpeSX1OMNQQ8FR1befx1VYvK/zz
J3eoTdvFMa/5RGO8lhuHqVRCPxPFQLzogqyRQAW1bQCcFFlII6TBMASwhRWkzwP0o9XiNg8DgpRTe0
OJR+ilFOcI0qkeuPO4t2Y8bUboTCeJTTLpgnM0wxkQVSBqJhOFLhJDKo4jKI8Sxd5WGpCWRAdE6qC5
xHK6DPmjLbmgQ98y+m27pT64tgP1e8dJRbHgbhDt5oN+3Cjjn+v2e3W4KQuk5jnJsS0WobUQ4qeXdX
9BgAA//8DAFBLAwQUAAYACAAAACEAGy41BwwBAADQAwAAHwAIAXBwdC9fcmVscy9wcmVzZW50YXRpb
24ueG1sLnJlbHMgogQBKKAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACsk0FOwzAQRfdI3MGaPXFSoEK
oTjcIqQskBOEAJpkkFo5teUwht8dqoSRVFXWR5f/2/HmasVfr706zLXpS1gjIkhQYmtJWyjQC3orHq
ztgFKSppLYGBfRIsM4vL1YvqGWIRdQqRyymGBLQhuDuOaeyxU5SYh2aeFJb38kQpW+4k+WHbJAv0nT
J/TAD8lEm21QC/Ka6Blb0Ds/JtnWtSnyw5WeHJpxowZ1HevY2NmOF9A0GAQcriWnAT0Ms5oQgrSr8B
9jJXzebgshmh3iSFNAfoezN0Y1JrOWcWEG+a3wNvcbBigbmFMjtrCCxdrCkndybk8O4mZNhq/
Dr6LUerD8IPvqH+Q8AAAD//wMAUEsDBBQABgAIAAAAIQDMrLSnQAIAALQMAAAUAAAAcHB0L3ByZXNl
bnRhdGlvbi54bWzsl12O2jAQx98r9Q6WXys2JIQkIMJK3QqpKishoAfwOgaidezIdijs6Ts2hgSqSn
uAvNme/3z9MnKS2fOp4ujIlC6yHH4NMSICSqLUuxz/Hu7GGQYaUNEQbgULMdnpvHz/
OuXWT2tFdNMGGLAFUEYoackxwdj6mkQaHpgFdFPsmYCbDupKmJgq/ZBocgfCF/
xIBoOk6AipcDeX33GX+52JWU/JG0qSH8Johh3dehDWetrtPoz0bpd3JdEZVUT8yoLaDsECuTINs2bZ
mYhhdHujDRGvoAMYuhVSU0DixwP8RzwaF68Em2Y+lkstXk4QWWR4yiM0zgbJTEwVlN7ApYQB/
NZ8B/3+/UlyDjpeEetd1e7+UD0BBVH4QQahGdMzzlOsnFmN4EVCWmY9rKrwakmYRzfVAXbkYabLTuZ
jTlzNp8Re7ZaKb9arxTixM4OE4Nfa1dNV8KPPKxBUxG1tKAQ4XuYO44RaLbkbfNxzQhNGe4kjCzFd/
XugNunLPwWTAdIBaO0agQ11t6pQkOkMLNx3pmyow2NO7uWvCwWJeduYweDvXCFjgSymdMF/4PKZUXm
XEP7FKbhWyUG3FglmTLyYGDkYqD6wUB1i2NtcQQ3Hh5N1KKJx6ktuOfjoHg+o5bPFULPx0LxfOKWTz
hKw6QHdKXiAY07gLIoc9X3gCwVDyhpAUVRlri3QA/IUvGA0g6gNB71d/
SNigeUtYAsnf6SvlHxgCYdQMk47S/pGxX35frvJ2Zw/6sx/wsAAP//AwBQSwMEFAAGAAgAAAAhANXR
kvG8AAAANwEAACwAAABwcHQvc2xpZGVMYXlvdXRzL19yZWxzL3NsaWRlTGF5b3V0OS54bWwucmVsc4
zPvQrCMBAH8F3wHcLtJq2DiDR1EcHBRfQBjuTaBtsk5KLo25vRgoPjf3i+XLN/TaN4UIZaViDI
m2Cd7zXcrsfVFgRn9BbH4EnDmxj27XXsRXGjEXI54cJFFUTxrGHKOO6XYDDQhyxDJl0kX0oS5lKlXEc
0de1Lrqtqo9G1AOzPFyWpIJ1uDuL4j/WOHrnOGDsE8JvL5R4Ti0Vk6I2dKhcXUU9Yg5Xd/
tlTLEgGqbdTs3fYDAAD//wMAUEsDBBQABgAIAAAAIQBL4kKCBwUAAH0SAAAhAAAAcHB0L3NsaWRlTG
F5b3V0cy9zbGlkZUxheW91dDEueG1szFjtbpw4FP2/0r4DYn+74A8MRJ1UfK5WTduo0z4AAU8Gla8F
zzTZqlJfa/dx+iR7bSAz06TdpElW+TM2xj6+957DwxcMfTo4+xjC+957DwczzFxd1ZWxFP5RtszDxM9s0RJO3RdmcL8z371
LkmcYgs6bIqrYRC/NSDOaL419/ed4dDVVxkl22G2kARjMcZQtzLWV3ZFlDvhZ1NjxrO9HAvVXb15mE
y/7cKvrsI2DXlUVsm1t1VjbmtL6/zfp2tSpzEbf5phaNHEF6UWUS4h/WZTfMaN1t0LpeDACjVx+GJC
87yFaWshKmoaf1WxjA5jFkni+rwmiyGgbeqRnGsioLoW8N3bteCNVrtr/33bI77fWK19vT3igLhTCt
NK3pxjRNXzZb3bG+WX4+d7OjiVfqxYKYVwsTODrUv1aakxcSCMfB/PdaL5+c8PcfJ3cMNuaN7D2Nl
```

VZjcFdT4fM6YyFUPXRcZwMco5o05cL81OaktBJUoZS6CFmhwyFCfNRSqiXEDeNCOWf1WrMj/
JeaEr+KGZpYX6NzrrM+3ZoV/JZ3taTLmZ5AZOYTUyqKD8xO/Ld0MWIEpKggCcR4gHhKA08TplHQpdF
n6cCQMxzq7OwpnynxGcihu6kzT8MRtMCUYrXkberGSOZqu3Wk5xy2esyTVPH+7qzK/
SNLGOHMNse+cOYQKnoIeM+ZuMExSRU1OX2NT5H7O5IXoRtcalWn0ELPGZNvm7h6TsbMatBLuVlJXR/
W+FOTanOGx2/5rcQq7cwOPy1MLnaddxomjv29zA69aOz6mFRlSlnEQ16vxy3k8dRVeYfDNkaoiil8S
obpOgN/eiB9QCIAhwp0SiiKU6zPlMRXIG9fDsF0elU5xR11j8WMJ0FvNycjXuSJ6phSqIwxQFFgRc7
KKUxQ9wjDNEo9sLIp5Sn3uNreNicjRqGoJQy76Vlym1iU+8HWsbccVxObqvl7wq4zvoTbXVlU4Dj6+
6hqM82r+ENpwH29K1i/Vbfukt2qMxxiYr3rtAHj47Cm6DpDnqsxZ2hsbcPrfAmaLaDxtTFyiXujL2z
lglwwnb2sD3iqRDuh60AJ2y+wybE06ZzP2wFOGG7e9guoz9D5SG2ApywvR22Av4pLg+wFeCE7e9hc0
c7/v2wFeADufcwm+njGzibDTzOpDBOqywX67YqIAj6RI3cpQG2Y5IiGiYBcqjHkR2EYOm2a/OIpH7s
pY9v5IU0tQbWWbWazXwUw3fdXB9Rf2i5+kJLZgVH5PG15To+PLMYhSljyE8ZnL98HyMXw4OcMjsJHP
p5PnAXwKEsa5GW55tevNlITeGB8kApxlDLqBJZc3WalcfEJtSyPcumO7VBDA+vN2fWW9q2Sur7imNP
VHGBz1wngFNDHMYJolByFDIfeqFLsJ/QFHP2+IpbgRdpyf25yXoo3ay6/zhD3EV1D0s1vzobqu874/
WmPvuGcOepEh4mNHFphBxmY0QIcJ34hKIotm0fuxT7nvM/
nBWrAmp2I+fjie5hnSZI4SBMuYcin3HE4ZMOefBhB04TxmEAjktiduU0g6K0gehuazBfv/z929cv/
zyAu+hm/ptgrrruTdoJQ5+TyAtRiFmKWOy7KEg5vCAcylgUekFEE6WdDrPr2oHB22mnaz+KvmtL/
TcKtif5bLMKzhPMJYRhl08sjRLZBat4X6r0oa36V1n3Zqs1Uuu3f6SHOqXLcepuikp9/
tvo+F8AAAD//wMAUEsDBBQABgAIAAAAIQCbF5vMVAQAAIkPAAAhAAAAcHB0L3NsaWRlTGF5b3V0cy9
zbGlkZUxheW91dDIueG1szFfbjts2EH0v0H8Q1GdGIkWJlBFvYN2KopvsIt58gFai12okUaVox24QI
L/Vfk6+pCQlrffWxgW8xb6YFD08nJlzZkS9frNramvLRF/
xdm7DV65tsbbgZdXezO0PVxmgttXLvC3zmrdsbu9Zb785+/GH192sr8vzfM830lIYbT/L5/
Zaym7mOH2xZk3ev+Ida9V/
Ky6aXKpHceOUIv+ksJvaQa4bOE1etfa4Xxyzn69WVcESXmwa1soBRLA6l8r/fl11/YTWHYPWCdYrGL
P7vkty36lo+fVvtmWMxFY9QvtMxV0s69Jq80YtXFWyZpbKjhXzViokY9B3V4IxPWu3P4tu2V0Ks+/d
9lJYValxxv22M/4xmpnHdmsmzoPtN9M0n+1WotGjSoa1m9uKs73+dfQa20mrGBaLw2qxvnjCtlinT1
g70wHOnUN1VINzj8NBUzhDOnSWjB/nvZw82ohqbn/OMhT5aYZBpmYAuxEGUYpDkCGPpohkMfKCL3o3
DGaFYIaWX8pJXjB4RGlTFYL3fCVfFbwZtTFJTLEJ8cim9vIzDMOEeCgBXpASsAg9ClKiTvfDxcKDEH
tx6H0ZE6B8nkYThTPGOwY+EdF357z42FstV0RpXgfebi0GMvXYrUdJSZ2j0W7400wOWR5VIHcRL/
f6kGs1msV8VvdyKfc1Mw+d/
jFuCEVEneuKZS34sBzIlWdxXRUfLcktVlbSepv3kgnLnK9KWqHoAIcwDQpry8tc5O/vgv36fkxIZ/y
cnHImOfyzKLxJFGNlWJd1XrA1r0vlBHqhEiGhm7ievwAedCOQhmkEQkJdQFxCYBoQ38+eVSJVuTuYn
EAdneZyW98W/H9Xi2bGiKW/p5ZBEQ9PMZ7/+ylLVnDVLmu2ZfURiOj7iFfrShwP6H0fMOMbIddHI+I
jEKvVk4Cnrjk81VySS3av4LyX2pNJ4tMo8wHFQQzSABJAgiAAkC78gNAUL/D/0JNLqS45f6hI8nqlH
dNFOLwKT1KFK3VfMNF6xA8RohBEGcYgzHCqXkMhBAQiGmTYTRe+inZySnEoq4Zl1c1GsIuNvls8UJh
SitU3Mq5Z3t7WuDxDLvIclzqud1Cb8uH0evMnvWWc625xV3H4hSrOJSijRInNd2MIXIy8QXs4DhOaU
S90IXl+xa2kGCT3+yYXKnWT6k7Y+09LdTBRvayrklnvNs31A8L9F0o4TnACkUuA5y4ykFIagQRhAlz
1dqd+ikIaoecnXH0sqZw9ybl5x5240yyyhEZeQEEc4gAEBMeARgSrThMl0cJNEErwbafpNaWt8u7YB
vPt658/
ffv61wm6ixmmr6Up62Y2aieKwgDFirMI4gzgJFQ39yzwQeZ7GMcRXcReqrXTQfxYO2rxOO10/
BMTHa/MFyV0R/
lsc32nQZASCl0ajDQNGjl4q4lf6vjVWIu3eXexNSJpzA0qNkudFuZgejDRsU+f0Gd/AwAA//8DAFBL
AwQUAAYACAAAACEAYyGBtz8FAABeFQAAIQAAAHBwdC9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQzLn
htbMxY247bNhB9L9B/ENRnRiJFitQi3kDXtugmWcTJBygSvRaqWyna2W0QIL/
Vfk6+pCQl2d5kkzptYvjFoqiZ4Rmew9HIj5/cNrW15WKounZhw0eubfG26MqqvVnYr15mgNnWIPO2z

```
Ouu5Qv7jg/2k8sff3jcXwx1eZXfdRtpqRjtcJEv7LWU/YXjDMWaN/
nwqOt5q56tOtHkUt2KG6cU+RsVu6kd5Lq+0+RVa0/+4hj/brWqCp50xabhrRyDCF7nUuEf1lU/zNH6
Y6L1gg8qjPG+D0ne9SrbgRe/8Ly0LWMotmoK2pcq92JZl1abN2piyQvtbmlDLszToX8pONejdvuz6J
f9tTBOz7bXwqpKHWRytp3pwWRmbtutGTgfud/Mw/zidiUafVW7Yd0ubEXanf519By/
lVYxThb72WL9/AHbYp0+YO3MCzgHi+qsRnCfpoPmdF5WsuaW3iKD42qQM6KNqBb22yxDEUkzDDI1At
iNMIhSHIAMeSxFNIuR57/T3tC/KAQ3vPxazvqC/
iecNlUhuqFbyUdF10zimDWm6IR4olOjfBv4mJCIeSAkJANZEmOAUOAChmMfoiTwExS/
mzZAYZ6vJgtnyndKfCZi6K+64vfBajtFlOZ15G1nMZKpr/160pTUezTZjQ/NYL/
LD1LMPMjIyB2kbkA9dp9t6BJIfHeiETGCqEc/JnOM3V/I26gr77T7a3VVJOZtse7U+Xs9Bq0HuZR3N
TfjbQ0nSCVfvVDGw58LW600S2VnoMcHjr3+MX5COdW5Lii8Ba+W4xryMq6r4ndLdhYvK2k9zQfJhWV
2R1UcFUQHHEkwUXhbXuci1wh2wX57MYHoTX5zXibVL0vW20lW7+B1nRd83dXqAFvoXNWr9IrTgIAYE
gjcCEIQRkrCCfUykrIoJTD5/urVgtGAtOT+j4gxYQH2vS+JGBLXhexoEX9OuVaTiytT4Kq2VMVeD43
X5pl6oxmvA2Ej7I6Ph66uyqyqa3OjCeNxLaxtXqsjfDuWOFm1cpyhZH8gdsbj3T6OM690/9yYIdojx
YQivQdHwdXLngquxjjB9fZwA4j1nh0FF7ITwtUYJ7h4Dxd6FBqJHYVXW54KrwY54SUHeBlietfOD68
GOeH193gRYublcH54NcgJLz3AS7F39HE7KV4NcsLL9ng12OPP2ynxapAT3uAAr0/
oeZ43DfLhrkWjVwa7Nvnruxj9RjNNzHCvi/kvnQqeO5Ukl/xep+KdaacSkhS7KEyAmzAEKKIxQIwyg
FPqxUGKMhefoM8upW00tc7r1dyxjCL5bMtivr6+2FeYG6ORlfoANNl6lASq6EEQZRiDIMMpCIMAAgp
VJcywm4bEezd/UJaKQ1k1PKtuNoI/30hD4T1pqZ7WGhoZ1zxvdwqUl8hFnuMyx/X2ilIYvn1nTGa9Z
V2ntXyoOHymioMBVZ90amEYQRd4oUcBVc0yIKmb+cRzcRieoDdeSTFK7o9NLtTWzar7l0b5a1T3ban
2Z6qXqkpy69mmef0R4eRcCffjIAziBIQwZCBkJNKsB8CLVIFRDR6DEf3+hA91qfbsQc7Rd6g0YZawy
PMZiNWnE/ApjgGLKFaVJkqi0E0QSvCu0gya0lahO7bAfHj/108f3v/9DaqLucz/
gM27bkaTdqIo8FHMIhBBnAGcBBSE6pyCjHgYxxELYy/V2ukh/lQ7avI47fTdGy76rjJ/
E0J3ko9pGDyKIKS+545flwbbfN1pZKnzV9daPM3751sjksa832Mz1WthjqZ7E537/L/o5T8AAAD//w
MAUEsDBBQABgAIAAAAIQBt8fretwQAAMITAAAhAAAAcHB0L3NsaWRlTGF5b3V0cy9zbGlkZUxheW91
dDQueG1s7Fhbbt02EP0v0D3MWiaBIbsbMAReL1VaNXKd5ru0GAbKtdTlZSkpLs+J
H4prALf+RHpKjhcGbOmdFIz1+ct42zFXKs+27fhc981xFd2Vd1d7rvvj3hgLnOqIquKpq+E/
vuhRjdFwc///R82Bub6mVx0W+Uo3V0416x766VGvY8byzXoi3GZ/0gOv1s1cu2UPpWnnqVLM607rbx
kO9Try3qzp33y13296tVXYqsLzet6NSkRIqmUNr+cV0P46Jt2EXbIMWo1djd101SF4P2Vp31h+/+cB
0rJ7d6BboH2vXyuKmcrmj1wslZ76R9p7Qa+2gcTqQQZtZtf5XD8XAk7Y7X2yPp1JXRMO90vfnBLGZv
u62deDe2ny7TYu98JVsz6kg45/uuBuzCXD2zJs6VU06L5dVquT68Q7Zc53dIe8sB3heHGq8m4267gx
Z3TmrVCMfEx9rxclSLRRtZ77sfOEcJyTkGXM8A9hMMkhxHgKOA5SjkKQroR7Mb0r1SCovJb9XCLUhv
4dnWpezHfqWelX07E2Phl4YS4hlKY+WHACVRpi0ASZpBkFBOAOKMA5JRppcYJjT/OAdA27yM1gtv9n
d2fAFiHF725fvR6XoNlMF1wu1SYgLTjMN64ZOJ0Sw3PbSTqyjPLFDnSV9dmEPe6dEuFnvNqI7VRSPs
zWAu1gypgWgKk66iA2+PJ3DVQdrU5XtH9Y6oauW8KkYlpGPP1/mstRgHJzetFtFVR4Us3nyp7Pc3c0
AGa+dilLfQ4eukCBZSzJnhHDVFKdZ9U2kj0BOlSJQHGUEUA4ITDiDCCaB+hkEMKeMo97Mwyh6TIuNf
2v6iWRlzzq+Ev8KTO0oBC5iuYDbHIUOEInK9KhDIIDUCJttxQGAQsJs5P6nemYGD4cu2uSwq389IY5
wl5HiNkRPrbp5iY/LtU45F2XeV04itaHbQiO7XeLKu5e4Kg/sV8n4j1XpnjXgHjfXqToUPndf4W3kd
PNG8JiRAOc58wFPCQRhBBqKYJSCLSUBTP419Fv6PeW0Z9115TWGIfiT2j8R+xMQmS2JnhRLXsho/0a
wOSZ4SyBlIGWIA04SBOMgSoNu7EOI85XkSP35DVyn31nt76qMfpMFb6c+MqX0NSYQQ050rxxhEHOcg
jiIIQogY5djPdS37uHy1VBpDVbeC16cbKQ435sPkBsM0U5yxVWkjiu4yx9UB8lHg+czzgyu2aRsenm
904Rvve1MtvmQceaKMyxkkPNH9IQ5oBnjMfMAwjAFjUZJAP02DnDw+41ZKTpT7c1NIHbqFdfe0i9/D
uoeFOlygPm7qSjivN+27G4DTJwo4jjONap6AkIUQaJQpIIwFuoXAuvRQAn2fPz7gY1PpmN2J+T2txH
+qNDHPWBJQXVcjTAENcQpYEmJdaZIsif0MoQxfVprRQNpp63YtMJ8//f3L50//
PEB1scPyq2WJup3N3EmSiKJU93kJxBzgLApBzCkBnAQYpwmLdb4a7gwQ3+aOXtyNO0N/JuTQ1/
```

ZfFPRn+mwL/XoPiSmnuhmbUZoocmWswf3YuK/HRr4qhsOt5UhrG6jULg2Gl5PolYhxffn3dvAvAAAA
//8DAFBLAwQUAAYACAAAACEAerfzZxEGAABVHwAAIQAAAHBwdC9zbGlkZUxheW91dHMvc2xpZGVMYX
lvdXQ1LnhtbOxZ627bNhj9P2DvIHi/WYt3KmhS6DoMS5ugSR9Akeham26TZCdZUaCvtT1On2QkJcV2
4qROmhQBlj8WLZFH38dzePhJev3mositpWzarCr3J/CVPbFkmVRpVn7cn3w4jYCYWG0Xl2mcV6Xcn1
zKdvLm4OefXtd7bZ4expfVorMURtnuxfuTedfVe9Npm8xlEbevqlqW6tqsaoq4U3+bj9O0ic8VdpFP
kW2zaRFn5WQY3+wyvprNskQGVbIoZNn1II3M407F386zuh3R6l3Q6ka2CsaM3gypu6xVtt15dXpxel
4dnf0xsUznZqlOw8mByj85yVOrjAt1wq+KOm6ytirNlbY+baTUrXL5a1Of1MeNGfBuedxYWaoBhoGT
6XBh6Gb+lkvTmF4b/nFsxnsXs6bQRzUb1sX+RJF2qX+n+py86KykP5mszibzoy19k3m4pfd0vMF07a
Y6qz64m+mgMZ3TrMulpafHxHHYdmNEiybbn3yKIuTRMCIgUi1AbI8ALyQOiBAWIeKRjzD7rEdDtpc0
0vDyWzrqC7IbnBZZ0lRtNeteJVUxiGPUmKITkoFOHeWnCNvQhRCC0PEEgJGDAA4RBAGJmO0IgpzA/
jxMgIp5PJospkO+Q+IjEW19WCV/tlZZKaI0rz1vVz16MvWxno+a0nM09OsvmsZqlrdSLLDDhTDcYUY
hoptkQ5tCyuyBRYgRpQxf57KHrve6C69KL/XwM3U0Wov38rY76S5zaf7U+seE0SiK81ibgSzBh5P+r
t2Bn2fJn1ZXWTLNOutt3HaysUxmyi0Uir5vP4EGRZbpcdzE79fBfn8/
xFeb4MagTJx3yw1fyU2nf5zHiZxXeaoiQM9UecQm2GcIA4cIDmDgYeC7WAAMfSFcxlnk/
gDlabZ1QFov3yNAyASEvbxWClT641zwXoACYQeiXfVnxWUyr5T9n/
WQoxRNe5lDNcwq4ubQuFRWpsqxddMALN6pbcmMSuVM66v9W/kR0SvhbEzzCmUARCtAQjnSfXdCtW+i
aqgBFa9QHUhMBLugQnETVUMNqGSFCjGHZonvBGt6bsJqrAGWrsEKJEwMD4XVWAMsW8EiJJiZsIfCaq
wBlq/BcoJ3ZmwbrMYaYMUKVmPuTtkWWI01wDprsIzy76JMY/XttTVh3FnfRHW42srv79Z65Rqzbjfc
+iGOTEZH9quyU4lumDJ+pqYcCuQ6ka9KXI9QZcUEAiowAg5lDiURhpzCpzRlzfk8zmeDJfd2+UBLRt
SmNr9WFGxYMmaCUNX7+2qCJ1bd9buYCbz7LicyqcrUyuVS5jsgmlm+G/F0njW7Aw7b4F2AUbVouvnO
iGQHxGy2FfCxKy16a6VFnmul5aIo8EIM7MBGIPA5BY4KBzBmq50+dANBnR9VaekF/tcibpTuhzXel+
T3WeMMcmT2u9vrLoGhdoGXuuul7nqpu/5fdRe7q+6iz9WiPQc7URSCkCEBkKMfhqkbAWwzIhzMMXfC
p667Nm3Z7LgPtuVbaq81W36pvV5qr3uvbT6u7SDu5MbCZs90Yftu4Lq2T4ATUAFcjiDgXkCAes6yWe
hhL6D86WuvtOsrr7VHK9i/1r51gZu37juuwlme9tliTh212ULgRUSlHJEQuI6jUoZqB46IqjUp/jx+
SEgVh11WyCj7uGjk0aIzFG4oTCnFaovOz2VcXq3x7gDZCE9tMbXxSm0qhsfXmxj1FlWVdot1xfHnqj
jOkCdgBGyBKWCQuCBgPgYRj1wcQi8Qgj294mZds63Yh994yXof1T0u1c5I9UmepdJ6tyjOrhEuninh
NoPqGYgrd3GxAwLqCKBoRoBRGmHP9ygOf4DFtHmq5mwr5994i/Mgp3GjQHiqhgC+QxhgnPhAeJwop/
ECz7UDhAJy5TStprRU0e1qMF+//PPL1y//PoK7mMP45XOcddMatON5DkO+8IAHSQRI4HDgRoyCiGJC
fE+4Pg61dmpIbmpHndxNO3V1Lpu6ysznYWgP8lnGansXUFdqgvOBpV4iq2A17yc6fXXMm7dxfbQ0Gi
lMAeWbU7XWZd911UWnPn4OP/gPAAD//wMAUEsDBBQABgAIAAAAIQD5r5wJ3AMAABQMAAAhAAAAcHB0
L3NsaWRlTGF5b3V0cy9zbGlkZUxheW91dDYueG1szFbbbts4EH1fYP9B0cihqBoE
4/gJXoWKhuS9FuvEWB/tbu5/RLOqTEJG1SIA8OkBeRomaGZ+YcjvjVrq6N2oRsXi02boE
4/gJXoWKhuS9FuvEWB/tbu5/RLOqTEJG1SIA8OkBeRomaGZ+YcjvjVrq6N2oRsXi02boE
qrvrlf3+iqLItkbJuoo1fcdX9pGP9uv177+9GpZjU52zY7+XFsToxiVb2Tsph6XjjOWOt2w86wfewb
dtL1om4VVcO5VgnyB22zjYdQOnZXVnz/7iKf79dluXPO/Lfcs7OQURvGES8I+7ehhNtOEp0QbBRwij
vX+EJI8DZCtr2fCLrjnaljYVB1j07DVkX26ayupYCwtXysrSZurLOFwJztWsO/wphs1wKbTD28OlsO
pKBZgdbWf+MJvp1+6gJ85P7tdmypY3W9GqEWph3axsoOyono5a4zfSKqfF8m613F08YlvuikesHbOB
c29TldUE7mE62KQz1UGVR+M4H6VBtBf1yv5MKU4XBSWIwgwRNyUoLUiMKPajAoc0w37wRXl7wbIUXL
PyV2XU5QUPGG3rUvRjv5VnZd/
O0jAKAzI9MpOpUH6Ok8JN86BAhAY+8pI8RAvPjVGCcZLFbpTHLuw+FQAwm1Fn4cz5zokbIsbhvC8/
jlbXA1GK14m3W4uJTDUOu/
uKmu2mj3pyV+VZBfIm7auj2uQDjHqRLZtRbuSx4fplUA8NQwARDVMHlnfo/WYiV66zpi4/
WrK3eFVL6w0bJReW3h9ONERRCU5p6ii8qy6ZYO/uB/v73VyQQeM0oBwjh1+LwjeiyJnk1mXDSr7rmw
oQ4BeqjzSjCaZpihKShAhnRYjyLEtREecJIRn2fIyfXx+VhH77L2TCmq0CBofTm47lSfSyhaals/XD
RYxx5KGUEoJiSgqUxLGHQg9HASVukSz8L6YNVsChrFtO6+u94Bd7qSn8QXagFGtsZdZw1t02GLnGLv

YdN3Jc/05tgOH0eiNGb7Tvlc7vK85/oYoLQi9MfJwiL3J9hIMItJflMTSowgcEOMn99PkVt5Viktw/
eyagdEZ1xvcEqjst1QtD9aapK2693bcffiKcvFDCIwJHy/cSRMkiQJjSEPlxECISBPowLmgUPT/
hcG+Dmj3KOX6GTpPQPEr9IEJZTAIUhCRDURoS6DRpniZujnFObjvNqCjtAN1TG8y3r//98e3r/
yfoLnowNzdTdT2btZOmcYAzOKSpRygieRyihAYLRBc+/B/
SKMn8Qmln8MhD7cDi07Qz9J+4GPpaX249d5bPgTVATwj30xCkYn4Ik0bu0CriNyp/
GBvxhg0XBy2SVv/7M700KGFOpncmKndzm19/BwAA//8DAFBLAwQUAAYACAAAACEAlIYDFYUDAAD2CQ
AAIQAAAHBwdC9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQ3LnhtbMyW3W6kNhTH7yv1HRC9dsDGMDDa
yQoDrqpNN9Fm+wAOeDJo+arxzGa6irSv1T7OPkmPDSTbJJVykUh7g83hHPuc8/
tjePP2pm2cg1Rj3XcbF5/4riO7sq/q7nrj/vGRo9h1Ri26SjR9JzfuUY7u29Off3ozrMemOhPHfq8d
WKMb12Lj7rQe1p43ljvZivGkH2QHz7a9aoWGW3XtVUp8hrXbxiO+H3mtqDt3jlfPie+327qUeV/uW9
npaRElG6Eh/3FXD+Oy2vCc1QYlR1jGRv83JX0coNqrRnSfXMe6qQMYsHsKlZeXTeV0ogUDsx7GOA4f
lZRm1h1+VcPlcKGs7/vDhXLqysTOMa43P5jd7G13sBPvQfj1MhXrm61qzQgtcG42LpA6mqtnbPJGO+
VkLO+t5e78Cd9yVzzh7S0beN9taqqakntcDlnKyYWWzkUjSrnrm0oqxzTJpnQ26iW5vao37hfOCQsL
ThGHGaI+o4gVNEGcBHFBVjwjQXRronG0LpW0XH6rFn3h6BHTti5VP/ZbfVL27SyORWOAE9MZp0n4S8
RSjBnjiKWwcR4lBGVpUpgUco5Z4tOY3869gJyX0VbhzaXPPViYjMNZX34ana4HZgbxhPDOY+JqxmE3
a6rS8Eb9BZWIZmsSAw54IrA428k9gFkg+ob11dFsegWjNYp1M+pLfWykvRnMZQvStNUGqzAhJMaIcU
pRwmmB0iTBaIVJHHHqF2kY3C5Cr4ChrlvJ6+u9kud7bREqQA36hpNAdujdB8i71VkjRXenJX1KfBJ4
fuz5gWnX1DTIwXLvqguhxIcHq0wNHmydS1HeorT/11uw6I33vQaVfa848oMqrshCHPAoRCwjGOEkKl
DEEx+twjSnMQ4xy6LXV9xWq0lyf+6FgtYtqltiX0B1L4uaLqgvm7qSzvt9e/UAePCDAg9zugrh9UI5
DhIUBkA9zYIQkSD3IxLyAqf09YHDlxl69iRz8gonTcrzmAVRjLKERiha0QzFbEXhpGE5S/2ckJzenT
SjQdpBds89YL59/fuXb1//eYHTxQ7LR3rpup3N2mEsiUgWM8Qw5YjmyQql5u3lYUBpxmJAWRjtDJg+
1g4Yn6edof8s1dDX9vcF+7N8DqIxePCKrnyfxDOmSSP32Rrwl6Z+GBv1uxjOD1YksBlAzqxpMMKcXO
9dTO3L/9rpvwAAAP//AwBQSwMEFAAGAAgAAAAhAIgycve5BQAAdxcAACEAAABwcHQvc2xpZGVMYXlv
dXRzL3NsaWRlTGF5b3V0OC54bWzMWO1ym0YU/d+ZvgOj/zZvgOj/t6I/YAFT+wMn51OnMQTOw+AYWXR8
xMZvJa7ePkSXp3YW1JVizFsaf+Iy7o7OHu3sPZCy9frcrCWIq2y+vqeIJfmBNDVGmd5dXV8eTDRYyc
idH1SZUlRV2J48m16CavTn795WVz1BXZaXJdL3oDOKruKDmezPu+OZpOu3QuyqR7UTeigv9mdVsmPZ
y2V9OsTT4Bd1lMiWna0zLJq8k4vj1kfD2b5akI63RRiqofSFpRJD3k383zptNszSFsTSs6oFGjN1Pq
rxuYbX3558VqYihYu4QLeHICM0/Pi8yokhIuBHXVA4PxKe/nRpA0kklhuuaiFUJG1fL3tjlvzlo19O
3yrDXyTFKNFJPp+McIU6fVUgXTreFXOkyOVrO2lEdYEWN1PIHCXcvfqbwmVr2RDhfT26vp/
N0ObDqPdqCn+gbTtZvKWQ3J3Z0O0dO5yPtCGHKhVB6nXa8zWrT58eRzHBPfimKGYogQM32G/
Ii5KCbUiQiPA0LtL3I0to/SVqja/JFpjWH7Tl3LPG3rrp71L9K6HAWidQYlxWwsqczyM7VxEGBmI07
DEAWuS5DJCUG+y0nkY+44lH8ZFwBy1kc1i+k433HiuhBdc1qnHzujqqFQsq5D3W4QQzHlsZmPuurlG
o244U8V3K7yzhI71IX8VO2YxUGsm8WmLiWE8qGI2DbNEbFeyoG5OepXfp1dy9GXcIQSJlU6r+EJvBw
4i64/768LoeJlgceEMjF7D+Dub7jbLfsNQMZrAxv5o8a1MKhIpKWICn04H+7RnwRFnn40+toQWd4bb
5KuF62h1gY8B0gk4VACxSKq7CxpE5nBDdnr92MSjZqfnpea6v2CpVqw+hE+K5JUzOsigyTIM5Uvtmx
qUs6QSQKCnAgTRIjDEbctn7uOH1qR+5TyzbPVLeRw5VrYoXiUrutwRqxN6doYnkFz9B/mcGoPiEOk+
xN6VSG5iyXOOlYDIKQ7sGwdqwEQsh1Ycx2rARBa+7AaAKG9D6sBEPJ9WA2A0NmH1QAI3X3YAbDLDxr
59C6Lm+3nx/1BikbZQ7fhD4MHbN9FafX+u5yLtK4yoxBLURzASPYzXszz9nBCup8wrhct9BeHMrIDG
PPZTsLHdll20xbIsq1bLH2mFuu41GYxo8jCboAC04uQ5Zkh8kLPM3nEPDv0nr5DkN42UQ/
WPClmMrXVKL2HtgzEtPjoVd/pGaiDsQXonzReo0zaU9VN5lUGG6sM1ajFW3iFUKPWfEM2Kt/15ZFq7
HYO49vw4y3vHvlczCTqML6NfWPL30c+TLmaxmGE920CmtAhjtyDHkC4tVOMhNAk2BL2AMKt7UQTcqZ
21AcQbu05I6FkO7wo921MmtC2+AOL8r/tXj/mrZb21jDpxYa3sufavppxFJrQuYY+gR7Wjj3EfUJRb
AcBpT5nnLGn99asv+OseKj+d61VvZPfa4DqRGlkVmTjuya3XHjwMPJjxpAbswh5rosRx/
A0xsyMPIt+0Z8aMqhhn5cizq8WrXi36FUJN6QFu7DRlX1QiKS6UWB/QkxCp6YzNemtoiCHx9/Lba23

uK6lltcVZz1TxbnMsjD3OWIUYxSz0EbwCuWg2IHG3Yl9D1zi6RU369tBcn8tkhaWTqtuz5vUj6jucU
vNdanPizwTxttFeblVcPuZFtwPY4ot7iHmRD5yScQQc32KPGKbUWCaPqb+0xe8KzJYs50139PEPchp
vDh0fGo7KHCZjWzOAuSAm4LT+KEPzSshIbtxmk6WtILsDjWYb1//
+e3b138fwV3UQX8X1auuolE7vu/aJHB85GMWIxa6HHmxbaHYoowFvuMFNJLaaTC7qx24eJh2mvqTaJ
s6Vx+QsTnKZ5nI90VmQcfMTHes0iCR22Rl3c/l9OFYtG+S5t1SaaRU23ugLjVSlwP0FiKnrj+Yn/
wHAAD//wMAUEsDBBQABgAIAAAAIQBr9+RZiAUAACYXAAAhAAAAcHB0L3NsaWRlTGF5b3V0cy9zbGlk
ZUxheW91dDkueG1szFjrbtQ4GP2/0r5DlP1tJnYcXypalJtXKwpUtDxASDKdiNw28QzTRUi81u7j8C
RrO8lcaAtDGVD/
TBzP55Pv8zk+dvL02boqrVXe9UVTn9rwiWNbeZ02WVFfn9pvrgRgttXLpM6SsqnzU/
sm7+1nZ7//9rQ96cvsPLlpltJSGHV/kpzaCynbk9msTxd5lfRPmjav1X/
zpqsSqW6761nWJe8VdlXOkOOQWZUUtT2O7w4Z38znRZpHTbqs8loOIF1eJlLl3y+Ktp/Q2kPQ2i7vF
YwZvZ+SvGlVtW2RXq1ty4R1K9UB7TNVeXpZZladVKrjokjlssut94VcWGHSaiQT07dXXZ7rVr36s2s
v24vODH25uuisItNQI4Q9G/8Yw8xtvTKN2RfDr6dmcrKed5W+qhmx1qe2Iu5G/850X76WVjp0ptved
PHqjth0Ed8RPZseMNt5qK5qSO52OWgq56qQZW7piTJ5nPdyymjZFaf2ByFQ4MUCA6FaADsBBkGMORD
IZTGiIkQu+ahHQ3KSdrnh5q9s0hgkt3itirRr+mYun6RNNQpk0pmiFOKRUp3lB0h8EUCKAOGRAHFEA
+B7GAIoREg49wlj8OM4ASrn6WqqmI31joVPRPTteZO+6626UURpXgfeNhEDmfraLkZdST1HY9zwp2l
sZ/lOipnLKWOGO+xRJdZ9sl3uIuTSgURIHGeM2KVyQG5P5Dposhs9+q26KgqTOl00agW+HTDLXl7Km
zI37VUJx4SyfP5aBff/qKdt0TcBur0zsNU/ZlynBpWJtpS8Bm8uh2fIs7As0neWbKw8K6T1Iull3ll
mbpTnKBANOFBgUPI6u0i6RGewAXv+ekyiNfVNdZlSvy5YdxLstIQvyiTNF02ZqSTQI5Uv9iPqOK4DC
GEuCHDEAMNKyGEcopAHzGck+vnyVbao81lvow8XsQeZC0cVc0Yx8vZVTNTi1NIyKsaMumSIOETF90n
XqpLu3PhbUWfK73XTjFq+VJuaGfUNZZsm2kKN6+8gPMR28TTIiOdu8TjE+GA8HbnB0yAjHt7iQZdqB
zgQ0NkF1CgjoLcDyBDTdTwAUKOMgGQLiBAjOuwBgBplBKQ7gBQb5h4AqFFGQLYF1GiHk7IHqFFGQL4
DSDz6QFI0yt3+elxTxJtdXK/HXUd0H6sjilBwKJQZ+p4LaAA5CHEsgI89zEUIsY/
pz3dE7T+24W2RlPPRHNGP7PDI8ei40O/
Z4l0Goaeif6k5Glc5ojnCPTP7cXOEe+Z9BHOExzbHfcAjmOM+4BHMcR/
wCOa4D3gEc9wHvN8cNbwK2LzrfP9hVK88cxbt9w6jD/FWb/
LWKJH7p038SL1VHTVpBFkAeCwEEH7gKG/FDoCRE1DXJ64b/oLTZiZvOSsc2L/XWs0r9FcN0NwYjczV
u7yp1qUeVwsPgkBgDLjAMfA5h4BCtRoFdmK1wXycvgxkikNZVLkortW7w6ulNBTuSUvtwlZfybDMk3
qjQHmGHOTOHDZz3K2iVA7H38vJpDfRNFrLu4rzHqniQicm2CPq9VzEGBBGIAh9pGRHMEcOhgH145+v
uLnsBsn9vUw6NXWT6r7xtvM9qjsu1XSi+rIsstx6uazefkE4eaSEx5Sw2CccOCKiwIVILT9XRIByjw
eIBgzTX3B868tMzdmdnH/jEPcgp/FFxAKXMBByTAChOAQsoFg5TRAFvhMhFOGN0/Sa0lpld6jBfP70
7x+fP/13BHcxl+kz5jTrpjVqJwg4QaHaIAKIBcARp8AXxAPCczEOA+aHbqy100J8Wzuq8zDttM37vG
ubwnzvhc4on1Wi9nWOKMfcIWRkaZDINlnN+6UuX13L7kXSvloZjVRmew9NV6t1OYRuQ3Tp0/
fts/8BAAD//wMAUEsDBBQABgAIAAAAIQAaQQcRcQQAAL4PAAAiAAAAcHB0L3NsaWRlTGF5b3V0cy9z
bGlkZUxheW91dDEwLnhtbMxX247bNhB9L9L9B0LvGkmUZMuGvUHsTYoCaRsgafpMS5RNRCJVkr5s
9qPyHf2xDinJ9mbTBdpFgb4YnDlzZjgzHJ4p+dvL02boqrVXe9UVTn9rwiWNbeZ02WVFfn9pvrgRgttXLpM6SsqnzU/
9qPydf0iElrfeWxim8wL6IFDU8nJlzZiS9fLWrK2PLRV+2zdx0XtimwZu8Lcrmem6+u6IoNI1esqZg
VdvwubnnvfnqOefXnazvirO2b7dSAMwmn7G5uZaym5mWX2+5jXrX7Qdb+DZqhU1k3Arrq1CsI+AXV
eWa9u+VbOyMcf94pj97WpV5nzR5puaN3IAEbxiEvzv12XXT2jdMWid4D3A6N13XZL7DqKFxMirnWlo
O7GFFcc8g9DzZVUYDath4aqUFTcgQcYfYFzmrDKu+E5qs767EpyrWbP9VXTL7lLo3W+2l8IoC4U2op
jW+GA007fNVk+se9uvpymb7VaiViNkxdjNTSBvr66WWgMnjHxYzA+r+friEdt8nT1ibU0HWLcOVVEN
zj0Mx53CGZKicqX9OO/l5NFGlHPzE6Vu4mWUIAozROyEoCQjEaIuDjM3oKmL/
c9qt+PPcsE1P78Vk84c/wG3dZmLtm9X8kXe1qNIJq0BrQ4ZaVVefnJxEOMgilFGPQf5cEWpb6fIx7E
f2DimmNLPYwLA52nUUVhjvGPgExF9d97mH3qjaYEoxevA243FQKYau/WoLalyNNoND/
XkkOVRBXKXtMVeHfIeRr3IZlUvl3JfcX3TqYt2QwARFVOlyxv0bjmQK8/Sqsw/

```
GLI1eFFK4zXrJReGPh9qG1BUgEOYGoU3xSUT7O1tsN/fjgnptJ+TU9Ykh2+LAk+iuFMfxmXFcr5uqw
JccZ+pUJIozWLPs9GCxBS5WUxQZNMIZTYNkiyMg8DHTy8URbtptKKETjW0JOXe7rD5R9Sjej2gcKbY
GPTxUEudYn5b3bSHH9eWYlBLq7+jrUE/90/Rcfz3KUuet9BiK77l1RGI7vcRr9alOB4Qfx+Qthsh10
cjkiMQy9WjgKeuUDJV6IJJfqcw8TMtTOqktudhqEnXgb6duBTFLjjjJlGQ2R7NAid5+sIsoBD7vyAS
Vq2mkhxenCfp6Cv4xtDR4sCLXDd0UEIJdCBKMhRHkYMCxw19SmzoUdCGJqeAQ1nWnJbXG8EvNupL5J
7CQClGX8u04qy5qXF55toutuzQsvFBbeDD6fXmTXqjbau6xW3FkWequDi1I5JFFGV4ESOa2OFAROB4
aZYmizjD3tMrbiXFILk/N0xA6ibV/Z83wTdUd1qq/YnqZVUW3Hizqd/fI9x7poRjL4WvQzdGMc0WcL
C9QARnDsJp5EVZhBMSpU9POPxjQc4e5Vy/407caWK6CBPshyiNiI/8gKQoTAICKk8WSWwvXHdBbjpN
ryhtwLtjG8zXL3//8vXLPyfoLnqY/
q2mrOvZqJ0kiXw3DROUOIQisogC4NH3EPUwIWkSxinOlHY6hzzUDiwep52u/chF15b6R9SxR/
lsGbzePTeKPBtHI0mDQg6+KtqXKnoYK/GadRdbLZFafz+leqlTshxMDyYq8um/++xfAAAA//8DAFBL
AwQUAAYACAAAACEAFVUf9KYEAACfEAAAIgAAAHBwdC9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQxMS
54bWzMWNlu3DYUfS/QfxDUZ0biIokyMg60sSjqxEbG6bsicTxCtJXiTMYNDOS32s/
Jl5SkJC9jx5kATuCXIUVdXp17z7lc5uWrXVNbWy6GqmsXNnzh2hZvi66s2ouF/
e6cAWpbg8zbMq+7li/sSz7Yr45//
eVlfzTU5Ul+2W2kpXy0w1G+sNdS9keOMxRr3uTDi67nrXq36kSTS/
UoLpxS5B+V76Z2kOv6TpNXrT3NF4fM71arquBpV2wa3srRieB1LhX+YV31w+ytP8RbL/
ig3JjZdyHJy15FqxIjzytZ86gtz3e2ZezFVr2B9rFKQbGsS6vNGzXwlzKtiry2jL2lMmad8500ZkN/
LjjXvXb7u+iX/Zkws99sz4RVldrb5MV2pheTmXlst6bj7E2/mLv50W4lGt2q7Fi7ha1IvNS/jh5TIK
xiHCxuRov16QO2xTp7wNqZP+Dc+qiOagR3Pxw0h7OXFJ00A+hkkDO0jagW9ifGUOxljACmeoC4MQFx
RkLAEKYZCliCsH+lZ0P/qBDcEPZHOQsP+vfIbqpCdEO3ki+KrplUM4tP8QzJxLOG+4mEmZsxFgIEMw
+QNEwACdwY4JSlNIKhi2J2NWVCYZ5bE4UzBT5lYGZk6E+64sNgtZ1iTBM8EnhtMbKq2349iU3qHNlW
JyolyVF706zR1HRukv8g8zRAJHRHTrHvQeTdFQHyETXvNbkehZBiuk/x6Lo/kru4Ky/17PeqVdRqRA
ub55rV0W09yKW8rLl56PWPASWUcZ3rFYS34N1ytJXHSV0VHyzZWbyspPU6HyQXlolaLTHKi0YxJtd4
4W15lov87W1nf76d0PYG6gzRoH5ck/i+JnVSzuq84OuuLhUU9EzlmcUhSRMWABi5KSDQJ4CEKQU4jB
FJWEip7/14eWoR7KlTwdvdTP4OlWKKHhFpEGCCf6RIey2pbX297H2/aDVUo9nhjmhHYe5/xSTo8a8s
edGpHaPmW14f4BF92+P5uhKHO8Tfdsi6jZDrgz2SAzxWqwcdPnXpk7n001zyOxWPn2nFpyzGzAsj4B
GSgcgnDLgYJiBhqRdDL/JChn58xZeqwod/VCR5vZprfTwQfLXYzXllvya/UoUrdXYy0eLACxGiEMSM
EBAyHXIYQhBARH1G3Czy8NV8JCsVh7JqOKsuNoKfbvQJa09hSinW0Mik5nl7XePyGLkIOy51XHyjNo
Xh6fXmzXpjXadXi9uKI89UcZRiGCVBAGLoQ7XRRARg5oeAulECA5jC5GcobiXFKLm/N7lQqZtV940t
5ntU97RU+zPVy7oqufVm07zfI9x7rmde16dhEgYgyVAMIHZV+aEsBEGIIQm9iPrBTyBc3SFVzh7k3O
xxT7zSROo8H2OfgiQkPvADkgAaB0StNHEaq8MVQim5XmkGTWmr0B26wHz5/O9vXz7/9wSri2nmO+Oc
ddObtBPHoY8SGqtyVXuDurEEIGK+B5iHCUliGiU409rpIbmvHTV4mHb67iMXfVeZizZ0J/
lsc7W9U8+nCOJgJnmUyA1YzftSh6/aWrzO+9Ot0UhjDlCJGeq1LkfTGxMd+vzHwvH/AAAA//8DAFBL
AwQUAAYACAAAACEA1dGS8bwAAAA3AQAALAAAAHBwdC9zbGlkZUxheW91dHMvX3JlbHMvc2xpZGVMYX
lvdXQ2LnhtbC5yZWxzjM+9CsIwEAfwXfAdwu9N9/f5cs39No3hSYhe8RwcFF9AGO5NoG2yTkoujbm9GCg+N9/f5c
s39No3hSYhe8hlpWIMibYJ3vNdyux9UWBGf0FsfgScObGPbtctFcaMRcjnhwkUVRPGsYco47pdgMNC
HLEMmXSRfShLmUqVcRzR17Uuuq2qj0bUA7M8XJakgnW4O4viP9Y4euc4YOwTwm8vlHhOLRWTojZ0qF
xdRT1iDld3+2VMsSAapt1Ozd9gMAAP//AwBQSwMEFAAGAAgAAAAhANXRkvG8AAAANwEAACwAAABwcH
Qvc2xpZGVMYXlvdXRzL19yZWxzL3NsaWRlTGF5b3V0Ny54bWwucmVsc4zPvQrCMBAH8F3wHcLvTf/n
iDR1EcHBRfQBjuTaBtsk5KLo25vRgoPjff3+XLN/4UmIXvIZaViMibYJ3vNdyux9UWBGf0FsfgScObGPbtctFcaMRcjnhwkUVRPGsYco47pdgMNC
nDmxj27XLRXGjEXI54cJFFUTxrGHKOO6XYDDQhyxDJl0kX0oS5lKlXEc0de1Lrqtqo9G1AOzPFyWpI
J1uDuL4j/WOHrnOGDsE8JvL5R4Ti0Vk6I2dKhcXUU9Yg5Xd/tlTLEgGqbdTs3fYDAAD//wMAUEsDBB
QABgAIAAAAIQDV0ZLxvAAAADcBAAAtAAAAcHB0L3NsaWRlTGF5b3V0cy9fcmVscy9zbGlkZUxheW91
```

dDExLnhtbC5yZWxzjM+9CsIwEAfwXfAdwu0mrYOINHURwcFF9AGO5NoG2yTkoujbm9GCg+N9/f5cs3
9No3hSYhe8hlpWIMibYJ3vNdyux9UWBGf0FsfgScObGPbtctFcaMRcjnhwkUVRPGsYco47pdgMNCHL
EMmXSRfShLmUqVcRzR17Uuuq2qj0bUA7M8XJakgnW4O4viP9Y4euc4YOwTwm8vlHhOLRWTojZ0qFxd
RT1iDld3+2VMsSAapt1Ozd9gMAAP//AwBQSwMEFAAGAAgAAAAhANXRkvG8AAAANwEAAC0AAABwcHQv
c2xpZGVMYXlvdXRzL19yZWxzL3NsaWRlTGF5b3V0MTAueG1sLnJlbHOMz70KwjAQB/
Bd8B3C7Satg4g0dRHBwUX0AY7k2gbbJOSi6Nub0YKD4339/
lyzf02jeFJiF7yGWlYgyJtgne813K7H1RYEZ/QWx+BJw5sY9u1y0VxoxFyOeHCRRVE8axhyjjul2Aw
0IcsQyZdJF9KEuZSpVxHNHXtS66raqPRtQDszxclqSCdbg7i+I/1jh65zhg7BPCby+UeE4tFZOiNnS
oXF1FPWIOV3f7ZUyxIBqm3U7N32AwAA//8DAFBLAwQUAAYACAAAACEA1dGS8bwAAAA3AQAALAAAAHB
wdC9zbGlkZUxheW91dHMvX3JlbHMvc2xpZGVMYXlvdXQ4LnhtbC5yZWxzjM+9CsIwEAfwXfAdwu0mr
YOINHURwcFF9AGO5NoG2yTkoujbm9GCg+N9/f5cs39No3hSYhe8hlpWIMibYJ3vNdyux9UWBGf0Fsf
gScObGPbtctFcaMRcjnhwkUVRPGsYco47pdgMNCHLEMmXSRfShLmUqVcRzR17Uuuq2qj0bUA7M8XJa
kgnW4O4viP9Y4euc4YOwTwm8vlHhOLRWTojZ0qFxdRT1iDld3+2VMsSAapt1Ozd9gMAAP//AwBQSwM
EFAAGAAgAAAAhAGmiXyEVAQAAxwcAACwAAABwcHQvc2xpZGVNYXN0ZXJzL19yZWxzL3NsaWRlTWFzd
GVyMS54bWwucmVsc8TVTWrDMBAF4H2hdzCzjyU7iZOUyNmEQqCrkh5AWOMfaktGUkp9+4qWQgxhaCG
gjcCS9ebjbbQ/
fA598oHWdUYLyFIOCerKqE43At7Oz4stJM5LrWRvNAqY0MGhfHzYv2Ivfbjk2m50SUjRTkDr/
fjEmKtaHKRLzYg6nNTGDtKHT9uwUVbvskGWc14we50B5SwzOSkB9qTC/
PM04l+yTV13FR5NdRlQ+xsjmOs7hS9yMhcfYqVt0AtI0+v92U/bNIwAdlu2jClbUrJNTNmGkmX5PWk
+3MUZ6nvnZ80ox10Z/20oJxuKKSM7K2LKCrKzuKWRra1j0tZkazxqa5yyrWLSVpRsF1O2+5Wx2fNbf
gEAAP//AwBQSwMEFAAGAAgAAAAhAMNEBGoTCAAAbDYAACEAAABwcHQvc2xpZGVNYXN0ZXJzL3NsaWR
lTWFzdGVyMS54bWzsWv9u47gR/r30CIJ0/
KkYne2abByrTK8vJyZP5ykViBaTSClhktqpKNzFvWmD+c/PUv7+thU2Q/0UYwboCNshnSkbkQoh4OB
k26YEvavKtqVsK9ecWXVMBHfjnIOP0VbC+LAbZtb7CkeWl2z/PHPF/
N53nKJlW6WrJStEY4K6gA/5tFXjfaWv0YazVnDZhRT2+5dALzS8+LTF5nl+3/
T2xu5NkNRMm2EWjQobLM4oIb17QYmbNLZA5O3g865W4kH27qC86YHJXXP/
L6vD7j6g0fr8842ASTplHSJcRXGlA3OjX1sbxWg8Gdxy/1kA5v5nwprxAeAzwEFG/
l/4GUsRthpK0w3UjTxc8P6KaL6QPaA/2CQe+lclatc/eng/V0LnJRMOOsoClbVEUGudJGDnw6bYT2b
sXzkflbkuDInSbESmBkETsiVjQloZVgJ5hiP4mx432RTyNvmHKmQPuQ6eRD3j3Al3nKq6aai3dptew
yRycgYI1Il37S4998fxwQjHxrMk2QNY0jx3ISz7PGeIqmGEdBGJMvXTDAZ31Vsxh0c++CoEFp6tMqv
WqMsgLQJMYthmuNFlh5rReGuK0hXkLGq9Nrb6rBJuIPwh04AaSuwtHxXITdbeCR7SLXkwoSUeRg1/
WcLVzpsOaN+JFVS0MORiZnqVBA0WuYYauqVZRPrSf1UNxEVXYrNWdwBfihlsDzi4p/
No3iQ9mMzBARAu8W6gNxfQwfeP/
ObOuOKOKqUPlHyxTsjMxUcOVLCcwdr0Q1zzuP2lfKW0UjzsVtwdS8a/lPiTk4VFBZylhp/XLehkWcx
EWeXhmiMliWC6MrYir0UOvAirTdIqyssDI7o5x+6hv756cugLUKhw6Disx+bjhrbkh8+tTAB0qNBLv
ID6cT4ALwI0JJYuE4wlbiTyMn9F17nHgvTw0JtnRIJvT/
wxAUYNfbTxHiuMhxgsOnyJNZUcscvlbPKuHTWSIjpkjSbLGkZcLdtyig9r/
lnKVVmRkFu2bFIyzi71u8WOT88QZVIdxvMKlWXCwebZE8wmI+f9Dgc9caomvNhIrtNuwcaK3Bse/
FHplaHh5PLTeBEfbGE2sajJFNiGtP0J/QhjMBq97PMBNazLuaoxLvf605ngMd172zHsM+cZSCLDmbv
v2KmrKaji4yanxdIMkDWlzCbqJQzmZsLrNZhhPJ6SpIqiLPkrwoHlhFi5t2iSjyUrQS6DC2Xoeuldt
PGzsD/SY17Bxpxz0HFVPnRdYmm+O7IcYBsqKEECuUyTYOQ2T5CAdeQuzp2HW+mDongEIiX7Ikv1xx9
vOqhWKL4EBUo1mKuGC0XJdYcYJt7AzsYGA7G7LP5d7iuenuaronVSWLdZ/
w5EAJH3uBE7rh2Bon08iKgrEHXMeRhXzfRpjAXmsavDzh55DNKkX/vaIcQteRvl0lP4X0xHYCtY7Yx

XqCEAneMuv1av3weP+8ZPM02c7BF2Z8XC1ndyjnHup63sVJiOOJ5cfx2CKJjawgdqZWZDsJ8SeB447
/BMo1RQYxe4h1agH1tFbrIXsv6958rz1Uzq177TiZBJHjBVYcEs/
yfBJbQeQT6LXRJBrbE4wnZN1rG0mpErLjsS3229f//u3b19+fob+qiz6t01mvRh13oyj0cBxAr0Iks
cgk9KF1ea6VuA4hMTQwoJLkbo3Ife6C8HHcratfGa+rXJ1xIrujr4IIBXbgY8+1NU9ajm68lcTrzi7
Tgv9Ea2N2iWA5JhDE9wZG2RWMZpdYyrCUYSmDEU1TVgrQ6AZagrVkreNoiaMlREuIlrha4mqJpyVQP
BdFXl5BMOTFNOZV8Y9WoEftSShUiVN6W63Eh6xDoidpzxoR8UngeCQE7gylhH/
IVAHaretKnmldtbLfo4t6umpBsEcX93QVPHt0nZ6uqkZ7dElP1/uOrtvT9b+j6/V01XnHHl2/
pxt+RzfoY6FYukd5CzjdOu4DL25UaWnUWB7W7dx2GFCdLujs/HNXYduqqkoqo6dlxK/Uibv81qDsPs
KtBRQIaJdnqzIV8r6yXJ7Xadvg0rO0q5GhvamRfYVInvlvq65L6frubPWxKtsTmV61bp28Ylx+CfPY
yt2Z7mupKakiOod1yMj8+/JfViG6Xkjv3GC0O/Rv7txIm872g1V+O/
q16nv3oFhSfgoQ43YxnJdQziGolhYcDlKi6apqr+/1wEoq6Iyb6Ix5TsHrmpZVAx9tbEew8CBw1X/
A1DoX6SKhy7yQiw0QpAvKGybW/Wq2ikGixCPz29f/tNJeOmC1Q3iJdCh3pUO5Kx3K/
emghngDuRe4yvlXALl7SIi/WAF4RsQlzB3izgZx2NA6MuZHyJ8IuQragUMuce4gJz3IAV61nTpC/
iTI0Wuo6xLnDnK318pt11cwHiF/e5BLnDvIvR7kLiKvZfl2hPyJkEucO8j9HuSh33p/
hPztQS5x7iAPNpA7BMugHyF/i5BLnDvIwx7kQeAdl29vFHKJc/vT0825TD2sxILx9SkNPHHWJkY3u/
uH4xuV7SOdF0mS1xbjh48+1Bc4x/jsPCjQQTjGZ8eu2vHlxvoYoF17UBTgQHl/
DNCOHZtq48cA7d7f6N8BHAO0YzcA7h6L9L61s+f6xyK9vdLsLy7VLyr0F7Xt97jtT8dP/gAAAP//Aw
BQSwMEFAAGAAgAAAAhANXRkvG8AAAANwEAACwAAABwcHQvc2xpZGVMYXlvdXRzL19yZWxzL3NsaWRl
TGF5b3V0MS54bWwucmVsc4zPvQrCMBAH8F3wHcLtJq2DiDR1EcHBRfQBjuTaBtsk5KLo25vRgoPjff
3+XLN/TaN4UmIXaviDIm2Cd7zXcrsfVFgRn9BbH4EnDmxj27XLRXGjEXI54cJFFUTxrGHKOO6XY
DDQhyxDJl0kX0oS5lKlXEc0de1Lrqtqo9G1AOzPFyWpIJ1uDuL4j/
WOHrnOGDsE8JvL5R4Ti0Vk6I2dKhcXUU9Yg5Xd/tlTLEgGqbdTs3fYDAAD//wMAUEsDBBQABgAIAAA
AIQDV0ZLxvAAAADcBAAAsAAAAcHB0L3NsaWRlTGF5b3V0cy9fcmVscy9zbGlkZUxheW91dDIueG1sL
nJlbHOMz70KwjAQB/Bd8B3C7Satg4g0dRHBBwUX0AY7k2gbbJOSi6Nub0YKD4339/
lyzf02jeFJiF7yGWlYgyJtgne813K7H1RYEZ/QWx+BJw5sY9u1y0VxoxFyOeHCRRVE8axhyjjul2Aw
0IcsQyZdJF9KEuZSpVxHNHXtS66raqPRtQDszxclqSCdbg7i+I/1jh65zhg7BPCby+UeE4tFZOiNnS
oXF1FPWIOV3f7ZUyxIBqm3U7N32AwAA//8DAFBLAwQUAAYACAAAACEA1dGS8bwAAAA3AQAALAAAAHB
wdC9zbGlkZUxheW91dHMvX3JlbHMvc2xpZGVMYXlvdXQzLnhtbC5yZWxzjM+9CsIwEAfwfwXfAdwu0mr
YOINHURwcFF9AGO5NoG2yTkoujbm9GcoMRcjnhwkUVRPGsYco47pdgMNCHLEMmXSRfShLmUqVcRzR17Uuuq2qj0bUA7M8XJa
kgnW4O4viP9Y4euc4YOwTwm8vlHhOLRWToj0ZGj0Z8eu2vHlxvoYoF17UBTgQHl/
lyzf02jeFJiF7yGWlYgyJtgne813K7H1RYEZ/QWx+BJw5sY9u1y0VxoxFyOeHCRRVE8axhyjjul2Aw
0IcsQyZdJF9KEuZSpVxHNHXtS66raqPRtQDszxclqSCdbg7i+I/1jh65zhg7BPCby+UeE4tFZOiNnS
oXF1FPWIOV3f7ZUyxIBqm3U7N32AwAA//8DAFBLAwQUAAYACAAAACEA1dGS8bwAAAA3AQAALAAAAHB
wdC9zbGlkZUxheW91dHMvX3JlbHMvc2xpZGVMYXlvdXQzLnhtbC5yZWxzjM+9CsIwEAfwXfAdwu0mr
YOINHURwcFF9AGO5NoG2yTkoujbm9GCg+N9s39No3hSYhe8hlpWIMibYJ3vNdyux9UWBGf0FsfgScOb
gScObGPbtctFcaMRcjnhwkUVRPGsYco47pdgMNCHLEMmXSRfShLmUqVcRzR17Uuuq2qj0bUA7M8XJa
kgnW4O4viP9Y4euc4YOwTwm8vlHhOLRWTojZ0qFxdRT1iDld3+2VMsSAapt1Ozd9gMAAP//AwBQSwM
EFAAGAAgAAAAhANXRkvG8AAAANwEAACwAAABwcHQvc2xpZGVMYXlvdXRzL19yZWxzL3NsaWRlTGF5b
3V0NC54bWwucmVsc4zPvQrCMBAH8F3wHcLtJq2DiDR1EcHBRfQBjuTaBtsk5KLo25vRgoPjff3+XLN/TaN4UmIXaviDIm2Cd7zXcrsfVFgRn9BbH4EnDmxj27XLRXGjEXI54cJFFUTxrGHKOO6XYDDQhy
xDJl0kX0oS5lKlXEc0de1Lrqtqo9G1AOzPFyWpIJ1uDuL4j/
WOHrnOGDsE8JvL5R4Ti0Vk6I2dKhcXUU9Yg5Xd/tlTLEgGqbdTs3fYDAAD//wMAUEsDBBQABgAIAAA
AIQDV0ZLxvAAAADcBAAAsAAAAcHB0L3NsaWRlTGF5b3V0cy9fcmVscy9zbGlkZUxheW91dDUueG1sL
nJlbHOMz70KwjAQB/Bd8B3C7Satg4g0dRHBBwUX0AY7k2gbbJOSi6Nub0YKD4339/
lyzf02jeFJiF7yGWlYgyJtgne813K7H1RYEZ/QWx+BJw5sY9u1y0VxoxFyOeHCRRVE8axhyjjul2Aw
0IcsQyZdJF9KEuZSpVxHNHXtS66raqPRtQDszxclqSCdbg7i+I/1jh65zhg7BPCby+UeE4tFZOiNnS
oXF1FPWIOV3f7ZUyxIBqm3U7N32AwAA//8DAFBLAwQUAAYACAAAACEAe0O8XcQGAADPIAAAFAAAAHB
wdC90aGVtZS90aGVtZTEueG1s7FnNixs3FL8X+j8Mc3f8NeOPECfYYYzubZDdZspuHLW2PKNYMzKSv
BsTAiU59VIopKWXUKWXQm89lNJAAw299I8I8wLF3ZBXReRjjGDsgn4jzqUJGUs/Pls
xF4QlHbd6ruI6OBmxMUnCjnvrcFhquY6rW4ZcLCcdfqv7ZBXReRjjGDsgn4jzquJGUs/Pls

hjBMBLn2Awn8GzCeIwkdHlYHnN0AnpjWq5VKo1yjEjiOgmKQe2NyYSMsHOoVLoXV8oHFL4SKdTAiPI
DpRobEho7nlbVj1iIgHLnGNGOC/
OM2ckhviddhyIh4UHHreiPW754obwWorJANic31J+l3FJgPK1pOR4erQU9z/
ca3bV+DaByGzdoDhqDxlqfBqDRCFaacjF1NmuBt8TmQGnTorvf7NerBj6nv76F7/rqz8BrUNr0tvDD
YZDZMAdKm/4W3u+1e31TvwalzcYWvlnp9r2mgdegiJJkuoWu+I16sFrtGjJhdMcKb/
vesFlbwjNUORddqXwii2ItRncZHwJAOxdJkjhyMcMTNAJcgCg54sTZJWEEgTdDCRMwXKlVhpU6fKs/
T7e0R9F5jHLS6dBIbA0pPo4YcTKTHfcqaHVzkBfPnj1/+PT5w9+fP3r0/
OGvy7m35XZQEublXv30zT8/fOn8/duPrx5/a8eLPP7lL1+9/OPP/1IvDVrfPXn59MmL77/
+6+fHFniXo6M8/JDEWDjX8Ylzk8WwQMsE+Ii/ncRhhEheopuEAiVIyVjQAxkZ6OsLRJEF18OmHW9zS
Bc24OX5XYPwQcTnkliA16LYAO4xRnuMW9d0Tc2Vt8I8Ce2T83kedxOhY9vcwYaXB/
MZxD2xqQwibNDcp+ByFOIES0c9Y1OMLWJ3CDHsukdGnAk2kc4d4vQQsZrkkBwZ0ZQJ7ZAY/LKwEQR/
G7bZu+30GLWp7+NjEwl7A1GbSkwNM15Gc4liK2MU0zxyF8nIRvJgwUeGwYUET4eYMmcwxkLYZG7whU
H3GqQZu9v36CI2kVySqQ25ixjLI/tsGkQonlk5kyTKY6+IKYQocvaZtJJg5g5RffADSgrdfZtgw92v
39u3IA3ZA0Q9mXPblsDM3I8LOkHYprzLYyPFdjmxRkdvHhqhvYsxRSdojLFz64oNz2aGzTPSVyPIKj
vYZpuryIxV1U+wgFpJFTcWxxJhhOwBDlkBn73FRuJZoCRGvEjz9akZMgM46mJrvNLR1EilhKtNaydx
Q8TG+gq17kfICCvVF/Z4XXDDf2+yx0Dm7jvI4LeWgcT+xrY5RNSYIAuYQwRVhi3dgojh/kxEbSctNr
fKTcxNm7mhvFH0xCR5bQW0Ufv4H6f2+WBVz+nXO0UpZbPKKcJt1jYB42Py6Zc2fTRP9jGcJmeVzVll
83+sbIr281k9c1bPnNUzH62eyUoYfRG0uu7RWuLCu58JofRALijeFbr4EbD3x0MY1B0ttL5qmkXQXE
5n4EKOdNvhTH5BZHQQoRlMU9UzhGKpOhTOjAkon/SwVbcuv+bxHhuno9Xq6nYTBJDMxqH8Wo1DsSbT
0UYzu8Zbq9e9UF+3rggo2bchkZvMJFG3kGiuBl9DQq/
sVFi0LSxaSn0hC/2z9AocTg5SF+O+lzKCcIOQHis/pfIr7566p4uMaS67ZlleW3E9HU8bJHLhZpLIh
WEEh8fm8Cn7up251KCnTLFNo9n6EL5WSWQjN9DE7DknsOfqPqgZoVnHncA/
TtCMZ6BPqEyFaJh03JFcGvpdMsuMC9lHIkph+lG6/phIzB1KYoj1vBtoknGr1ppqjZ8ouXbl07Oc/
sk7GU8meCQLRrIuPEuVWJ++J1h12BxIH0TjE+eIzvlNBIbym1VlwDERcm3NMeG54M6suJGullvReOu
SbVFEZxFanij5ZJ7CdXtNJ7cOzXRzVWZ/uZijUDnpvU/d1wttJM2CA0Sdmvb88eEO+RyrLO8brNLUv
Znr2qtcV3RKvP+BkKOWTWZQU4wt1IrOjlMsCHLTrUOz6Iw47dNgM2rVAbGqK3Vv6/U2O7oLkd+HanV
OpUgvyO5B+R2sXkymmUCPrrLLPenMOem49yt+1wtqflCqtPxByat7lVLL79ZLXd+vVwd+tdLv1R6AU
WQUV/107iH8s08Xy7f3enzrDX68KrXPjVhcZroOLmth/Qa/
Wit+g+8QsMz9Rm3Yrrd7jVK73h2WvH6vVWoHjV6p3wia/
WE/8Fvt4QPXOdZgr1sPvMagVWpUg6DkNSqKfqtdanq1WtdrdlsDr/tgaWtY+ep3ZV7N6+K/AAAA//8
DAFBLAwQKAAAAAAAAACEAeyDYDDwIAAA8CAAAFwAAAGRvY1Byb3BzL3RodW1ibmFpbC5qcGVn/9j/4
AAQSkZJRgABAQAAYABgAAD/4QCMRXhpZgAATU0AKgAAAAgABQESAAMAAAABAAEAAAEaAAUAAAABAAA
ASgEbAAUAAAABAAAAUgEoAAMAAAABAAIAAIdpAAQAAAABAAAAWgAAAAAAAABgAAAAAQAAAGAAAAABA
AOgAQADAAAAAQABAACgAgAEAAAAAQAAAQCgAwAEAAAAAQAAAJAAAAAA/
+0AOFBob3Rvc2hvcCAzLjAAOEJJTQQEAAAAAAAOEJJTQQlAAAAAAAQ1B2M2Y8AsgTpgAmY7PhCfv/
AABEIAJABAAMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACA
QMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiU
mJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXm
JmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/
T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAAB
AncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3OD
k6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeo
qaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2wBDAAICAgICAg
MCAgMFAwMDBQYFBQUFBggGBgYGBggKCAgICAgICgoKCgoKCgoMDAwMDAwODg4ODg8PDw8PDw8PDw//

2wBDAQICAgQEBAcEBAcQCwkLEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEB
AQEBAQEBAQEBD/3QAEABD/2gAMAwEAAhEDEQA/AP38oooOAKKKACiiigAooooAKKKACiiigAoooo
AKKKACiiigAooooAKKKACiiigAooooAKKKACiiigAooooA//Q/fyiiigAooooAKKKACiiigAoo
ooAKKKACiiigAooooAKKKACiiigAooooAKKKACiiigAooooAKKKACiiigD/9H9/KKKKACiiigA
ooooAKKKACiiigAooooAKKKACiiigAooooAKKKACiiigAooooAKKKACiiigAooooAKKKACiiigA
ooooAKKKACiiigAooooAKKKACiiigAooooAKKKAP/0v
38ooooAKKKACiiigAooooAKKKACiiigAooooAKK+fvjz8WfF/wALJfA6eE/C8viX/
hJteh0u4WKS3jZI5IpH2oZ7iACVynyE5QBW3EErnD8TftReHPC99rP2vw3q1xo3hOW3t/EOqQ/
ZDbaTPcRxytHIpuBNP9nSZGuGt45FjU9SQwAB9O0V4fZ/Gl9b8Z+I/
CXhXwjqmtxeFp5LK9voZLGOFb9bRLxbcRz3Mc/
zpIirKYxHvYDdtBYVf2bvip4r+Mfwr0jxx4v8NyeHry/
iV+XgaC4ySC8AinmdUGMYl2tntjmgD3qiiigAooooAKKKACiiigAooooAKKKACiiigD/9P9/
KKKACiiigAooooAKKKACiiigAooooAKKKAPHfjR8P/
EXj3R9Bm8IXtrZa34X1qz1q0+3I72sz2wdGil8oh1V45Wwy5IbBwRkV4Le/su6pH4817xVb6L4I8Qw
+Lr2DU76TX9He8u7K78iGG5S1kDZlgfyfMjSRkMbM3zMDgfY+ta3pPhzTJtZ1y6SzsrfbvlkOAC7BF
HqSzEKoHJJAAyRXBXPxs+E9mzR3HiiyV1kaLYJMsXXggKASfm+XI43/
L97igDzG/8Agx4z1n466T8T7q40PSrTRbmSQXmmW1xBrGo2TW8kMen37mUwywo7iTcQ3MabEjOTXa/
Ab4f+L/hZ4Gh+H3iS+sdSsNEZoNLntI5YpntN7MhuVdmUSgMAdnynGe+BoWvx4+D15FDJD4ssQbj7O
FR5CkmbpDJEGRgGUsgJIIG3B3Ywa77wz4l0Xxj4fsPFHh24+1aZqcSzW8pR498bdDtcKw+hANAG7RR
RQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB//1P38oooOAKKKACiiigAooooAKKKACiiigAooooA
xtf0DSvE+lyaNrUTTWsrRuQkjxOrwyLLG6SRsro6OqsrKwIIBBrhoPgr8MbeWaddEV5Jz8xkmml4M6
3O0b3O1POXzNgwu4scZZs+pUUAeS6d8DPhVpSyLZaEqiXG7dPPJnERhyd8jcmMkMerdTk816JoWh6Z
4b0m20LRojBY2a7IYy7ybEySFDOWbAzhRnAGAMAAVrUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAF
FFFAH/9X9/KKKACiiigAooooAKKKACiiigAooooAKKKACiiigAooooAKKKACiiigAooooAKKKK
ACiiigAooooAKKKKAP/1v38oooOAKKKACiiigAooooAKKKACiiigAooooAKKKACiiigAooooAKK
KKACiiigAooooAKKKACiiigAooooA//X/fyiiigAooooAKKKACiiigAooooAKKKACiiigAooooA
KKKACiiigAooooAKKKACiiigAooooA//9lQSwMEFAAGAAgAAAAhAKNkI2uNAQAAMgM
AABEAAABwcHQvcHJlc1Byb3BzLnhtbKzSUW/bIBAA4PdJ+w8WIxgmaQhI/qaq/QIxgmaM
QhI26nqfx9z0iirdNKma9x8D5JeSveDYap0WIWdAgxOP8QI9oRRjhrRQM
7jku4/km3FUveSmP+kIOBdxclok/qisf61mP1Ltuo93pG1sUj6Frz5cVsnJqRo8dwXbdWXWQIbpDmY
kS2Fbdi1knNACY4KbtHj5lU2yalC+F274osVBdoMKXATxiiPZHzyteme8GcOqN/rSJ7LmUTpr1NIqw
Zf3ehBTDTBA2w1acO+NnJIGs7SBRbluYEbTEjYt57Btm3XOWIpzgt+MchSnKSxGbtV/5NG0YMXfiHu
ed/um4RB3uw5mOe1guaYEZqxNadvFQLMzMa/6o3Dhzon+e5ybWzm2wsvhDZr/
CzS9hpJr5Dku345+H/PtTwAAAP//
AwBQSwMEFAAGAAgAAAAhAJNPwrtuAQAABQMAABEAAABwcHQvdmlld1Byb3BzLnhtbIxSPW/DIBDdK/
U/IPYGO3KdxGsfZLnhtbIxSPW/DIBDdK/
KGuok2XAFWadtQD6GtSWnpAYgbSaZRlJKGCoXHfvtIv64qwfi7ZruGKz+QWC6pB/FuK4w7splH2Izl
Dmj67itJBZhToVD+9BZDDLVeW16ueOWR+4VVvaZxjMklttGmh+ZJOk8CRG55nBQlP4dsLcshQk5Rs9
GfVpQ5jnoNbET21K4ZlbD9uM+7EBQLmrkWhU+bxhhBUxxFYSikuztpcuozmbaiFgq1OZ4lCUYdFM3S
sWgcGsrqHWhdOT8CJ6kD2bURpT13G976C28Xrv8pHoRdqT2n7iuNep1HKyfufsU3o2vY4tpQBseGWL
AJnwuHzLrjc2AZLrj4AwAA//8DAFBLAwQUAAYACAAAACEA2P2Nj6wAAAC2AAAEwAAAHBwdC90YWJs
ZVN0eWxlcy54bWwMzEkOgjAYQOG9iXdo/n0tQ1EkFMIgK3fqASqUIelAAAC2AAAEwAAAHBwdC90YWJs
xkNAP/4AESujXdpAcGj3uDY0DWcd1xabRgsAoLebbfpTxxT3lzqxRX69CmaJtwBqNzc0KIbUehuD2Y

Wejt9WZR3G25DKRb+HvTlSSB5x2J4pMG1ImewTeqgiCitMCny+WIaUgDXHo0xnFU1tW5qf0qLH5Asj
8AAAD//wMAUEsDBBQABgAIAAAAIQA4JSIgCQIAAAMFAAAQAAgBZG9jUHJvcHMvYXBwLnhtbCCiBAEo
oAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAKRUUW/aMBB+n7T/YOW9BLoNTcikqqgQD2VFIrTP1/hCrD
m2ZXus7NfvYpcUWlSpW17y3d2Xz5fPPvOrp1axHTovjZ5mo8EwY6grI6TeTrNNOb/4njEfQAtQRuM0
26PProrPn/jKGYsuSPSMJLSfZk0IdpLnvmqwBT+gsqZKbVwLgUK3zU1dywpvTPWrRR3yy+FwnONTQC
1QXNheMEuKk134V1Fhqq4/f1/uLekVvDQBVClbLL7w/CXgD8YJ3+US4NfWKllBIDeKpayc8aYObAmV
1MH4hq3Mb3QrQxHPj7lkB3paPkbz2F3xIAX6yiFqnp8p8xU42DqwjS8uifES8bXqPi1GPH9G/
IcJ9BryPAG+kEKgfq5S+iTmy+VMSRsLB8jXFSickSlFDcojSfcJvkDoNnwF0hFzFyY7rIJxzMs/
tOXjjD2Cx87KabYDJ0GHLNFSELGyPrhibnTwbONR8LxPRnjMPcbya2d/Au8Sk1ZJ5wA/
oD36gHa0j5UyKPT/v0QMoo+ETx1OS9zVtOfhjOHfjg2PPSS7UzvXpK+O2+vRDJR8dPK9GruV2yacZd
zFOWJvDO4RHbyTL09+8NUvzUxrQe+p0KNbqX/6jS3NDQQ8HMLTJF834FDQLPeHtE/
wBdnhVMefNaC3KA6ct4VujO/TnVaMxoMhPXFcD7luHg+XTfEXAAD//wMAUEsDBBQABgAIAAAAIQCdD
F17YAEAAJgCAAARAAgBZG9jUHJvcHMvY29yZS54bWwgogQBKKAAAQAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAACMkt9KwzAUh+8F36Hkvk3/OJmlzUBlVw4EJ4p3IT3bgk0akrhu976C4EN44Y0vJfgOpt3abbgLo
Tenvy8f55wkG61E6S1BG17JHEVBiDyQrCq4nOfofjr2h8gzlsqClpWEHK3BoBE5PcmYSlml4VZXCrT
lYDxnkiZlKkcLa1WKsWELENQEjpAunFVaUOtKPceKsmc6BxyH4TkWYGlBLcWN0Fe9EW2VBeuV6kWXr
aBgGEoQIK3BURDhHWtBC3P0QJvskYLbtYKjaBf29MrwHqzrOqiTFnX9R/
hxcnPXjupz2eyKASJZwVLLbQnEgrEZ7ssmYBqorTT5/nz7eX/9/vpo8+5vs9qSGjtxtzDjUFyu98G/
YcNrWPLmCkncEn2ZbfexkUPhuTnSzdRd8pBcXU/HiMRhnPjh0A+TaRSng8h9T01fB+d3QrFt4P/
Gs3RwsWfsBKTt+PAtkV8AAAD//wMAUEsBAi0AFAAGAAgAAAAhAN/MGPWtAQAARgwAABMAAAAAAAAAAAA
AAAAAAAAAAAAFtDb250ZW50X1R5cGVzXS54bWxQSwECLQAUAAYACAAAACEAaPh0oQMBAADiAgAACwA
AAAAAAAAAAAAAAADmAwAAX3JlbHMvLnJlbHNQSwECLQAUAAYACAAAACEAY1wjtMAAAAA3AQAAIAAAA
AAAAAAAAAAAAAAaBwAAcHB0L3NsaWRlcy9fcmVscy9zbGlkZTEueG1sLnJlbHNQSwECLQAUAAYACAA
AACEAfp3iy7ECAAAMBwAAFQAAAAAAAAAAAAAAAAAYCAAAcHB0L3NsaWRlcy9zbGlkZTEueG1sUEsBA
i0AFAAGAAgAAAAhABsuNQcMAQAA0AMAAB8AAAAAAAAAAAAAAAAA/AoAAHBwdC9fcmVscy9wcmVzZW5
0YXRpb24ueG1sLnJlbHNQSwECLQAUAAYACAAAACEAzKy0p0ACAAC0DAAAFAAAAAAAAAAAAAAAAABND
QAAcHB0L3ByZXNlbnRhdGlvbi54bWxQSwECLQAUAAYACAAAACEA1dGS8bwAAAA3AQAALAAAAAAAAAAAAA
AAAAAAAC/DwAAcHB0L3NsaWRlTGF5b3V0cy9fcmVscy9zbGlkZUxheW91dDkueG1sLnJlbHNQSwECL
QAUAAYACAAAACEAS+JCggcFAAB9EgAAIQAAAAAAAAAAAAAAAAADFEAAAcHB0L3NsaWRlTGF5b3V0cy9
zbGlkZUxheW91dDEueG1sUEsBAi0AFAAGAAgAAAAhAJsXm8xUBAAAiQ8AACEAAAAAAAAAAAAAAAAAC
xYAAHBwdC9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQyLnhtbFBLAQItABQABgAIAAAAIQBjIYG3PwU
AAF4VAAAhAAAAAAAAAAAAAAAAAAJ4aAABwcHQvc2xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0My54bWxQS
wECLQAUAAYACAAAACEAbfH63rcEAADCEwAAIQAAAAAAAAAAAAAAAAAcIAAAcHB0L3NsaWRlTGF5b3V
0cy9zbGlkZUxheW91dDQueG1sUEsBAi0AFAAGAAgAAAAhAHq382cRBgAAVR8AACEAAAAAAAAAAAAAAA
AAAEiUAAHBwdC9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQ1LnhtbFBLAQItABQABgAIAAAAIQD5r5w
J3AMAABQMAAAhAAAAAAAAAAAAAAAAAAGIrAABwcHQvc2xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0Ni54b

```
WxQSwECLQAUAAYACAAAACEAlIYDFYUDAAD2CQAAIQAAAAAAAAAAAAAAAAB9LwAAcHB0L3NsaWRlTGF
5b3V0cy9zbGlkZUxheW91dDcueG1sUEsBAi0AFAAGAAgAAAAhAIgycve5BQAAdxcAACEAAAAAAAAAA
AAAAAAAQTMAAHBwdC9zbGlkZUxheW91dHMvc2xpZGVMYXlvdXQ4LnhtbFBLAQItABQABgAIAAAAIQB
r9+RZiAUAACYXAAAhAAAAAAAAAAAAAAAAAADk5AABwcHQvc2xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0O
S54bWxQSwECLQAUAAYACAAAACEAGkEHEXEEAAC+DwAAIgAAAAAAAAAAAAAAAAAAPwAAcHB0L3NsaWR
lTGF5b3V0cy9zbGlkZUxheW91dDEwLnhtbFBLAQItABQABgAIAAAAIQAVVR/0pgQAAJ8QAAAiAAAAA
AAAAAAAAAAALFDAABwcHQvc2xpZGVMYXlvdXRzL3NsaWRlTGF5b3V0MTEueG1sUEsBAi0AFAAGAAg
AAAAhANXRkvG8AAAANwEAACwAAAAAAAAAAAAAAAAAl0gAAHBwdC9zbGlkZUxheW91dHMvX3JlbHMvc
2xpZGVMYXlvdXQ2LnhtbC5yZWxzUEsBAi0AFAAGAAgAAAAhANXRkvG8AAAANwEAACwAAAAAAAAAAA
AAAAAnUkAAHBwdC9zbGlkZUxheW91dHMvX3JlbHMvc2xpZGVMYXlvdXQ3LnhtbC5yZWxzUEsBAi0AF
AAGAAgAAAAhANXRkvG8AAAANwEAAC0AAAAAAAAAAAAAAAAAAo0oAAHBwdC9zbGlkZUxheW91dHMvX3J
lbHMvc2xpZGVMYXlvdXQxMS54bWwucmVsc1BLAQItABQABgAIAAAAIQDV0ZLxvAAAADcBAAAtAAAAA
AAAAAAAAAAAKpLAABwcHQvc2xpZGVMYXlvdXRzL19yZWxzL3NsaWRlTGF5b3V0MTAueG1sLnJlbHN
QSwECLQAUAAYACAAAACEA1dGS8bwAAAA3AQAALAAAAAAAAAAAAAAAAACxTAAAcHB0L3NsaWRlTGF5b
3V0cy9fcmVscy9zbGlkZUxheW91dDgueG1sLnJlbHNQSwECLQAUAAYACAAAACEAaaJfIRUBAADHBwA
ALAAAAAAAAAAAAAAAAAC3TQAAcHB0L3NsaWRlTWFzdGVycy9fcmVscy9zbGlkZU1hc3RlcjEueG1sL
nJlbHNQSwECLQAUAAYACAAAACEAw0QEahMIAABsNgAAIQAAAAAAAAAAAAAAAAWTwAAcHB0L3NsaWR
lTWFzdGVycy9zbGlkZU1hc3RlcjEueG1sUEsBAi0AFAAGAAgAAAAhANXRkvG8AAAANwEAACwAAAAAA
AAAAAAAAAAaFcAAHBwdC9zbGlkZUxheW91dHMvX3JlbHMvc2xpZGVMYXlvdXQxLnhtbC5yZWxzUEs
BAi0AFAAGAAgAAAAhANXRkvG8AAAANwEAACwAAAAAAAAAAAAAAAAblgAAHBwdC9zbGlkZUxheW91d
HMvX3JlbHMvc2xpZGVMYXlvdXQyLnhtbC5yZWxzUEsBAi0AFAAGAAgAAAAhANXRkvG8AAAANwEAACw
AAAAAAAAAAAAAAAAAdFkAAHBwdC9zbGlkZUxheW91dHMvX3JlbHMvc2xpZGVMYXlvdXQzLnhtbC5yZ
WxzUEsBAi0AFAAGAAgAAAAhANXRkvG8AAAANwEAACwAAAAAAAAAAAAAAAAAeloAAHBwdC9zbGlkZUx
heW91dHMvX3JlbHMvc2xpZGVMYXlvdXQ0LnhtbC5yZWxzUEsBAi0AFAAGAAgAAAAhANXRkvG8AAAAN
wEAACwAAAAAAAAAAAAAAAAAgFsAAHBwdC9zbGlkZUxheW91dHMvX3JlbHMvc2xpZGVMYXlvdXQ1Lnh
tbC5yZWxzUEsBAi0AFAAGAAgAAAAhAHtDvF3EBgAAzyAAABQAAAAAAAAAAAAAAAAAhlwAAHBwdC90a
GVtZS90aGVtZTEueG1sUEsBAi0ACgAAAAAAAAAhAHsg2Aw8CAAAPAgAABcAAAAAAAAAAAAAAAAAfGM
AAGRvY1Byb3BzL3RodW1ibmFpbC5qcGVnUEsBAi0AFAAGAAgAAAAhAKNkI2uNAQAAMgMAABEAAAAAA
AAAAAAAAAA7WsAAHBwdC9wcmVzUHJvcHMueG1sUEsBAi0AFAAGAAgAAAAhAJNPwrtuAQAABQMAABE
AAAAAAAAAAAAAAAAAqW0AAHBwdC92aWV3UHJvcHMueG1sUEsBAi0AFAAGAAgAAAAhANj9jY+sAAAAt
gAAABMAAAAAAAAAAAAAAAAARm8AAHBwdC90YWJsZVN0eWxlcy54bWxQSwECLQAUAAYACAAAACEAOCU
iIAkCAAADBQAAEAAAAAAAAAAAAAAAAAAjcAAAZG9jUHJvcHMvYXBwLnhtbFBLAQItABQABgAIAAAAI
QCdDF17YAEAAJgCAAARAAAAAAAAAAAAAAAAAGJzAABkb2NQcm9wcy9jb3JlLnhtbFBLBQYAAAAAJQA
lAE0LAAD5dQAAAAA=
```

langchain4j\src\test\resources\test-file.xls

BASE64:

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAPgADAP7/
CQAGAAAAAAAAAAAAAABAAAAMQAAAAAAAAAAEAAA/v///wAAAAD+////AAAAADAAAAD//////////
//
//
//
//
//
//
//
/////////////////8JCBAAAAYFAFpPzQfBAAIABggAAOEAAgCwBMEAAgAAAOIAAABcAHAAAwABFa
wE1mytICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIEIAAgCwBGEBAg
AAAMABAAA9AQQAAQACAJwAAgARABkAAgAAABIAAgAAABMAAgAAAK8BAgAAALwBAgAAAD0AEgA8LRgV
iG0gRDgAAAAAAAEAWAJAAAIAAACNAAIAAAAiAAIAAAAOAAIAAQC3AQIAAADaAAIAAAAxAB4A8AAAAA
gAkAEAAAACAAAHAUMAYQBsAGkAYgByAGkAMQAeAPAAAAAIAJABAAAAAgAABwFDAGEAbABpAGIAcgBp
ADEAHgDwAAAACACQAQAAAAIAAAcBQwBhAGwAaQBiAHIAaQAxAB4A8AAAAAgAkAEAAAACAAAHAUMAYQ
BsAGkAYgByAGkAMQAeAPAAAAAIAJABAAAAAgAABwFDAGEAbABpAGIAcgBpADEAKgBoAQAANgCQAQAA
AAIAAA0BQwBhAGwAaQBiAHIAaQAgAEwAaQBnAGgAdAAxAB4ALAEBADYAvAIAAAACAAAHAUMAYQBsAG
kAYgByAGkAMQAeAAQBAQA2ALwCAAAAAgAABwFDAGEAbABpAGIAcgBpADEAHgDcAAEANgC8AgAAAAIA
AAcBQwBhAGwAaQBiAHIAaQAxAB4A8AAAABEAkAEAAAACAAAHAUMAYQBsAGkAYgByAGkAMQAeAPAAAA
AUAJABAAAAAgAABwFDAGEAbABpAGIAcgBpADEAHgDwAAAAPACQAQAAAAIAAAcBQwBhAGwAaQBiAHIA
aQAxAB4A8AAAAD4AkAEAAAACAAAHAUMAYQBsAGkAYgByAGkAMQAeAPAAAQA/ALwCAAAAAgAABwFDAG
EAbABpAGIAcgBpADEAHgDwAAEANAC8AgAAAAIAAAcBQwBhAGwAaQBiAHIAaQAxAB4A8AAAADQAkAEA
AAACAAAHAUMAYQBsAGkAYgByAGkAMQAeAPAAAQAJALwCAAAAAgAABwFDAGEAbABpAGIAcgBpADEAHg
DwAAAACgCQAQAAAAIAAAcBQwBhAGwAaQBiAHIAaQAxAB4A8AACABcAkAEAAAACAAAHAUMAYQBsAGkA
YgByAGkAMQAeAPAAAQAIALwCAAAAAgAABwFDAGEAbABpAGIAcgBpADEAHgDwAAAACQCQAQAAAAIAAA
cBQwBhAGwAaQBiAHIAaQAeBDMABQAXAAEiAKkgIgAjACwAIwAjADAAXwApADsAXAAoACIAqSAiACMA
LAAjACMAMABcACkAHgQ9AAYAHAABIgCpICIAIwAsACMAIwAwAF8AKQA7AFsAUgBlAGQAXQBcACgAIg
CpICIAIwAsACMAIwAwAFwAKQAeBD8ABwAdAAEiAKkgIgAjACwAIwAjADAALgAwADAAXwApADsAXAAo
ACIAqSAiACMALAAjACMAMAAuADAAMABcACkAHgRJAAgAIgABIgCpICIAIwAsACMAIwAwAC4AMAAwAF
8AKQA7AFsAUgBlAGQAXQBcACgAIgCpICIAIwAsACMAIwAwAC4AMAAwAFwAKQAeBGkAKgAyAAFfACgA
IgCpICIAKgAgACMALAAjACMAMABfACkAOwBfACgAIgCpICIAKgAgAFwAKAAjACwAIwAjADAAXAApAD
sAXwAoACIAqSAiACoAIAAiAC0AIgBfACkAOwBfACgAQABfACkAHgQuACkAKQAAXygqICMsIyMwXyk7
XygqIFwoIywjIzBcKTtfKCogIi0iXyk7XyhAXykeBHkALAA6AAFfACgAIgCpICIAKgAgACMALAAjAC
MAMAAuADAAMABfACkAOwBfACgAIgCpICIAKgAgAFwAKAAjACwAIwAjADAALgAwADAAXAApADsAXwAo
ACIAqSAiACoAIAAiAC0AIgA/AD8AXwApADsAXwAoAEAAXwApAB4ENgArADEAAF8oKiAjLCMjMC4wMF
8pO18oKiBcKCMsIyMwLjAwXCk7XygqICItIj8/Xyk7XyhAXyngABQAAAAAAPX/
IAAAAAAAAAAAAAAAwCDgABQAAQAAAPX/IAAA9AAAAAAAAAAAwCDgABQAAQAAAPX/
IAAA9AAAAAAAAAAwCDgABQAAgAAAPX/IAAA9AAAAAAAAAAwCDgABQAAgAAAPX/
IAAA9AAAAAAAAAAwCDgABQAAAAAAPX/IAAA9AAAAAAAAAAwCDgABQAAAAAAPX/

IAAA9AAAAAAAAAAwCDgABQAAAAAPX/IAAA9AAAAAAAAAAwCDgABQAAAAAPX/
IAAA9AAAAAAAAAAwCDgABQAAAAAPX/IAAA9AAAAAAAAAAwCDgABQAAAAAPX/
IAAA9AAAAAAAAAAwCDgABQAAAAAPX/IAAA9AAAAAAAAAAwCDgABQAAAAAPX/
IAAA9AAAAAAAAAAwCDgABQAAAAAPX/IAAA9AAAAAAAAAAwCDgABQAAAAAPX/
IAAA9AAAAAAAAAAwCDgABQAAAAAAEAIAAAAAAAAAAAAAACwCDgABQABQAAAPX/
IAAAtAAAAAAAAAEnyDgABQABQAAAPX/IAAAtAAAAAAAAAEryDgABQABQAAAPX/
IAAAtAAAAAAAAAEiSDgABQABQAAAPX/IAAAtAAAAAAAAAEmiDgABQABQAAAPX/
IAAAtAAAAAAAAAEmyDgABQABQAAAPX/IAAAtAAAAAAAAAEqiDgABQABQAAAPX/
IAAAtAAAAAAAAAErCDgABQABQAAAPX/IAAAtAAAAAAAAAEryDgABQABQAAAPX/
IAAAtAAAAAAAAAEliDgABQABQAAAPX/IAAAtAAAAAAAAAEqyDgABQABQAAAPX/
IAAAtAAAAAAAAAErCDgABQABQAAAPX/IAAAtAAAAAAAAAEqyDgABQABQAAAPX/
IAAAtAAAAAAAAAEsSDgABQABQAAAPX/IAAAtAAAAAAAAAEryDgABQABQAAAPX/
IAAAtAAAAAAAAAEliDgABQABQAAAPX/IAAAtAAAAAAAAAEqyDgABQABQAAAPX/
IAAAtAAAAAAAAAErCDgABQABQAAAPX/IAAAtAAAAAAAAAEuSDgABQAFQAAAPX/
IAAAtAAAAAAAAAEviDgABQAFQAAAPX/IAAAtAAAAAAAAAEtSDgABQAFQAAAPX/
IAAAtAAAAAAAAAEtyDgABQAFQAAAPX/IAAAtAAAAAAAAAEsyDgABQAFQAAAPX/
IAAAtAAAAAAAAAEsSDgABQAFQAAAPX/IAAAtAAAAAAAAAEuSDgABQACwAAAPX/
IAAAtAAAAAAAAAErSDgABQADwAAAPX/IAAAlBERlwuXCwAEliDgABQAEQAAAPX/IAAAlGZmvx+/
HwAEtyDgABQABQArAPX/IAAA+AAAAAAAAAAwCDgABQABQApAPX/
IAAA+AAAAAAAAAAwCDgABQABQAsAPX/IAAA+AAAAAAAAAAwCDgABQABQAqAPX/
IAAA+AAAAAAAAAAwCDgABQAEwAAAPX/IAAA9AAAAAAAAAAwCDgABQACgAAAPX/
IAAAtAAAAAAAAAEqiDgABQABwAAAPX/IAAA1ABQAAAAHwAAwCDgABQACAAAAPX/
IAAA1ABQAAAAFgAAwCDgABQACQAAAPX/IAAA1AAgAACAGAAAwCDgABQACQAAAPX/
IAAA9AAAAAAAAAAwCDgABQADQAAAPX/IAAAlBERlwuXCwAEryDgABQAEAAAAPX/
IAAA1ABgAAAAGgAAwCDgABQADAAAAPX/IAAAtAAAAAAAAAEqyDgABQABQAAAPX/
IAAAnBERFgsWCwAEmiDgABQADgAAAPX/IAAAlBERvx+/HwAEliDgABQABQAJAPX/
IAAA+AAAAAAAAAwCDgABQABgAAAPX/IAAA9AAAAAAAAAAwCDgABQAFAAAAPX/
IAAA1ABhAAA+HwAAwCDgABQAEgAAAPX/IAAA9AAAAAAAAAAwCB8CBQAfAgAAAAAAAAAAAAAAAA+AN
fEaCB9CC0AfQgAAAAAAAAAAAAAAAAAAAAAgANABQAAwAAAAEAAAAwMFwpO18oKg4ABQACfQgtAH0I
AAAAAAAAAAAAAAAAQAAAAIADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAn0ILQB9CAAAAAAAAAAAAA
AAAAIAAAACAA0AFAADAAAAAQAAADAwXCk7XygqDgAFAAJ9CC0AfQgAAAAAAAAAAAAAAADAAAAAgAN
ABQAAwAAAAEAAAAwMFwpO18oKg4ABQACfQgtAH0IAAAAAAAAAAAAAAAABAAAAAIADQAUAAMAAAABAA
AAMDBcKTtfKCoOAAUAAn0ILQB9CAAAAAAAAAAAAAAAAUAAAACAA0AFAADAAAAAQAAADAwXCk7Xygq
DgAFAAJ9CC0AfQgAAAAAAAAAAAAAAAGAAAAAgANABQAAwAAAAEAAAAwMFwpO18oKg4ABQACfQgtAH
0IAAAAAAAAAAAAAAAABwAAAAIADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAn0ILQB9CAAAAAAAAAAA
AAAAAAgAAAACAA0AFAADAAAAAQAAADAwXCk7XygqDgAFAAJ9CC0AfQgAAAAAAAAAAAAAAAJAAAAAg
ANABQAAwAAAAEAAAAwMFwpO18oKg4ABQACfQgtAH0IAAAAAAAAAAAAAAAACgAAAAIADQAUAAMAAAAB
AAAAMDBcKTtfKCoOAAUAAn0ILQB9CAAAAAAAAAAAAAAAAsAAAACAA0AFAADAAAAAQAAADAwXCk7Xy
gqDgAFAAJ9CC0AfQgAAAAAAAAAAAAAAAAMAAAAgANABQAAwAAAAEAAAAwMFwpO18oKg4ABQACfQgt
AH0IAAAAAAAAAAAAAAAADQAAAAIADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAn0ILQB9CAAAAAAAAA
AAAAAAAA4AAAACAA0AFAADAAAAAQAAADAwXCk7XygqDgAFAAJ9CC0AfQgAAAAAAAAAAAAAAAPAAAA
AgANABQAAwAAAAEAAAAwMFwpO18oKg4ABQACfQgtAH0IAAAAAAAAAAAAAAAAKwAAAAIADQAUAAMAAA
ABAAAAMDBcKTtfKCoOAAUAAn0ILQB9CAAAAAAAAAAAAAAAACwAAAACAA0AFAADAAAAAQAAADAwXCk7
XygqDgAFAAJ9CC0AfQgAAAAAAAAAAAAAAAtAAAAAgANABQAAwAAAAEAAAAwMFwpO18oKg4ABQACfQ

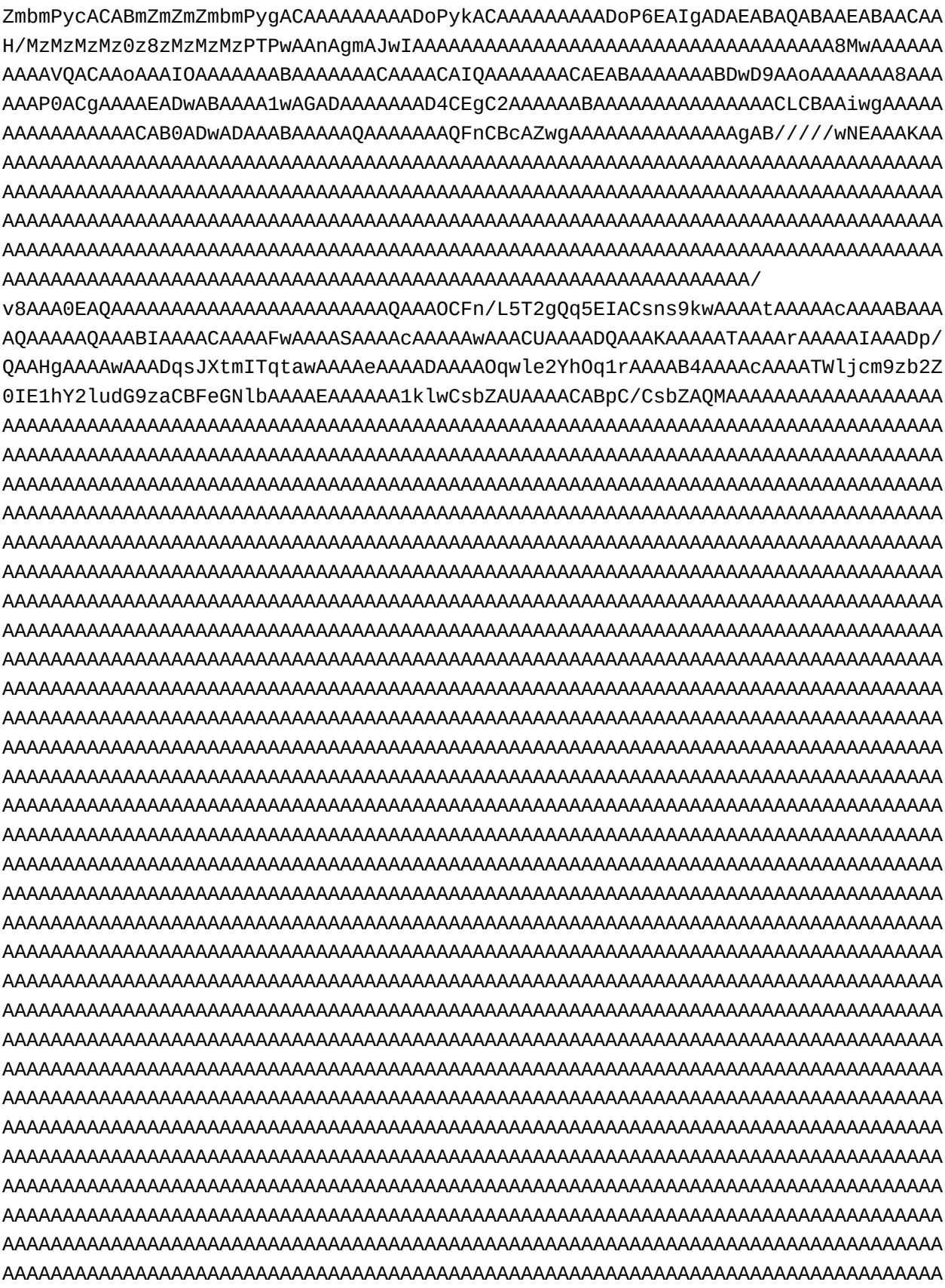
gtAH0IAAAAAAAAAAAAAAAALgAAAAIADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAn0ILQB9CAAAAAAA
AAAAAAAAADoAAAACAA0AFAADAAAAAQAAADAwXCk7XygqDgAFAAJ9CC0AfQgAAAAAAAAAAAAAAAA7AA
AAAgANABQAAwAAAAMAAAAwMFwpO18oKg4ABQABfQhBAH0IAAAAAAAAAAAAAAAAMQAAAAMADQAUAAMA
AAADAAAAMDBcKTtfKCoOAAUAAggAFAADAAAABAAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAMg
AAAAMADQAUAAMAAAADAAAAMDBcKTtfKCoOAAUAAggAFAADAP8/BAAAACgAIwAsACMAfQhBAH0IAAAA
AAAAAAAAAAAMwAAAAMADQAUAAMAAAADAAAAMDBcKTtfKCoOAAUAAggAFAADADIzBAAAACgAIwAsAC
MAfQgtAH0IAAAAAAAAAAAAAAAANAAAAAIADQAUAAMAAAADAAAAMDBcKTtfKCoOAAUAAn0IQQB9CAAA
AAAAAAAAAAAADAAAAADAA0AFAACAAAAAGEA/zAwXCk7XygqDgAFAAIEABQAAgAAAMbvzv8oACMALA
AjAH0IQQB9CAAAAAAAAAAAAAAACgAAAADAA0AFAACAAAAnAAG/
zAwXCk7XygqDgAFAAIEABQAAgAAAP/
Hzv8oACMALAAjAH0IQQB9CAAAAAAAAAAAAAAADcAAAADAA0AFAACAAAAnFcA/
zAwXCk7XygqDgAFAAIEABQAAgAAAP/
rnP8oACMALAAjAH0IkQB9CAAAAAAAAAAAAAAADUAAAAHAA0AFAACAAAAPz92/
zAwXCk7XygqDgAFAAIEABQAAgAAAP/Mmf8oACMALAAjAAcAFAACAAAAf39//
ykAOwBfACgACAAUAAIAAAB/f3//LQAiAD8APwAJABQAAgAAAH9/
f/9fACkAAAAAAAoAFAACAAAAf39//
wAAAAAAAAAAfQiRAH0IAAAAAAAAAAAAAAAAOQAAAAcADQAUAAIAAAA/Pz//
MDBcKTtfKCoOAAUAAgQAFAACAAAA8vLy/ygAIwAsACMABwAUAAIAAAA/Pz//
KQA7AF8AKAAIABQAAgAAAD8/P/8tACIAPwA/AAkAFAACAAAAPz8//18AKQAAAAAACgAUAAIAAAA/
Pz//AAAAAAAAAAB9CJEAfQgAAAAAAAAAAAAAAAApAAAABwANABQAAgAAAPp9AP8wMFwpO18oKg4ABQ
ACBAAUAAIAAADy8vL/KAAjACwAIwAHABQAAgAAAH9/f/8pADsAXwAoAAgAFAACAAAAf39//y0AIgA/
AD8ACQAUAAIAAAB/f3//XwApAAAAAAAKABQAAgAAAH9/
f/8AAAAAAAAAAH0IQQB9CAAAAAAAAAAAAAAADYAAAADAA0AFAACAAAA+n0A/zAwXCk7XygqDgAFAA
IIABQAAgAAAP+AAf8oACMALAAjAH0IkQB9CAAAAAAAAAAAAAAACoAAAAHAA0AFAADAAAAAAAADAw
XCk7XygqDgAFAAIEABQAAgAAAKWlpf8oACMALAAjAAcAFAACAAAAPz8//
ykAOwBfACgACAAUAAIAAAA/Pz//LQAiAD8APwAJABQAAgAAAD8/
P/9fACkAAAAAAAoAFAACAAAAPz8//
wAAAAAAAAAAfQgtAH0IAAAAAAAAAAAAAAAAPQAAAAIADQAUAAIAAAD/AAD/MDBcKTtfKCoOAAUAAn0
IkQB9CAAAAAAAAAAAAAAADgAAAAHAA0AFAADAAAAAQAAADAwXCk7XygqDgAFAAIEABQAAgAAAP//
zP8oACMALAAjAAcAFAACAAAAsrKy/ykAOwBfACgACAAUAAIAAACysrL/
LQAiAD8APwAJABQAAgAAALKysv9fACkAAAAAAAoAFAACAAAAsrKy/
wAAAAAAAAAAfQgtAH0IAAAAAAAAAAAAAAAALwAAAAIADQAUAAIAAAB/f3//MDBcKTtfKCoOAAUAAn0
IVQB9CAAAAAAAAAAAAAAADwAAAAEAA0AFAADAAAAAQAAADAwXCk7XygqDgAFAAIHABQAAwAAAAQAA
AAoACMALAAjAAgAFAADAAAABAAAACkAOwBfACgAfQhBAH0IAAAAAAAAAAAAAAAAIgAAAAMADQAUAAM
AAAAAAAAAMDBcKTtfKCoOAAUAAgQAFAADAAAABAAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAE
AAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADAGVmBAAAACgAIwAsACMAfQhBAH0IAAA
AAAAAAAAAAAAFgAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADAMxMBAAAACgAIwAsA
CMAfQhBAH0IAAAAAAAAAAAAAAAAHAAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADADI
zBAAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAIwAAAAMADQAUAAMAAAAAAAAAMDBcKTtfKCoOA
AUAAgQAFAADAAAABQAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAEQAAAAMADQAUAAMAAAABAAA
AMDBcKTtfKCoOAAUAAgQAFAADAGVmBQAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAFwAAAAMAD
QAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADAMxMBQAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAA
AAAAAHQAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADADIzBQAAACgAIwAsACMAfQhBA
H0IAAAAAAAAAAAAAAAAJAAAAAMADQAUAAMAAAAAAAAAMDBcKTtfKCoOAAUAAgQAFAADAAAABgAAACg

AIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAEgAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAF
AADAGVmBgAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAGAAAAAMADQAUAAMAAAABAAAAMDBcKTt
fKCoOAAUAAgQAFAADAMxMBgAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAHgAAAAMADQAUAAMAA
AABAAAAMDBcKTtfKCoOAAUAAgQAFAADADIzBgAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAJQA
AAAMADQAUAAMAAAAAAAAAMDBcKTtfKCoOAAUAAgQAFAADAAAABwAAACgAIwAsACMAfQhBAH0IAAAAA
AAAAAAAAAAEwAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADAGVmBwAAACgAIwAsACM
AfQhBAH0IAAAAAAAAAAAAAAAAGQAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADAMxMB
wAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAHwAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAU
AAgQAFAADADIzBwAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAJgAAAAMADQAUAAMAAAAAAAAAM
DBcKTtfKCoOAAUAAgQAFAADAAAACAAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAFAAAAAMADQA
UAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADAGVmCAAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAA
AAAGgAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADAMxMCAAAACgAIwAsACMAfQhBAH0
IAAAAAAAAAAAAAAAAIAAAAAMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADADIzCAAAACgAI
wAsACMAfQhBAH0IAAAAAAAAAAAAAAAAJwAAAAMADQAUAAMAAAAAAAAAMDBcKTtfKCoOAAUAAgQAFAA
DAAAACQAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAFQAAAAMADQAUAAMAAAABAAAAMDBcKTtfK
CoOAAUAAgQAFAADAGVmCQAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAGwAAAAMADQAUAAMAAAA
BAAAAMDBcKTtfKCoOAAUAAgQAFAADAMxMCQAAACgAIwAsACMAfQhBAH0IAAAAAAAAAAAAAAAAIQAAA
AMADQAUAAMAAAABAAAAMDBcKTtfKCoOAAUAAgQAFAADADIzCQAAACgAIwAsACMAkwISABAADQAAMjA
lIC0gQWNjZW50MZIITQCSCAAAAAAAAAAAAAAABBB7/DQAyADAAJQAgAC0AIABBAGMAYwBlAG4AdAAxA
AAAAwABAAwABwRlZtnh8v8FAAwABwEAAAAAAP8lAAUAApMCEgARAA0AADIwJSAtIEFjY2VudDKSCE0
AkggAAAAAAAAAAAAAAQQi/
w0AMgAwACUAIAAtACAAQQBjAGMAZQBuAHQAMgAAAAMAAQAMAAcFZWb85Nb/BQAMAAcBAAAAAAD/JQA
FAAKTAhIAEgANAAAyMCUgLSBBY2NlbnQzkghNAJIIAAAAAAAAAAAAAAAEEJv8NADIAMAAlACAALQAgA
EEAYwBjAGUAbgB0ADMAAAADAAEADAAHBmVm7e3t/wUADAAHAQAAAAA/
yUABQACkwISABMADQAAMjAlIC0gQWNjZW50NJIITQCSCAAAAAAAAAAAAAABBCr/
DQAyADAAJQAgAC0AIABBAGMAYwBlAG4AdAA0AAAAAwABAAwABwdlZv/yzP8FAAwABwEAAAAAP8lAA
UAApMCEgAUAA0AADIwJSAtIEFjY2VudDWSCE0AkggAAAAAAAAAAAAAAQQu/
w0AMgAwACUAIAAtACAAQQBjAGMAZQBuAHQANQAAAAMAAQAMAAcIZWbd6/f/BQAMAAcBAAAAAAD/JQA
FAAKTAhIAFQANAAAyMCUgLSBBY2NlbnQ2kghNAJIIAAAAAAAAAAAAAAAEEMv8NADIAMAAlACAALQAgA
EEAYwBjAGUAbgB0ADYAAAADAAEADAAHCWVm4u/a/wUADAAHAQAAAAA/
yUABQACkwISABYADQAANDAlIC0gQWNjZW50MZIITQCSCAAAAAAAAAAAAAABBB//DQA0ADAAJQAgAC0
AIABBAGMAYwBlAG4AdAAxAAAAAwABAAwABwTMTLTG5/8FAAwABwEAAAAAP8lAAUAApMCEgAXAA0AA
DQwJSAtIEFjY2VudDKSCE0AkggAAAAAAAAAAAAAAQQj/
w0ANAAwACUAIAAtACAAQQBjAGMAZQBuAHQAMgAAAAMAAQAMAAcFzEz4y63/BQAMAAcBAAAAAAD/JQA
FAAKTAhIAGAANAAA0MCUgLSBBY2NlbnQzkghNAJIIAAAAAAAAAAAAAAAEEJ/8NADQAMAAlACAALQAgA
EEAYwBjAGUAbgB0ADMAAAADAAEADAAHBsxM29vb/wUADAAHAQAAAAA/
yUABQACkwISABkADQAANDAlIC0gQWNjZW50NJIITQCSCAAAAAAAAAAAAAABBCv/
DQA0ADAAJQAgAC0AIABBAGMAYwBlAG4AdAA0AAAAAwABAAwABwfMTP/mmf8FAAwABwEAAAAAP8lAA
UAApMCEgAaAA0AADQwJSAtIEFjY2VudDWSCE0AkggAAAAAAAAAAAAAAQQv/
w0ANAAwACUAIAAtACAAQQBjAGMAZQBuAHQANQAAAAMAAQAMAAcIzEy91+7/BQAMAAcBAAAAAAD/JQA
FAAKTAhIAGwANAAA0MCUgLSBBY2NlbnQ2kghNAJIIAAAAAAAAAAAAAAAEEM/8NADQAMAAlACAALQAgA
EEAYwBjAGUAbgB0ADYAAAADAAEADAAHCcxMxuC0/wUADAAHAQAAAAA/
yUABQACkwISABwADQAANjAlIC0gQWNjZW50MZIITQCSCAAAAAAAAAAAAAABBCD/DQA2ADAAJQAgAC0
AIABBAGMAYwBlAG4AdAAxAAAAAwABAAwABwQyM46p2/8FAAwABwEAAAAAP8lAAUAApMCEgAdAA0AA

DYwJSAtIEFjY2VudDKSCE0AkggAAAAAAAAAAAAAQQk/
w0ANgAwACUAIAAtACAAQQBjAGMAZQBuAHQAMgAAAAMAAQAMAAcFMjP0sIT/BQAMAAcBAAAAAAD/JQA
FAAKTAhIAHgANAAA2MCUgLSBBY2NlbnQzkghNAJIIAAAAAAAAAAAAAAAEEKP8NADYAMAAlACAALQAgA
EEAYwBjAGUAbgB0ADMAAAADAAEADAAHBjIzycnJ/wUADAAHAQAAAAAA/
yUABQACkwISAB8ADQAANjAlIC0gQWNjZW50NJIITQCSCAAAAAAAAAAAAAAABBCz/
DQA2ADAAJQAgAC0AIABBAGMAYwBlAG4AdAA0AAAAAwABAAwABwcyM//ZZv8FAAwABwEAAAAAAP8lAA
UAApMCEgAgAA0AADYwJSAtIEFjY2VudDWSCE0AkggAAAAAAAAAAAAAAQQw/
w0ANgAwACUAIAAtACAAQQBjAGMAZQBuAHQANQAAAAMAAQAMAAcIMjObwub/BQAMAAcBAAAAAAD/JQA
FAAKTAhIAIQANAAA2MCUgLSBBY2NlbnQ2kghNAJIIAAAAAAAAAAAAAAEENP8NADYAMAAlACAALQAgA
EEAYwBjAGUAbgB0ADYAAAADAAEADAAHCTIzqdCO/wUADAAHAQAAAAAA/
yUABQACkwIMACIABwAAQWNjZW50MZIIQQCSCAAAAAAAAAAAAAAABBB3/BwBBAGMAYwBlAG4AdAAxAAA
AAwABAAwABwQAAERyxP8FAAwABwAAAP////8lAAUAApMCDAAjAAcAAEFjY2VudDKSCEEAkggAAAAAA
AAAAAAAAQQh/wcAQQBjAGMAZQBuAHQAMgAAAAMAAQAMAAcFAADtfTH/BQAMAAcAAAD/////JQAFAAK
TAgwAJAAHAABBY2NlbnQzkghBAJIIAAAAAAAAAAAAAAAEEJf8HAEEAYwBjAGUAbgB0ADMAAAADAAEAD
AAHBgAApaWl/wUADAAHAAAA/////
yUABQACkwIMACUABwAAQWNjZW50NJIIQQCSCAAAAAAAAAAAAAAABBCn/
BwBBAGMAYwBlAG4AdAA0AAAAAwABAAwABwcAAP/
AAP8FAAwABwAAAP////8lAAUAApMCDAAmAAcAAEFjY2VudDWSCEEAkggAAAAAAAAAAAAAAQQt/
wcAQQBjAGMAZQBuAHQANQAAAAMAAQAMAAcIAABbm9X/BQAMAAcAAAD/////JQAFAAKTAgwAJwAHAAB
BY2NlbnQ2kghBAJIIAAAAAAAAAAAAAAAEEMf8HAEEAYwBjAGUAbgB0ADYAAAADAAEADAAHCQAAcK1H/
wUADAAHAAAA/////
yUABQACkwIIACgAAwAAQmFkkgg5AJIIAAAAAAAAAAAAAAEBG/8DAEIAYQBkAAAAAwABAAwABf8AAP/
Hzv8FAAwABf8AAJwABv8lAAUAApMCEAApAAsAAENhbGN1bGF0aW9ukgiBAJIIAAAAAAAAAAAAAAECF
v8LAEMAYQBsAGMAdQBsAGEAdABpAG8AbgAAAAcAAQAMAAX/AADy8vL/BQAMAAX/AAD6fQD/
JQAFAAIGAA4ABf8AAH9/f/8BAAcADgAF/wAAf39//wEACAAOAAX/AAB/f3//AQAJAA4ABf8AAH9/
f/8BAJMCDwAqAAoAAENoZWNrIENlbGySCH8AkggAAAAAAAAAAAAAAQIX/
woAQwBoAGUAYwBrACAAQwBlAGwAbAAAAAcAAQAMAAX/AAClpaX/BQAMAAcAAAD/////
JQAFAAIGAA4ABf8AAD8/P/8GAAcADgAF/wAAPz8//wYACAAOAAX/AAA/Pz//BgAJAA4ABf8AAD8/
P/8GAJMCBAArgAP/kgggAJIIAAAAAAAAAAAAAAAEFA/8FAEMAbwBtAG0AYQAAAAAkwIEACyABv+SCC
gAkggAAAAAAAAAAAAAAQUG/wkAQwBvAG0AbQBhACAAWwAwAF0AAAAAJMCBAAtgAT/
kggmAJIIAAAAAAAAAAAAAAAEFBP8IAEMAdQByAHIAZQBuAGMAeQAAAAAkwIEAC6AB/
+SCC4AkggAAAAAAAAAAAAAAQUH/wwAQwB1AHIAcgBlAG4AYwB5ACAAWwAwAF0AAAAAJMCFQAvABAA
AEV4cGxhbmF0b3J5IFRleHSSCEcAkggAAAAAAAAAAAAAQI1/
xAARQB4AHAAbABhAG4AYQB0AG8AcgB5ACAAVABlAHgAdAAAAAIABQAMAAX/AAB/f3//
JQAFAAKTAgkAMAAEAABHb29kkgg7AJIIAAAAAAAAAAAAAAEBGv8EAEcAbwBvAGQAAAADAAEADAAF/
wAAxu/O/wUADAAF/wAAAGEA/yUABQACkwIOADEACQAASGVhZGluZyAxkghHAJIIAAAAAAAAAAAAAAAE
DEP8JAEgAZQBhAGQAaQBuAGcAIAAxAAAAAwAFAAwABwMAAERUav8lAAUAAgcADgAHBAAARHLE/wUAk
wIOADIACQAASGVhZGluZyAykghHAJIIAAAAAAAAAAAAAAAEDEf8JAEgAZQBhAGQAaQBuAGcAIAAyAAA
AAwAFAAwABwMAAERUav8lAAUAAgcADgAHBP8/orjh/wUAkwIOADMACQAASGVhZGluZyAzkghHAJIIA
AAAAAAAAAAAAAEDEv8JAEgAZQBhAGQAaQBuAGcAIAAzAAAAAwAFAAwABwMAAERUav8lAAUAAgcADgA
HBDIzjqnb/wIAkwIOADQACQAASGVhZGluZyA0kgg5AJIIAAAAAAAAAAAAAAEDE/8JAEgAZQBhAGQAa
QBuAGcAIAA0AAAAAgAFAAwABwMAAERUav8lAAUAApMCCgA1AAUAAElucHV0kgh1AJIIAAAAAAAAAAA
AAAECFP8FAEkAbgBwAHUAdAAAAAcAAQAMAAX/AAD/zJn/BQAMAAX/AAA/P3b/
JQAFAAIGAA4ABf8AAH9/f/8BAAcADgAF/wAAf39//wEACAAOAAX/AAB/f3//AQAJAA4ABf8AAH9/f/8/

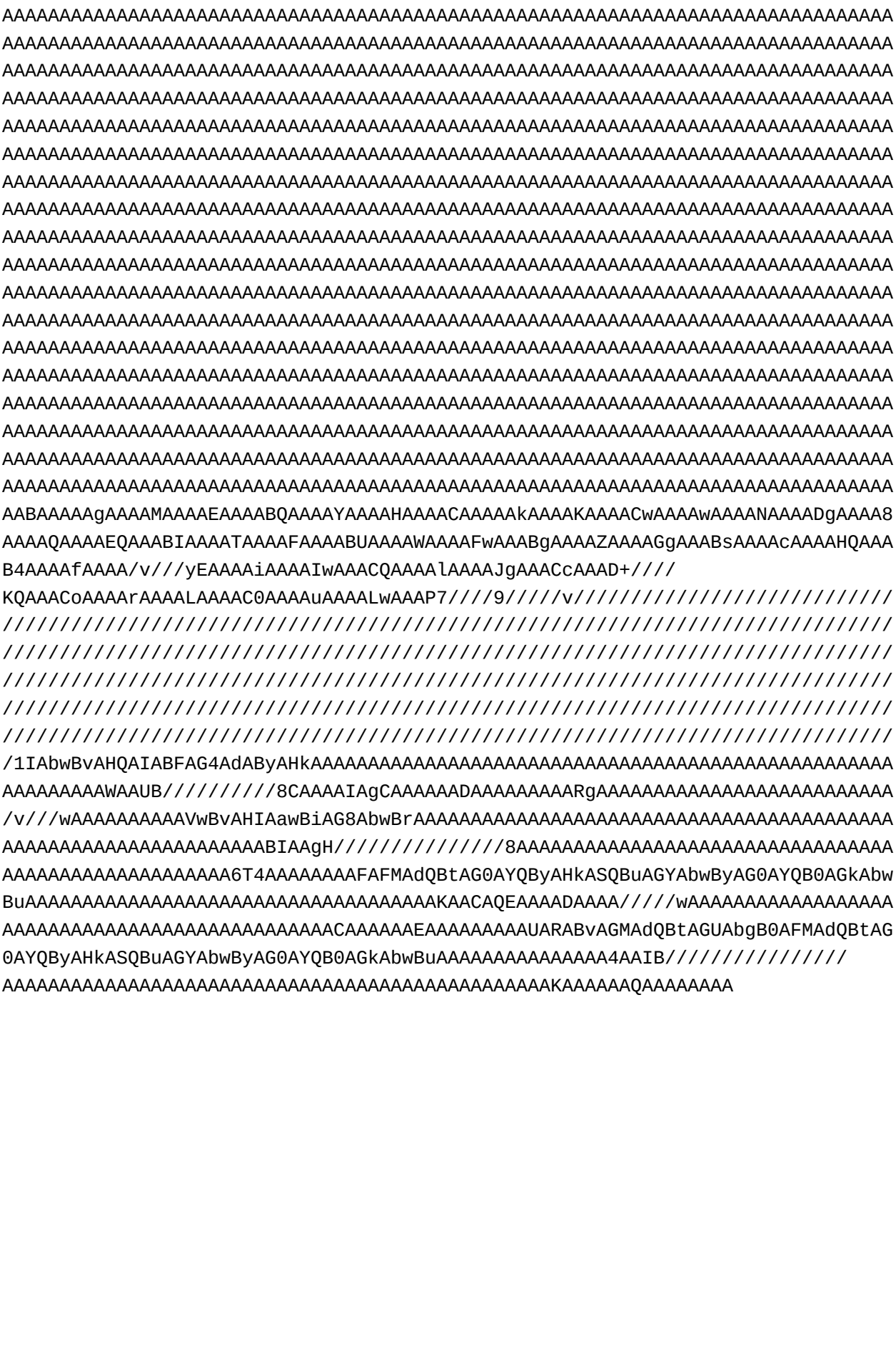
8BAJMCEAA2AAsAAExpbmtlZCBDZWxskghLAJIIAAAAAAAAAAAAAAECGP8LAEwAaQBuAGsAZQBkACAA
QwBlAGwAbAAAAAMABQAMAAX/AAD6fQD/
JQAFAAIHAA4ABf8AAP+AAf8GAJMCDAA3AAcAAE5ldXRyYWySCEEAkggAAAAAAAAAAAAAAQEc/
wcATgBlAHUAdAByAGEAbAAAAAMAAQAMAAX/AAD/65z/BQAMAAX/AACcVwD/
JQAFAAKTAgQAAIAA/5IIMwCSCAAAAAAAAAAAAAABAQD/
BgBOAG8AcgBtAGEAbAAAAAIABQAMAAcBAAAAAAD/
JQAFAAKTAgkAOAAEAABOb3RlkghiAJIIAAAAAAAAAAAAAAECCv8EAE4AbwB0AGUAAAAFAAEADAAF/
wAA///M/wYADgAF/wAAsrKy/wEABwAOAAX/AACysrL/AQAIAA4ABf8AALKysv8BAAkADgAF/
wAAsrKy/wEAkwILADkABgAAT3V0cHV0kgh3AJIIAAAAAAAAAAAAAAECFf8GAE8AdQB0AHAAdQB0AAA
ABwABAAwABf8AAPLy8v8FAAwABf8AAD8/P/8lAAUAAgYADgAF/wAAPz8//wEABwAOAAX/AAA/Pz//
AQAIAA4ABf8AAD8/P/8BAAkADgAF/wAAPz8//wEAkwIEADqABf+SCCQAkggAAAAAAAAAAAAAAQUF/
wcAUABlAHIAYwBlAG4AdAAAAAAkwIKADsABQAAVGl0bGWSCDEAkggAAAAAAAAAAAAAAQMP/wUAVAB
pAHQAbABlAAAAAgAFAAwABwMAAERUav8lAAUAAZMCCgA8AAUAAFRvdGFskghNAJIIAAAAAAAAAAAAA
AEDGf8FAFQAbwB0AGEAbAAAAAQABQAMAAcBAAAAAD/
JQAFAAIGAA4ABwQAAERyxP8BAAcADgAHBAAARHLE/wYAkwIRAD0ADAAAV2FybmluZyBUZXh0kgg/AJ
IIAAAAAAAAAAAAAAECC/8MAFcAYQByAG4AaQBuAGcAIABUAGUAeAB0AAAAAgAFAAwABf8AAP8AAP8l
AAUAAo4IWACOCAAAAAAAAAAAAAACQAAAAEQARAFQAYQBiAGwAZQBTAHQAeQBsAGUATQBlAGQAaQB1AG
0AMgBQAGkAdgBvAHQAUwB0AHkAbABlAEwAaQBnAGgAdAAxADYAYAECAAAAhQAOAGE7AAAAAAYAU2hl
ZXQxhQAOACU9AAAAAAYAU2hlZXQymggYAJoIAAAAAAAAAAAAAAAEAAAAAAAAACAAAAKMIEACjCAAAAA
AAAAAAAAAAAAAAjAAEAAEAAQDBAQgAwQEAACXDDgD8ABkABAAAAAIAAAAEAAB0ZXN0BwAAY29udGVu
dP8ACgAIAKEuAAAMAAAAYwgWAGMIAAAAAAAAAAAAABYAAAAAAAAAAgCWCFcMlggAAAAAAAAAAAADY
wCAFBLAwQUAAYACAAAACEA6d4Pv/8AAAAcAgAAEwAAAFtDb250ZW50X1R5cGVzXS54bWyskctOwzAQ
RfdI/IPlLUqcskAIJemCx47HonzAyJkkFsnYsqdV+/dM0lRCqCAWbCzZM/eeO+NyvR8HtcOYnKdKr/
JCKyTrG0ddpd83T9mtVomBGhg8YaUPmPS6vrwoN4eASYmaUqV75nBnTLI9jpByH5Ck0vo4Ass1diaA
/YAOzXVR3BjriZE448lD1+UDtrAdWD3u5fmYJOKQtLo/Nk6sSkMIg7PAktTsqPlGyRZCLsq5J/
UupCuJoc1ZwlT5GbDoXmU10TWo3iDyC4wSw7AMiV/PZyAZLea/
O56J7NvWWWy83Y6yjnw2XsxOwf8UYPU/6BPTzH9bfwIAAP//AwBQSwMEFAAGAAgAAAAhAKXWp+fAAA
AANgEAAAsAAABfcmVscy8ucmVsc4SPz2rDMAyH74W9g9F9UdLDGCV2L6WQQy+jfQDhKH9oIhvbG+vb
T8cGCrsIhKTv96k9/q6L+eGU5yAWmqoGw+JDP8to4XY9v3+CyYWkpyUIW3hwhqN727VfvFDRozzNMR
ulSLYwlRIPiNlPvFKuQmTRyRDSSkXbNGIkf6eRcV/XH5ieGeA2TNP1FlLXN2Cuj6jJ/7PDMMyeT8F/
ryzlRQRuN5RMaeRioagv41O9kKhlqtQe0LW4+db9AQAA//8DAFBLAwQUAAYACAAAACEAa3mWFoMAAA
CKAAAAHAAAAHRoZW1lL3RoZW1lL3RoZW1lTWFuYWdlci54bWwMzE0KwyAQQOF9oXeQ2TdjuyhFYrLL
rrv2AEOcGkHHoNKf29fl44M3zt8U1ZtLDVksnAcNimXNLoi38Hwspxuo2kgcxSxs4ccV5ul4GMm0jR
PfSchzUX0j1ZCFrbXdINa1K9Uh7yzdXrkkaj2LR1fo0/cp4kXrKyYKAjj9AQAA//8DAFBLAwQUAAYA
CAAAACEATtJt0c8GAACwHwAAFgAAAHRoZW1lL3RoZW1lL3RoZW1lMS54bWzsWUtvGzcQvhfof1fAA
Fk6xEZkQPrFSexbCNSUuRISdQuLe5yQVJ2dCuSUy8FCqRFLwV666EoGqABGvTSH2MgQZv+iA650i4p
UfEDRpEWtgBDS30z/DgzOzM7e+fus4h6J5gLwuK6X7xV8D0cD9mIxEHdf9zvfHbb94RE8QhRFuO6P8
PCv7vz6Sd30LYMcYQ9kI/FNqr7oZTJ9saGGMIyErdYgmP4bcx4hCRc8mBjxNEp6I3oxmahUNmIEIl9
L0YRqD0cj8kQe32l0t9ZKG9TuIylUAtDyntKNbYkNHY0KSqEmIkm5d4JonUf9hmx0z5+Jn2PIiHhh7
pf0H/+xs6dDbQ9F6Jyjawh19F/c7m5wGiyqffkwSDbtFQqlyq7mX4NoHIV1662K+1Kpk8D0HAIJ025
2Dqrm83SHGuA0q8O3a1qa6to4Q39Wyucd8vqY+E1KNVfWsF3Ok2wooXXoBRfXsGXG7VGy9avQSm+so
KvFnZbpaqlX4NCSuLJCrpQrmw1F6fNIGNG95zwWrnUqW7OlecoiIYsutQWYxbLdbEWoWPGOwBQQIok
iT05S/AYDSGKm4iSASfePglCCLwExUzAcmGz0ClswX/1Kelv2qNoGyNDWvECJmJlSfHxxJCTRNb9B6
DVNyBv37w5e/767PlvZy9enD3/Zb63VmXJ7aE4MOXe//j1399/4f316w/

vX36Tbr2MFyb+3c9fvvv9jw+phxPnpnj77at3r1+9/
e6rP3966dC+y9HAhPdJhIV3gE+9RyyCAzr44wG/nEQ/RMSSQCHodqhuy9ACHswQdeEa2DbhEw5ZxgW
8Nz22uPZCPpXEsfPDMLKAXcZog3GnAR6qvQwL96dx4N6cT03cI4ROXHs3UWw5uD1NIL0Sl8pmiC2aR
xTFEgU4xtJTv7EJxo7TPSXEsmuXDDkTbCy9p8RrIOI0SZ8MrEDKhfZIBH6ZuQiCqy3bdJ94DUZdp27
hExsJtwWiDvJ9TC0z3kNTiSKXyj6KqGnwfSRDF8nejA9NXFtI8HSAKfPaIyyES+aQw3kNpz+EDON2e
5fOIhvJJZm4dO4jxkxki02aIYoSF7ZH4tDE3hcTCFHkHTHpgneZfYeoa/ADite6+wnBlrvPTwSPIbm
alPIAUb9MucOX9zCz4rc3o2OEXVlml0dWdt3lxBkdjWlghfY+xhSdohHG3uP7DgYNllg2z0k/CCGr7
GFXYD1Adqyq6xgLaJNUX7OaIveJsEK2hwO2hk93tpR4ZiiOEF+n+QC8btq8DVXOmUoP6XBiAg8ItH8
QL06jHArQYQT3Wq1HIbJql7oW7nidcct/
F7nH4L48tmhc4L4EGXxpGUjspswHbdNH1NogD5g+ggbDlW5BxHJ/
LqLqqhabOuXG9k2buwEaI6vfiUh8bvOz1PaU/522x3E3XE/D41ZspaxLtjrrUsreUoOzDvcfbGtaaB
ofYagkqznrpqu56Wr8/31Xs+5evull1nUcN72MDz3GTS8zn6xcTy+Tty/Q2ahpRzrl0TOfaO3IZ0wo
7ckZxftCT30EPNGMOrCo5PS4E2cjwCSEr6rMwQYWLuBIy3icyc+JDHshSmA0VPSVkkDMVQfCS5iAiZ
FedupWeDqNumyUTjqLRTXVTCurQDJfL5SzdZhSyRRdqebTu0y9ZhvoKeuCgJK9DAljM5vEloNEdbGo
jKRnumA0Bwl9smthUXOwuK3UL1y1wgKoZV6BR24PHtTrfrkEIiAEwzhoz0fKT6mrF97VzrxOT68zph
UB0GIvIiD3dE1xXXs8dbo01C7gaYuEEW42CW0Z3eCJEB6E59GpVi9C47K+ruUutegpU+j9ILRyGtXb
H2JxVV+D3HJuoLGZKWjsndb9ylYZQmaIkro/hokxfI0SiB2hnroQDeC1y1Dy9Ia/SmZJuJAtJMLU4D
rppNkgIhJzj5Ko7qvjZ26gsc4hmltxExLCR0uuBmnlYyMHTredjMdjPJSm240VZen0EjJ8miucv2rx
q4OVJJuCu3vh6NQb0Cl/hCDEytWiMuCICHhxUEytOSLwJixLZHn8LRWmedo1X0XpGErXEU1CNK8oZj
JP4TqVZ3T0VWYD42p+ZjCoYZJ5IRwEqsCaRrWqaVY1Ug5rq+75QspyRtLMa6aVVVTVdGcxa4dFGViy
5dWKvMFqYWLIaWaFT1P3csqtLXLdUp+QVQkweGY/R9W9QEEwqOWbWdQU49U0rHL2fNWuHYsDnkPtIk
XCyPqVhdolu2U1wrkdLF6p8oPcctTC0njRV2pL61fm5lttNjiG5NGCLndKpdCuhNkuR9AQ9XRPkqUN
LbrzDwAAAP//AwBQSwMEFAAGAAgAAAAhAA3RkJ+2AAAAGwEAACcAAAB0aGVtZS90aGVtZS9fcmVscy
90aGVtZU1hbmFnZXIueG1sLnJlbHOEj00KwjAUhPeCdwhvb9O6EJEm3YjQrdQDhOQ1DTY/JFHs7Q2u
LAguh2G+mWm7l53JE2My3jFoqhoIOumVcZrBbbjsjkBSFk6J2TtksGCCjm837RVnkUsoTSYkUiguMZ
hyDidKk5zQilT5gK44o49W5CKjpkHIu9BI93V9oPGbAXzFJL1iEHvVABmWUJr/s/04GolnLx8WXf5R
QXPZhQUoosbM4CObqkwEylu6usTfAAAA//8DAFBLAQItABQABgAIAAAAIQDp3g+//wAAABwCAAATAA
AAAAAAAAAAAAAAAAAAAABbQ29udGVudF9UeXBlc10ueG1sUEsBAi0AFAAGAAgAAAAhAKXWp+fAAAAA
NgEAAAsAAAAAAAAAAAAAAAAAAAAMAEAAF9yZWxzLy5yZWxzUEsBAi0AFAAGAAgAAAAhAGt5lhaDAAAAig
AAABwAAAAAAAAAAAAAAAAAAAGQIAAHRoZW1lL3RoZW1lL3RoZW1lTWFuYWdlci54bWxQSwECLQAUAAYA
CAAAACEATtJt0c8GAACwHwAAFgAAAAAAAAAAAAAAAADWAgAAdGhlbWUvdGhlbWUvdGhlbWUxLnhtbF
BLAQItABQABgAIAAAAIQAN0ZCftgAAABsBAAAnAAAAAAAAAAAAAAAAANkJAAB0aGVtZS90aGVtZS9f
cmVscy90aGVtZU1hbmFnZXIueG1sLnJlbHNQSwUGAAAAAAUABQBdAQAA1AoAAAAAmwgQAJsIAAAAAA
AAAAAAAAEAAACMCBAAjAgAAAAAAAAAAAAAAAAAAoAAAAJCBAAAAYQAFpPzQfBAAIABggAAAsCFAAA
AAAAAAAAAAEAAAB3PAAAvzwAAA0AAgABAAwAAgBkAA8AAgABABEAAgAAABAACAD8qfHSTWJQP18AAg
ABACoAAgAAACsAAgAAAIIAAgABAIAACAAAAAAAAAAAACUCBAAAAEABgQACAMEEFAAAABUAAACDAAIA
AACEAAIAAAAmAAgAZmZmZmZm5j8nAAgAZmZmZmZm5j8oAAgAAAAAAAA6D8pAAgAAAAAAAA6D+hAC
IAAwBAAQEAAQABAAQADYwCADMzMzMzM9M/MzMzMzMz0z8Pv5wIJgCcCAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAPDMAAAAAAAAAFUAAgAKAAACDgAAAAAAQAAAAAAAgAAAAgCEAAAAAAAAgBAAQAAAA
AAAQ8A/QAKAAAAAAAPAAAAAAD9AAoAAAABAA8AAQAAANcABgAwAAAAAAA+AhIAtgYAAAAQAAAAAAA
AAAAAAAAiwgQAIsIAAAAAAAAAAAAAAAAAAgAdAA8AAxMABQAAAAEAEwATAAUFZwgXAGcIAAAAAAAAAA
AAAAIAAf////8DRAAACgAAAAkIEAAABhAAWk/NB8EAAgAGCAAACwIUAAAAAAAAAAAAAQAAADs+AACD
PgAADQACAAEADAACAGQADwACAAEAEQACAAAAEAAIAPyp8dJNYlA/XwACAAEAKgACAAAAKwACAAAAgg
ACAAEAgAAIAAAAAAAAAAAAJQIEAAAAQAGBAAIAwQQUAAAAFQAAAIMAAgAAAIQAAgAAACYACABmZmZm



AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
P7/AAANBAEAAAAAAAAAAAAAAAAAAAAAAAEAAAAC1c3VnC4bEJOXCAArLPmuMAAAANAAAAAJAAAAAQA
AAFAAAAAPAAAAWAAAABcAAABkAAAACwAAAGwAAAAQAAAAdAAAABMAAAB8AAAAFgAAAIQAAAANAAAAj
AAAAAwAAACqAAAAAgAAABAnAAAeAAAABAAAAAAAAAADAAAAAAAQAAsAAAAAAAAACwAAAAAAAAALAAA
AAAAAAAsAAAAAAAAAHhAAAAIAAAAHAAAAU2hlZXQxAAcAAABTaGVldDIADBAAAAIAAAAeAAAACwAAA
FdvcmtzaGVldHMAAwAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA



langchain4j\src\test\resources\test-file.xlsx

BASE64:

UEsDBBQABgAIAAAAIQASGN7dZAEAABgFAAATAAgCW0NvbnRlbnRfVHlwZXNdLnhtbCCiBAIooAACAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADElM9uwjAMxu+T9g5VrlMb
4DBNE4XD/hw3pLEHyBpDI9Ikig2Dt58bYJqmDoRA2qVRG/v7fnFjD8frxmYriGi8K0W/6IkMXOW1cf
NSvE+f8zuRISmnlfUOSrEBFOPR9dVwugmAGWc7LEVNFO6lxKqGRmHhAzjemfnYKOLXOJdBVQs1Bzno
9W5l5R2Bo5xaDTEaPsJMLS1lT2v+vCWJYFFkD9vA1qsUKgRrKkVMKldO/3LJdw4FZ6YYrE3AG8YQst
Oh3fnbYJf3yqWJRkM2UZFeVMMYcm3lp4+LD+8XxWGRDko/
m5kKtK+WDVegwBBBaawBqLFFWotGGbfnPuCfglGmpX9hkPZ8SfhEjsE/
cRDfO5DpeX4pksyRgyNtLOClf38SPeZcqwj6jSJ36MUBfmof4uD7O4k+IHdyhNOrsG/
VNjsPLASRDHw3a9el/3bkKXB22aGdMxp0h7dMc230BQAA//8DAFBLAwQUAAYACAAAACEAtVUwI/QAA
ABMAgAACwAIAl9yZWxzLy5yZWxzIKIEAiigAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAKySTU/DMAyG70j8h8j31d2QEEJLd0FIuyFUfoBJ3A+1jaMkG92/
JxwQVBqDA0d/vX78ytvdPI3qyCH24Gl/pxdQcqJnKWRnGs4cQRdtX11faZR0p5KH
a9jyqruKihS8nfI0bT8USxEM8uVxoJE6UchhY9mYFaxk1Z3mL4rgHVQlPtrYawtzeg6pPPm3/
XlqbpDT+IOUzs0pkVyHNiZ9mufMhsIfX5GlVTaDlpsGKecjoieV9kbMDzRJu/
E/18LU6cyFIiNBL4Ms9HxyWg9X9atDTxy515xDcJw6vI8MmCix+o3gEAAP//AwBQSwMEFAAGAAgAAA
AhAEqppmH6AAAARwMAABoACAF4bC9fcmVscy93b3JrYm9vay54bWwucmVscyCiBAEooAABAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAALySzWrEMAyE74W+g9G9cZL+UMo6eymFvbbbBzCxEodNbGOpP3n3n3n3n7mpTuNr
Ckl9CjJDTzMcxm+zn04h0jdd4pKLIcBLram861Cl73T1f3IIi1M7r3DhWMSLCtLi82z9hrTk9ku0Ai
qThSYJnDg5RUWxw0ZT6gS5fGx0FzGmMrg64PukVZ5vmdjL81oJppip1REHfmGsR+DMn5b23fNF2Nj7
5+G9DxGQvJiQuToI4tsoJp/F4WWQIFeZ6hXJPhw8cDWUQ+cRxXJKdLuQRT/

DPMYjK3a8KQ1RHNC8dUPjqlM1svJXOzKgyPfer6sSs0zT/2clb/6gsAAP//
AwBQSwMEFAAGAAgAAAAhANnBlct9AwAA8AgAAA8AAAB4bC93b3JrYm9vay54bWysVdFuozgUfV9p/
wHxTrEJJgSVjiCAtlI7qtJMO/NUuWCKVcBZY5pU1fz7XJOQtpPRKNvZKLGxfXN8ju+55vTTpqmNJyY
7LtrQxCfINFibi4K3D6H5ZZlZvml0irYFrUXLQvOZdeans7//
Ol0L+XgvxKMBAG0XmpVSq8C2u7xiDe1OxIq1sFIK2VAFQ/lgdyvJaNFVjKmmth2EPLuhvDW3CIE8Bk
OUJc9ZIvK+Ya3agkhWUwX0u4qvuhGtyY+Ba6h87FdWLpoVQNzzmqvnAdQ0mjw4f2iFpPc1yN5gYmwk
fD34YQSNM+4ESwdbNTyXohOlOgFoe0v6QD9GNsbvjmBzeAbHIbm2ZE9c53DPSnofZOXtsbxXMIz+GA
2DtQavBHB4H0Qje26OeXZa8prdbK1r0NXqM210pmrTqGmn0oIrVoTmFIZizd5NyH4V97yGVYwIdk37
bG/nK2kUrKR9rZZg5BEeAj1v5hAdCcaIasVkSxWbi1aBD3e6/tRzA/a8EuBwY8H+7blkUFjgL9AKLc
0Det9dUVUZvaxD0/7SgXioOPFsjxXR2W9sSQ9r4D8Yk+ZarQ1yt5S2zz9LB2YyGM13paQBz+fJBSTg
mj5BOiDpxa5az/V5T+7aXAb47mWKHT9GhFgomrmWj13HilycWLEfEZLOUyeep99BjPSCXNBeVbtMa+
jQdCGtB0uXdDOuYBT0vHil8YJ2H0v3PzXj2nctWN9pN5ytu1dP6KGxueVtIdYgARMfVD2PY+IiGK6H
1VteqCo0HR+5+7l/GH+ogDKeunoSzK+pheYLSRDyUTq1nMzLLDJzZ9bMI57lud40jiM/cZLJQMl+w2
m4PoHb0BvtYPlrfaViuKd1P5yyachA7yHPC6xFHUTD7bWPhud9tDPkfNwkp3UOBaG7AdbHyJnpCLZR
F50aevAiBzHYRdEUQSZROiGW688cy3cnjjV3Eycl0zRJY6LTqV8Wwf9xZQ4lEYxvIc2yolItJc0f4d
21YGVMO/DfIN8Gvm/JxgSsNwGKboYzy8UzZMWx51okySZkipN5SrJXslp+
+cELy7eHfzOqeihmXcfDONBttpvdT5bbiV1W35VqsEj0ue/+/
bvAa1BfsyODs5sjA+efL5eXR8ZepMu72+zY4OgyTqLj46PFIvq2TL+OW9i/PNBtwnU72NQebXL2AwA
A//8DAFBLAwQUAAYACAAAACEA8AhY9KUCAABSBgAADQAAAHhsL3N0eWxlcy54bWykVW1r2zAQ/
j7YfxD67sp24ywJtsvS1FDoxqAd7Ktiy4moXowkZ87G/vtOdl4cOrbRfolO59Nzz91zUtKbTgq0Y8Z
yrTIcXYUYMVXqiqtNhr8+FcEMI+uoqqjQimV4zyy+yd+/S63bC/a4ZcwhgFA2w1vnmgUhttwySe2Vb
piCL7U2kjrYmg2xjWG0sv6QFCQOwymRlCs8ICxk+T8gkprntglKLRvq+JoL7vY9FkayXNxvlDZ0LYB
qF01oibpoamLUmWOS3vsij+Sl0VbX7gpwia5rXrKXdOdkTmh5RgLk1yFFCQnji9o780qkCTFsx718O
E9rrZxFpW6VAzGBqG/B4lnp76rwn7xziMpT+wPtqABPjEmellpogxxIB52LvEdRyYaIWyr42nDvrKn
kYj+4+3O92oc4yaH3Pop4HofFwiEuxIlV7AmAI09BPseMKmCDDvbTvoH0CiZtgOnj/
hG9MXQfxcnoAOkT5ulamwom+9yPoytPBasdEDV8s/Wr0w38rrVzoH6eVpxutKLClzKAnAwop2RCPPr
p/1ZfYHc1Uq0spLuvMgz3yDfhaEIhB3PAGzYef4w2YL8ZFnX1JT4gjmhfkD6lR17vDH/211XA5Bwg0
LrlwnH1B8KAWXXnFoReAeevXt+cUxboRMVq2gr3dPqY4bP9iVW8lfEp6gvfaddDZPhsP3iloqnPwTr
3YGG8YEWt4Rn+ebf8MF/dFXEwC5ezYHLNkmCeLFdBMrldrlbFPIzD21+jB+AN179/r/IULtbCCngkz
KHYQ4mPZ1+GR5uBfj+jQHvMfR5Pw49JFAbFdRgFkymdBbPpdRIUSRSvppPlXVIkI+7JK5+JkETR8OB
48snCcckEV0etjgqNvSASbP9SBDkqQc5/BvlvAAAA//8DAFBLAwQUAAYACAAAACEAwRcQvk4HAADGI
AAAEwAAAHhsL3RoZW1lL3RoZW1lMS54bWzsWc2LGzzd/
w1448l3uDPbJDtJGSdlBy1tuxRVjMykrwbEwIlOfVSKKSll0JvPZTSQAMNvfSPCSS06R/
RJ83YI63lJJtsSlp2DYtH/r2np/eefnrzdPHSvZh6R5gLwpKWX75Q8j2cjNiYJNOWf2s4KDR8T0iUj
BFlCW75Cyz8S9uffnIRbckIx9gD+URsoZYfSTnbKhbFCIaRuMBmOIHfJozHSMIjnxbHHB2D3pgWK6V
SrRgjkvhegmJQe30yISPsDZVKf3upvE/
hMZFCDYwo31eqsSWhsePDskKIhehS7h0h2vJhnjE7HuJ70vcoEhJ+aPkl/
ecXty8W0VYmROUGWUNuoP8yuUxgfFjRc/LpwWrSIAiDWnulXwOoXMf16/1av7bSpwFoNIKVprbYOuu
VbpBhDVD61aG7V+9Vyxbe0F9ds7kdqo+F16BUf7CGHwy64EULr0EpPlzDh51mp2fr16AUX1vD10vtX
lC39GtQRElyuIYuhbVqd7naFWTC6I4T3gyDQb2SKc9RkA2r7FJTTFgiN+VajO4yPgCAAlIkSeLJxQx
P0AiyuIsoOeDE2yXTCBJvhhImYLhUKQ1KVfivPoH+piOKtjAypJVdYIlYG1L2eGLEyUy2/
Cug1TcgL549e/7w6fOHvz1/9Oj5w1+yubUqS24HJVNT7tWPX//9/RfeX7/+8OrxN+nUJ/
HCxL/8+cuXv//xOvWw4twVL7598vLpkxffffXnT48d2tscHZjwIYmx8K7hY+8mi2GBDvvxAT+dxDBC
xJJAEeh2qO7LyAJeWyDqwnWw7cLbHFjGBbw8v2vZuh/xuSSOma9GsQXcY4x2GHc64Kqay/DwcJ5M3Z

PzuYm7idCRa+4uSqwA9+czoFfiUtmNsGXmDYoSiaY4wdJTv7FDjB2ru0OI5dc9MuJMsIn07hCvg4jT
JUNyYCVSLrRDYojLwmUghNryzd5tr8Ooa9U9fGQjYVsg6jB+iKnlxstoLlHsUjlEMTUdvotk5DJyf8
FHJq4vJER6iinz+mMshEvmOof1GkG/
CgzjDvseXcQ2kkty6NK5ixgzkT122I1QPHPaTJLIxH4mDiFFkXeDSRd8j9k7RD1DHFCyMdy3CbbC/W
YiuAXkapqUJ4j6Zc4dsbyMmb0fF3SCsItl2jy22LXNiTM7OvOpldq7GFN0jMYYe7c+c1jQYTPL57nR
VyJglR3sSqwryM5V9ZxgAWWSqmvWKXKXCCtl9/GUbbBnb3GCeBYoiRHfpPkaRN1KXTjlnFR6nY4OTe
A1AuUf5IvTKdcF6DCSu79J640IWWeXehbufF1wK35vs8dgX9497b4EGXxqGSD2t/
bNEFFrgjxhhggKDBfdgogV/lxEnatabO6Um9ibNg8DFEZWvROT5I3Fz4myJ/
x3yh53AXMGBY9b8fuUOpsoZedEgbMJ9x8sa3pontzAcJKsc9Z5VXNe1fj/
+6pm014+r2XOa5nzWsb19vVBapm8fIHKJu/
y6J5PvLHlMyGU7ssFxbtCd30EvNGMBzCo21G6J7lqAc4i+Jo1mCzclCMt43EmPycy2o/QDFpDZd3An
IpM9VR4MyagY6SHdSsVn9Ct+07zeI+N005nuay6mqkLBZL5eClcjUOXSqboWj3v3q3U637oVHdZlwY
o2dMYYUxmG1F1GFFfDkIUXmeEXtmZWNF0WNFQ6pehWkZx5QowbRUVeOX24EW95YdB2kGGZhyU52MVp
7SZvIyuCs6ZRnqTM6mZAVBiLzMgj3RT2bpxeWp1aaq9RaQtI4x0s40w0jCCF+EsO82W+1nGupmH1DJ
PuWK5G3Iz6o0PEWtFIie4gSYmU9DEO275tWoItyojNGv5E+gYw9d4Brkj1FsXolO4dhlJnm74d2GWG
Reyh0SUOlyTTsoGMZGYe5TELV8tf5UNNNEcom0rV4AQPlrjmkArH5txEHQ7yHgywSNpht0YUZ5OH4H
hU65w/qrF3x2sJNkcwr0fjY+9AzrnNxGkWFgvKweOiYCLg3LqzTGBm7AVkeX5d+JgymjXvIrSOZSOI
zqLUHaimGSewjWJrszRTysfGE/
ZmsGh6y48mKoD9r1P3Tcf1cpzBmnmZ6bFKurUdJPphzvkDavyQ9SyKqVu/
U4tcq5rLrkOEtV5Srzh1H2LA8EwLZ/MMk1ZvE7DirOzUdu0MywIDE/UNvhtdUY4PfGuJz/IncxadUA
s60qd+PrK3LzVZgd3gTx6cH84p1LoUEJvlyMo+tIbyJQ2YIvck1mNCN+8OSct/34pbAfdStgtlBphv
xBUg1KhEbarhXYYVsv9sFzqdSoP4GCRUVwO0+v6AVxh0EV2aa/H1y7u4+UtzYURi4tMX8wXteH64r5
c2Xxx7xEgnfu1yqBZbXZqhWa1PSgEvU6j0OzWOoVerVvvDXrdsNEcPPC9Iw0O2tVuUOs3CrVyt1sIa
iVlfqNZqAeVSjuotxv9oP0gK2Ng5Sl9ZL4A92q7tv8BAAD//wMAUEsDBBQABgAIAAAAIQAPEi/
MbwIAAGIEAAAYAAAAeGwvd29ya3NoZWV0cy9zaGVldDEueG1snJNbb6MwEIXfV9r/
gPxOHAhhAwqptKJo+7baS/tszBCs+MLazk2r/e8diJJWyktUCaRhsL8zBx/mD0clgz1YJ4wuSDQakw
A0N7XQm4L8+b0OZyRwnumaSaOhICdw5GHx9cv8YOzWtQA+QIJ2BWm973JKHW9BMTcyHWh80xirmMdH
u6Gus8DqYZOSNB6PU6qY0ORMyO09DNM0gkNp+E6B9meIBck8zu9a0bkLTfF7cIrZ7a4LuVEdIiohhT
8NUBIonj9vtLGskuj7GCWMB0eLV4z35CIz9G+UlODWONP4EZLpeeZb+xnNKONX0q3/
uzBRQi3sRX+A76j4cyNF0ysrfodNPglLr7D+c9l8J+qC/JvEyfRpnSXhajadhXGWlOEqK9MwW5XJ7N
vkaZmmy/9kMa8FnnDvKrDQFGQZ5auI0MV8yM+LgIP7UAeeVb9AAveAGhEJ+nhWxmz7hc/
YGiPRDQt6IuNe7OERpCxIGSUY8b+DSF+jBL1qfKwveush0j9sUDEHj0a+itq3KIq/
Tg0N20n/0xy+g9i0HrspWu+zk9enEhzH0OIwoxhl3gAAAP//AAAA//+yKc5ITS1xSSxJtLMpyi9XKL
JVMlRSKC5IzCsGsqyMlBQqDE0Sk61SKl1Si5NT80pslQz0jJTsbJJBSh2BaoEixUB+mZ2BjX6ZnY1+
MlTOCVnOEC6nD7QFqAphLQAAAP//AAAA//+yKUhMT/VNLErPzCtWyElNK7FVMtAzV1IoykzPgLFL8g
vAoqZKCkn5JSX5uTBeRmpiSmoRiGespJCWn18C4+jb2eiX5xdlF2ekppbYAQAAAP//AwBQSwMEFAAG
AAgAAAAhABEFfjykAAAAywAAABQAAAB4bC9zaGFyZWRTdHJpbmdzLnhtbEyOQQrCMBBF94J3CLO3qU
VEJEkXgifQA4R2bAPNpGamorc30o3L9z4PvmnfcVIvzBwSWdhXNSikLvWBBgv323V3AsXiqfdTIrTw
QYbWbTeGWVRpiS2MIvNZa+5GjJ6rNCOV5ZFy9FIwD5rnjL7nEVHipJu6PuroA4Hq0kJi4QBqofBc8L
JyA85wcEacIIvR4oz+8eq6RIL0p3U5474AAAD//wMAUEsDBBQABgAIAAAAIQCLquW5YgIAAFAEAAAY
AAAAeGwvd29ya3NoZWV0cy9zaGVldDIueG1snNNLi9swEIDvge2HEADge2H/
g9DdkZ9pbOws2w2meyul7Z4VeRyLSJYrKS/3vHDkkKuYQFG8sS/mbGGpXPR63IHqyTpyA4Hq0kJi4QBqofBc8L
JyA85wcEacIIvR4oz+8eq6RIL0p3U5474AAAD//wMAUEsDBBQABgAIAAAAIQCLquW5YgIAAFAEAAAY
AAAAeGwvd29ya3NoZWV0cy9zaGVldDIueG1snNNLi9swEIDvge2HEADge2H/
g9DdkZ9pbOws2w2meyul7Z4VeRyLSJYrKS/3vHDkkKuYQFG8sS/mbGGpXPR63IHqyTpyA4Hq0kJi4QBqofBc8L

Ip0aJ42/TG8rXCuo9RygU5WrxivJNLmGn+LpKWwhpnWj9DmZ1zvi8/Zznj4ird1/8QE6XMwl6OG3ij
4o+lFGVXK75hyQex+RUbf5ctdrKp6J86e6nD+ec8SJJFEmR1mgWLPFoF8ywNk1VeL9I0+UuXZSNxh8
eqiIW2oi9R8SWibFlO/
fNLwsH9NyZjO66N2Y4LbxgmRMGBAjE2BuH42MMrKFVRVIj7PZlnkF3FZXkbX/
R6auBvlqy5g1ej3mXjOzwpeFAaaPlO+e/m8BXkpvM4O8dCx04pmtMKnMAWxVRmMeb9DwAA//8AAAD/
/7IpzkhNLXFJLEm0synKL1coslUyVFIoLkjMKwayrIyUFCoMTRKTrVIqXVKLk1PzSmyVDPSMlOxskk
FKHYFqgSLFQH6ZnYGNfpmdjX4yVM4JWc4QLqcPtAWoCmEtAAAA//8AAAD//7IpSExP9U0sSs/MK1bI
SU0rsVUy0DNXUijKTM+AsUvyC8CipkoKSfklJfm5MF5GamJKahGIZ6ykkJafXwLj6NvZ6JfnF2UXZ6
SmltgBAAAA//8DAFBLAwQUAAYACAAAACEA/3lsfk4BAABjAgAAEQAIAWRvY1Byb3BzL2NvcmUueG1s
IKIEASigAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAjJJBSsQwGIX3gnco2bdJW5AxtB1QmZUDgiOKu5
D8M1Ns0pBEO7P3CoKHcOHGSwnewbSdqZVx4TL/e/ny3k+y6UZWwRMYW9YqR3FEUACK16JUqxzdLGbh
BAXWMSVYVSvI0RYsmhbHRxnXlNcGrkytwbgSbOBJylKuc7R2TlOMLV+DZDbyDuXFZW0kc/5oVlgz/s
BWgBNCTrAExwRzDLfAUA9EtEMKPiD1o6k6gOAYKpCgnMVxFOMfrwMj7Z8XOmXklKXbat9pF3fMFrwX
B/fGloOxaZqoSbsYPn+M7+aX113VsFTtrjigIhOccgPM1ab4fH/5en3+/
HjL8GjabrBi1s39spcliLPt2HgoemJXoMeCCHwk2hfYK7fp+cVihoqEJGlIJiFJF3FKCaFxct++/
et+G7EfyF2C/xMTSk5HxD2gyPDBtyi+AQAA//8DAFBLAwQUAAYACAAAACEAtphM8ZUBAAA2AwAAEAA
IAWRvY1Byb3BzL2FwcC54bWwgogQBKKAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACck0Fv2zAMhe8D9
h8M3Rs5WVEMgayiSFv0sKIBknZnTqZjobYkiKyR7NdXtlHX2Q4DdiP5Hp4+U7K6PrZN1mEk610hlot
cZOiML607FOJ5f3/xXWTE4EpovMNCnJDEtf76RW2jDxGayQlskuKZWPLXBq4
0H6qrIGb715a9GxXOX5lcQjoyuxvAhToBgT1x3/b2jpTc9HL/tTSMBa3YTQWAOcvlI/WhM9+YqzRzD
Wsac6uzsabJSc21Ti3KF5i5ZPOldy3qqdgQY36QhdQUOo5OdAPSD069uCjaRVx+sODfuYkf2dFrgS2
S8g7MEK0UG04DgB9raxGeomEEf908dXqhGZlEyGcTiUc++8tpd6NRhScW7sA0aQJJwj7i03SE/
VFiL/i3hgGHlHnF3Pt5zzTaSDNOLM0YdtJIg/
jt34NoA7JWGqflj3Ss9h72+B8WPT50O1qyFimS5nuolpoB7SkmPTh2xqcAcsPzx/C/0LeRl/
A728WuTf8nTls5mSnw9evwMAAP//AwBQSwECLQAUAAYACAAAACEAEhje3WQBAAAYBQAAEwAAAAAAAAA
AAAAAAAAAAAAAAW0NvbnRlbnRfVHlwZXNdLnhtbFBLAQItABQABgAIAAAAIQC1VTAj9AAAAEwCAAAL
AAAAAAAAAAAAAAAAAJ0DAABfcmVscy8ucmVsc1BLAQItABQABgAIAAAAIQBKqaZh+gAAAEcDAAAaAA
AAAAAAAAAAAAAAAMIGAAB4bC9fcmVscy93b3JrYm9vay54bWwucmVsc1BLAQItABQABgAIAAAAIQDZ
wZXLfQMAAPAIAAAPAAAAAAAAAAAAAAAAAPwIAAB4bC93b3JrYm9vay54bWxQSwECLQAUAAYACAAAAC
EA8AhY9KUCAABSBgAADQAAAAAAAAAAAAAAAACmDAAAeGwvc3R5bGVzLnhtbFBLAQItABQABgAIAAAA
IQDBFxC+TgcAAMYgAAATAAAAAAAAAAAAAAAAAHYPAAB4bC90aGVtZS90aGVtZTEueG1sUEsBAi0AFA
AGAAgAAAAhAA8SL8xvAgAAYgQAABgAAAAAAAAAAAAAAAAA9RYAAHhsL3dvcmtzaGVldHMvc2hlZXQx
LnhtbFBLAQItABQABgAIAAAAIQARBX48pAAAAMsAAAAUAAAAAAAAAAAAAAAAAJoZAAB4bC9zaGFyZW
RTdHJpbmdzLnhtbFBLAQItABQABgAIAAAAIQCLquW5YgIAAFAEAAAYAAAAAAAAAAAAAAAAAHAaAAB4
bC93b3Jrc2hlZXRzL3NoZWV0Mi54bWxQSwECLQAUAAYACAAAACEA/3lsfk4BAABjAgAAEQAAAAAAAAA
AAAAAAAAAIHQAAZG9jUHJvcHMvY29yZS54bWxQSwECLQAUAAYACAAAACEAtphM8ZUBAAA2AwAAEAAA

AAAAAAAAAAAAAACNHwAAZG9jUHJvcHMvYXBwLnhtbFBLBQYAAAAACwALAMYCAABYIgAAAAA=

langchain4j-azure-open-ai\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-azure-open-ai</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Azure OpenAI</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-open-ai</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-azure-open-ai\src\main\java\dev\langchain4j\model\azure\AzureOpenAiChatModel.java

```
package dev.langchain4j.model.azure;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.chat.ChatCompletionRequest;
import dev.ai4j.openai4j.chat.ChatCompletionResponse;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.chat.TokenCountEstimator;
import dev.langchain4j.model.output.Response;
import java.net.Proxy;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.*;
import static java.time.Duration.ofSeconds;
import static java.util.Collections.singletonList;
/**
 * Represents an OpenAI language model, hosted on Azure, that has a chat
completion interface, such as gpt-3.5-turbo.
 * <p>
 * Mandatory parameters for initialization are: baseUrl, apiVersion and
apiKey.
 * <p>
 * There are two primary authentication methods to access Azure OpenAI:
 * <p>
 * 1. API Key Authentication: For this type of authentication, HTTP requests
must include the
 * API Key in the "api-key" HTTP header.
 * <p>
 * 2. Azure Active Directory Authentication: For this type of authentication,
HTTP requests must include the
 * authentication/access token in the "Authorization" HTTP header.
 * <p>
 * <a href="https://learn.microsoft.com/en-us/azure/ai-services/openai/
reference">More information</a>
 * <p>
 * Please note, that currently, only API Key authentication is supported by
this class,
 * second authentication option will be supported later.
 */
public class AzureOpenAiChatModel implements ChatLanguageModel,
TokenCountEstimator {
    private final OpenAiClient client;
    private final Double temperature;
    private final Double topP;
    private final Integer maxTokens;
    private final Double presencePenalty;
    private final Double frequencyPenalty;
    private final Integer maxRetries;
    private final Tokenizer tokenizer;
    public AzureOpenAiChatModel(String baseUrl,
                                String apiVersion,
                                String apiKey,
```

```
                                Tokenizer tokenizer,
                                Double temperature,
                                Double topP,
                                Integer maxTokens,
                                Double presencePenalty,
                                Double frequencyPenalty,
                                Duration timeout,
                                Integer maxRetries,
                                Proxy proxy,
                                Boolean logRequests,
                                Boolean logResponses) {
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .azureApiKey(apiKey)
                .apiVersion(apiVersion)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.temperature = getOrDefault(temperature, 0.7);
        this.topP = topP;
        this.maxTokens = maxTokens;
        this.presencePenalty = presencePenalty;
        this.frequencyPenalty = frequencyPenalty;
        this.maxRetries = getOrDefault(maxRetries, 3);
        this.tokenizer = tokenizer;
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages) {
        return generate(messages, null, null);
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
List<ToolSpecification> toolSpecifications) {
        return generate(messages, toolSpecifications, null);
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
ToolSpecification toolSpecification) {
        return generate(messages, singletonList(toolSpecification),
toolSpecification);
    }
    private Response<AiMessage> generate(List<ChatMessage> messages,
                                         List<ToolSpecification>
toolSpecifications,
                                         ToolSpecification
toolThatMustBeExecuted
    ) {
        ChatCompletionRequest.Builder requestBuilder =
ChatCompletionRequest.builder()
                .messages(toOpenAiMessages(messages))
                .temperature(temperature)
                .topP(topP)
                .maxTokens(maxTokens)
                .presencePenalty(presencePenalty)
                .frequencyPenalty(frequencyPenalty);
```

```
        if (toolSpecifications != null && !toolSpecifications.isEmpty()) {
            requestBuilder.functions(toFunctions(toolSpecifications));
        }
        if (toolThatMustBeExecuted != null) {
            requestBuilder.functionCall(toolThatMustBeExecuted.name());
        }
        ChatCompletionRequest request = requestBuilder.build();
        ChatCompletionResponse response = withRetry(() ->
client.chatCompletion(request).execute(), maxRetries);
        return Response.from(
                aiMessageFrom(response),
                tokenUsageFrom(response.usage()),
                finishReasonFrom(response.choices().get(0).finishReason())
        );
    }
    @Override
    public int estimateTokenCount(List<ChatMessage> messages) {
        return tokenizer.estimateTokenCountInMessages(messages);
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String baseUrl;
        private String apiVersion;
        private String apiKey;
        private Tokenizer tokenizer;
        private Double temperature;
        private Double topP;
        private Integer maxTokens;
        private Double presencePenalty;
        private Double frequencyPenalty;
        private Duration timeout;
        private Integer maxRetries;
        private Proxy proxy;
        private Boolean logRequests;
        private Boolean logResponses;
        /**
         * Sets the Azure OpenAI base URL. This is a mandatory parameter.
         *
         * @param baseUrl The Azure OpenAI base URL in the format: https://
{resource}.openai.azure.com/openai/deployments/{deployment}
         * @return builder
         */
        public Builder baseUrl(String baseUrl) {
            this.baseUrl = baseUrl;
            return this;
        }
        /**
         * Sets the Azure OpenAI API version. This is a mandatory parameter.
         *
         * @param apiVersion The Azure OpenAI api version in the format:
2023-05-15
         * @return builder
         */
        public Builder apiVersion(String apiVersion) {
            this.apiVersion = apiVersion;
            return this;
        }
        /**
         * Sets the Azure OpenAI API key. This is a mandatory parameter.
```

```
     *
     * @param apiKey The Azure OpenAI API key.
     * @return builder
     */
    public Builder apiKey(String apiKey) {
        this.apiKey = apiKey;
        return this;
    }
    public Builder tokenizer(Tokenizer tokenizer) {
        this.tokenizer = tokenizer;
        return this;
    }
    public Builder temperature(Double temperature) {
        this.temperature = temperature;
        return this;
    }
    public Builder topP(Double topP) {
        this.topP = topP;
        return this;
    }
    public Builder maxTokens(Integer maxTokens) {
        this.maxTokens = maxTokens;
        return this;
    }
    public Builder presencePenalty(Double presencePenalty) {
        this.presencePenalty = presencePenalty;
        return this;
    }
    public Builder frequencyPenalty(Double frequencyPenalty) {
        this.frequencyPenalty = frequencyPenalty;
        return this;
    }
    public Builder timeout(Duration timeout) {
        this.timeout = timeout;
        return this;
    }
    public Builder maxRetries(Integer maxRetries) {
        this.maxRetries = maxRetries;
        return this;
    }
    public Builder proxy(Proxy proxy) {
        this.proxy = proxy;
        return this;
    }
    public Builder logRequests(Boolean logRequests) {
        this.logRequests = logRequests;
        return this;
    }
    public Builder logResponses(Boolean logResponses) {
        this.logResponses = logResponses;
        return this;
    }
    public AzureOpenAiChatModel build() {
        return new AzureOpenAiChatModel(
                baseUrl,
                apiVersion,
                apiKey,
                tokenizer,
                temperature,
                topP,
                maxTokens,
```

```
                    presencePenalty,
                    frequencyPenalty,
                    timeout,
                    maxRetries,
                    proxy,
                    logRequests,
                    logResponses
                );
            }
        }
    }
```

langchain4j-azure-open-ai\src\main\java\dev\langchain4j\model\azure\AzureOpenAiEmbeddingModel.java

```
package dev.langchain4j.model.azure;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.embedding.EmbeddingRequest;
import dev.ai4j.openai4j.embedding.EmbeddingResponse;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.embedding.TokenCountEstimator;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import java.net.Proxy;
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static java.time.Duration.ofSeconds;
import static java.util.stream.Collectors.toList;
/**
 * Represents an OpenAI embedding model, hosted on Azure, such as text-embedding-ada-002.
 * <p>
 * Mandatory parameters for initialization are: baseUrl, apiVersion and apiKey.
 * <p>
 * There are two primary authentication methods to access Azure OpenAI:
 * <p>
 * 1. API Key Authentication: For this type of authentication, HTTP requests must include the
 * API Key in the "api-key" HTTP header.
 * <p>
 * 2. Azure Active Directory Authentication: For this type of authentication, HTTP requests must include the
 * authentication/access token in the "Authorization" HTTP header.
 * <p>
 * <a href="https://learn.microsoft.com/en-us/azure/ai-services/openai/reference">More information</a>
 * <p>
 */
public class AzureOpenAiEmbeddingModel implements EmbeddingModel, TokenCountEstimator {
    private static final int BATCH_SIZE = 16;
    private final OpenAiClient client;
    private final Integer maxRetries;
    private final Tokenizer tokenizer;
    public AzureOpenAiEmbeddingModel(String baseUrl,
                                     String apiVersion,
                                     String apiKey,
                                     Tokenizer tokenizer,
                                     Duration timeout,
                                     Integer maxRetries,
                                     Proxy proxy,
                                     Boolean logRequests,
                                     Boolean logResponses) {
        timeout = getOrDefault(timeout, ofSeconds(60));
```

```
        this.client = OpenAiClient.builder()
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .azureApiKey(apiKey)
                .apiVersion(apiVersion)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.maxRetries = getOrDefault(maxRetries, 3);
        this.tokenizer = tokenizer;
    }
    /**
     * Embeds the provided text segments, processing a maximum of 16 segments
at a time.
     * For more information, refer to the documentation <a href="https://
learn.microsoft.com/en-us/azure/ai-services/openai/faq#i-am-trying-to-use-
embeddings-and-received-the-error--invalidrequesterror--too-many-inputs--the-
max-number-of-inputs-is-1---how-do-i-fix-this-">here</a>.
     *
     * @param textSegments A list of text segments.
     * @return A list of corresponding embeddings.
     */
    @Override
    public Response<List<Embedding>> embedAll(List<TextSegment> textSegments)
{
        List<String> texts = textSegments.stream()
                .map(TextSegment::text)
                .collect(toList());
        return embedTexts(texts);
    }
    private Response<List<Embedding>> embedTexts(List<String> texts) {
        List<Embedding> embeddings = new ArrayList<>();
        int inputTokenCount = 0;
        for (int i = 0; i < texts.size(); i += BATCH_SIZE) {
            List<String> batch = texts.subList(i, Math.min(i + BATCH_SIZE,
texts.size()));
            EmbeddingRequest request = EmbeddingRequest.builder()
                    .input(batch)
                    .build();
            EmbeddingResponse response = withRetry(() ->
client.embedding(request).execute(), maxRetries);
            embeddings.addAll(response.data().stream()
                    .map(openAiEmbedding ->
Embedding.from(openAiEmbedding.embedding()))
                    .collect(toList()));
            inputTokenCount += response.usage().promptTokens();
        }
        return Response.from(
                embeddings,
                new TokenUsage(inputTokenCount)
        );
    }
    @Override
    public int estimateTokenCount(String text) {
        return tokenizer.estimateTokenCountInText(text);
    }
    public static Builder builder() {
```

```
        return new Builder();
    }
    public static class Builder {
        private String baseUrl;
        private String apiVersion;
        private String apiKey;
        private Tokenizer tokenizer;
        private Duration timeout;
        private Integer maxRetries;
        private Proxy proxy;
        private Boolean logRequests;
        private Boolean logResponses;
        /**
         * Sets the Azure OpenAI base URL. This is a mandatory parameter.
         *
         * @param baseUrl The Azure OpenAI base URL in the format: https://
{resource}.openai.azure.com/openai/deployments/{deployment}
         * @return builder
         */
        public Builder baseUrl(String baseUrl) {
            this.baseUrl = baseUrl;
            return this;
        }
        /**
         * Sets the Azure OpenAI API version. This is a mandatory parameter.
         *
         * @param apiVersion The Azure OpenAI api version in the format:
2023-05-15
         * @return builder
         */
        public Builder apiVersion(String apiVersion) {
            this.apiVersion = apiVersion;
            return this;
        }
        /**
         * Sets the Azure OpenAI API key. This is a mandatory parameter.
         *
         * @param apiKey The Azure OpenAI API key.
         * @return builder
         */
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        public Builder tokenizer(Tokenizer tokenizer) {
            this.tokenizer = tokenizer;
            return this;
        }
        public Builder timeout(Duration timeout) {
            this.timeout = timeout;
            return this;
        }
        public Builder maxRetries(Integer maxRetries) {
            this.maxRetries = maxRetries;
            return this;
        }
        public Builder proxy(Proxy proxy) {
            this.proxy = proxy;
            return this;
        }
        public Builder logRequests(Boolean logRequests) {
```

```
            this.logRequests = logRequests;
            return this;
        }
        public Builder logResponses(Boolean logResponses) {
            this.logResponses = logResponses;
            return this;
        }
        public AzureOpenAiEmbeddingModel build() {
            return new AzureOpenAiEmbeddingModel(
                    baseUrl,
                    apiVersion,
                    apiKey,
                    tokenizer,
                    timeout,
                    maxRetries,
                    proxy,
                    logRequests,
                    logResponses
            );
        }
    }
}
```

langchain4j-azure-open-ai\src\main\java\dev\langchain4j\model\azure\AzureOpenAiLanguageModel.java

```
package dev.langchain4j.model.azure;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.completion.CompletionChoice;
import dev.ai4j.openai4j.completion.CompletionRequest;
import dev.ai4j.openai4j.completion.CompletionResponse;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.language.TokenCountEstimator;
import dev.langchain4j.model.output.Response;
import java.net.Proxy;
import java.time.Duration;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static
dev.langchain4j.model.openai.InternalOpenAiHelper.finishReasonFrom;
import static
dev.langchain4j.model.openai.InternalOpenAiHelper.tokenUsageFrom;
import static java.time.Duration.ofSeconds;
/**
 * Represents an OpenAI language model, hosted on Azure, such as text-
davinci-003.
 * However, it's recommended to use {@link AzureOpenAiChatModel} instead,
 * as it offers more advanced features like function calling, multi-turn
conversations, etc.
 * <p>
 * Mandatory parameters for initialization are: baseUrl, apiVersion and
apiKey.
 * <p>
 * There are two primary authentication methods to access Azure OpenAI:
 * <p>
 * 1. API Key Authentication: For this type of authentication, HTTP requests
must include the
 * API Key in the "api-key" HTTP header.
 * <p>
 * 2. Azure Active Directory Authentication: For this type of authentication,
HTTP requests must include the
 * authentication/access token in the "Authorization" HTTP header.
 * <p>
 * <a href="https://learn.microsoft.com/en-us/azure/ai-services/openai/
reference">More information</a>
 * <p>
 * Please note, that currently, only API Key authentication is supported by
this class,
 * second authentication option will be supported later.
 */
public class AzureOpenAiLanguageModel implements LanguageModel,
TokenCountEstimator {
    private final OpenAiClient client;
    private final Double temperature;
    private final Integer maxRetries;
    private final Tokenizer tokenizer;
    public AzureOpenAiLanguageModel(String baseUrl,
                                    String apiVersion,
                                    String apiKey,
                                    Tokenizer tokenizer,
                                    Double temperature,
```

```
                                    Duration timeout,
                                    Integer maxRetries,
                                    Proxy proxy,
                                    Boolean logRequests,
                                    Boolean logResponses) {
        timeout = timeout == null ? ofSeconds(60) : timeout;
        this.client = OpenAiClient.builder()
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .azureApiKey(apiKey)
                .apiVersion(apiVersion)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.temperature = getOrDefault(temperature, 0.7);
        this.maxRetries = getOrDefault(maxRetries, 3);
        this.tokenizer = tokenizer;
    }
    @Override
    public Response<String> generate(String prompt) {
        CompletionRequest request = CompletionRequest.builder()
                .prompt(prompt)
                .temperature(temperature)
                .build();
        CompletionResponse response = withRetry(() ->
client.completion(request).execute(), maxRetries);
        CompletionChoice completionChoice = response.choices().get(0);
        return Response.from(
                completionChoice.text(),
                tokenUsageFrom(response.usage()),
                finishReasonFrom(completionChoice.finishReason())
        );
    }
    @Override
    public int estimateTokenCount(String prompt) {
        return tokenizer.estimateTokenCountInText(prompt);
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String baseUrl;
        private String apiVersion;
        private String apiKey;
        private Tokenizer tokenizer;
        private Double temperature;
        private Duration timeout;
        private Integer maxRetries;
        private Proxy proxy;
        private Boolean logRequests;
        private Boolean logResponses;
        /**
         * Sets the Azure OpenAI base URL. This is a mandatory parameter.
         *
         * @param baseUrl The Azure OpenAI base URL in the format: https://
{resource}.openai.azure.com/openai/deployments/{deployment}
         * @return builder
```

```
         */
        public Builder baseUrl(String baseUrl) {
            this.baseUrl = baseUrl;
            return this;
        }
        /**
         * Sets the Azure OpenAI API version. This is a mandatory parameter.
         *
         * @param apiVersion The Azure OpenAI api version in the format:
2023-05-15
         * @return builder
         */
        public Builder apiVersion(String apiVersion) {
            this.apiVersion = apiVersion;
            return this;
        }
        /**
         * Sets the Azure OpenAI API key. This is a mandatory parameter.
         *
         * @param apiKey The Azure OpenAI API key.
         * @return builder
         */
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        public Builder tokenizer(Tokenizer tokenizer) {
            this.tokenizer = tokenizer;
            return this;
        }
        public Builder temperature(Double temperature) {
            this.temperature = temperature;
            return this;
        }
        public Builder timeout(Duration timeout) {
            this.timeout = timeout;
            return this;
        }
        public Builder maxRetries(Integer maxRetries) {
            this.maxRetries = maxRetries;
            return this;
        }
        public Builder proxy(Proxy proxy) {
            this.proxy = proxy;
            return this;
        }
        public Builder logRequests(Boolean logRequests) {
            this.logRequests = logRequests;
            return this;
        }
        public Builder logResponses(Boolean logResponses) {
            this.logResponses = logResponses;
            return this;
        }
        public AzureOpenAiLanguageModel build() {
            return new AzureOpenAiLanguageModel(
                    baseUrl,
                    apiVersion,
                    apiKey,
                    tokenizer,
                    temperature,
```

```
                    timeout,
                    maxRetries,
                    proxy,
                    logRequests,
                    logResponses
                );
            }
        }
    }
```

langchain4j-azure-open-ai\src\main\java\dev\langchain4j\model\azure\AzureOpenAiStreamingChatModel.java

```
package dev.langchain4j.model.azure;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.chat.ChatCompletionChoice;
import dev.ai4j.openai4j.chat.ChatCompletionRequest;
import dev.ai4j.openai4j.chat.ChatCompletionResponse;
import dev.ai4j.openai4j.chat.Delta;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.chat.TokenCountEstimator;
import dev.langchain4j.model.openai.OpenAiStreamingResponseBuilder;
import dev.langchain4j.model.output.Response;
import java.net.Proxy;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.toFunctions;
import static
dev.langchain4j.model.openai.InternalOpenAiHelper.toOpenAiMessages;
import static java.time.Duration.ofSeconds;
import static java.util.Collections.singletonList;
/**
 * Represents an OpenAI language model, hosted on Azure, that has a chat
completion interface, such as gpt-3.5-turbo.
 * The model's response is streamed token by token and should be handled with
{@link StreamingResponseHandler}.
 * <p>
 * Mandatory parameters for initialization are: baseUrl, apiVersion and
apiKey.
 * <p>
 * There are two primary authentication methods to access Azure OpenAI:
 * <p>
 * 1. API Key Authentication: For this type of authentication, HTTP requests
must include the
 * API Key in the "api-key" HTTP header.
 * <p>
 * 2. Azure Active Directory Authentication: For this type of authentication,
HTTP requests must include the
 * authentication/access token in the "Authorization" HTTP header.
 * <p>
 * <a href="https://learn.microsoft.com/en-us/azure/ai-services/openai/
reference">More information</a>
 * <p>
 * Please note, that currently, only API Key authentication is supported by
this class,
 * second authentication option will be supported later.
 */
public class AzureOpenAiStreamingChatModel implements
StreamingChatLanguageModel, TokenCountEstimator {
    private final OpenAiClient client;
    private final Double temperature;
    private final Double topP;
    private final Integer maxTokens;
```

```
    private final Double presencePenalty;
    private final Double frequencyPenalty;
    private final Tokenizer tokenizer;
    public AzureOpenAiStreamingChatModel(String baseUrl,
                                         String apiVersion,
                                         String apiKey,
                                         Tokenizer tokenizer,
                                         Double temperature,
                                         Double topP,
                                         Integer maxTokens,
                                         Double presencePenalty,
                                         Double frequencyPenalty,
                                         Duration timeout,
                                         Proxy proxy,
                                         Boolean logRequests,
                                         Boolean logResponses) {
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .azureApiKey(apiKey)
                .apiVersion(apiVersion)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logStreamingResponses(logResponses)
                .build();
        this.temperature = getOrDefault(temperature, 0.7);
        this.topP = topP;
        this.maxTokens = maxTokens;
        this.presencePenalty = presencePenalty;
        this.frequencyPenalty = frequencyPenalty;
        this.tokenizer = tokenizer;
    }
    @Override
    public void generate(List<ChatMessage> messages,
StreamingResponseHandler<AiMessage> handler) {
        generate(messages, null, null, handler);
    }
    @Override
    public void generate(List<ChatMessage> messages, List<ToolSpecification>
toolSpecifications, StreamingResponseHandler<AiMessage> handler) {
        generate(messages, toolSpecifications, null, handler);
    }
    @Override
    public void generate(List<ChatMessage> messages, ToolSpecification
toolSpecification, StreamingResponseHandler<AiMessage> handler) {
        generate(messages, singletonList(toolSpecification),
toolSpecification, handler);
    }
    private void generate(List<ChatMessage> messages,
                          List<ToolSpecification> toolSpecifications,
                          ToolSpecification toolThatMustBeExecuted,
                          StreamingResponseHandler<AiMessage> handler
    ) {
        ChatCompletionRequest.Builder requestBuilder =
ChatCompletionRequest.builder()
                .stream(true)
                .messages(toOpenAiMessages(messages))
```

```
                .temperature(temperature)
                .topP(topP)
                .maxTokens(maxTokens)
                .presencePenalty(presencePenalty)
                .frequencyPenalty(frequencyPenalty);
        Integer inputTokenCount = tokenizer == null ? null :
tokenizer.estimateTokenCountInMessages(messages);
        if (toolSpecifications != null && !toolSpecifications.isEmpty()) {
            requestBuilder.functions(toFunctions(toolSpecifications));
            if (tokenizer != null) {
                inputTokenCount +=
tokenizer.estimateTokenCountInToolSpecifications(toolSpecifications);
            }
        }
        if (toolThatMustBeExecuted != null) {
            requestBuilder.functionCall(toolThatMustBeExecuted.name());
            if (tokenizer != null) {
                inputTokenCount +=
tokenizer.estimateTokenCountInToolSpecification(toolThatMustBeExecuted);
            }
        }
        ChatCompletionRequest request = requestBuilder.build();
        OpenAiStreamingResponseBuilder responseBuilder = new
OpenAiStreamingResponseBuilder(inputTokenCount);
        client.chatCompletion(request)
                .onPartialResponse(partialResponse -> {
                    responseBuilder.append(partialResponse);
                    handle(partialResponse, handler);
                })
                .onComplete(() -> {
                    Response<AiMessage> response = responseBuilder.build();
                    handler.onComplete(response);
                })
                .onError(handler::onError)
                .execute();
    }
    private static void handle(ChatCompletionResponse partialResponse,
                               StreamingResponseHandler<AiMessage> handler) {
        List<ChatCompletionChoice> choices = partialResponse.choices();
        if (choices == null || choices.isEmpty()) {
            return;
        }
        Delta delta = choices.get(0).delta();
        String content = delta.content();
        if (content != null) {
            handler.onNext(content);
        }
    }
    @Override
    public int estimateTokenCount(List<ChatMessage> messages) {
        return tokenizer.estimateTokenCountInMessages(messages);
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String baseUrl;
        private String apiVersion;
        private String apiKey;
        private Tokenizer tokenizer;
        private Double temperature;
```

```
        private Double topP;
        private Integer maxTokens;
        private Double presencePenalty;
        private Double frequencyPenalty;
        private Duration timeout;
        private Proxy proxy;
        private Boolean logRequests;
        private Boolean logResponses;
        /**
         * Sets the Azure OpenAI base URL. This is a mandatory parameter.
         *
         * @param baseUrl The Azure OpenAI base URL in the format: https://
{resource}.openai.azure.com/openai/deployments/{deployment}
         * @return builder
         */
        public Builder baseUrl(String baseUrl) {
            this.baseUrl = baseUrl;
            return this;
        }
        /**
         * Sets the Azure OpenAI API version. This is a mandatory parameter.
         *
         * @param apiVersion The Azure OpenAI api version in the format:
2023-05-15
         * @return builder
         */
        public Builder apiVersion(String apiVersion) {
            this.apiVersion = apiVersion;
            return this;
        }
        /**
         * Sets the Azure OpenAI API key. This is a mandatory parameter.
         *
         * @param apiKey The Azure OpenAI API key.
         * @return builder
         */
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        public Builder tokenizer(Tokenizer tokenizer) {
            this.tokenizer = tokenizer;
            return this;
        }
        public Builder temperature(Double temperature) {
            this.temperature = temperature;
            return this;
        }
        public Builder topP(Double topP) {
            this.topP = topP;
            return this;
        }
        public Builder maxTokens(Integer maxTokens) {
            this.maxTokens = maxTokens;
            return this;
        }
        public Builder presencePenalty(Double presencePenalty) {
            this.presencePenalty = presencePenalty;
            return this;
        }
        public Builder frequencyPenalty(Double frequencyPenalty) {
```

```
            this.frequencyPenalty = frequencyPenalty;
            return this;
        }
        public Builder timeout(Duration timeout) {
            this.timeout = timeout;
            return this;
        }
        public Builder proxy(Proxy proxy) {
            this.proxy = proxy;
            return this;
        }
        public Builder logRequests(Boolean logRequests) {
            this.logRequests = logRequests;
            return this;
        }
        public Builder logResponses(Boolean logResponses) {
            this.logResponses = logResponses;
            return this;
        }
        public AzureOpenAiStreamingChatModel build() {
            return new AzureOpenAiStreamingChatModel(
                    baseUrl,
                    apiVersion,
                    apiKey,
                    tokenizer,
                    temperature,
                    topP,
                    maxTokens,
                    presencePenalty,
                    frequencyPenalty,
                    timeout,
                    proxy,
                    logRequests,
                    logResponses
            );
        }
    }
}
```

langchain4j-azure-open-ai\src\main\java\dev\langchain4j\model\azure\AzureOpenAiStreamingLanguageModel.java

```
package dev.langchain4j.model.azure;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.completion.CompletionRequest;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.language.StreamingLanguageModel;
import dev.langchain4j.model.language.TokenCountEstimator;
import dev.langchain4j.model.openai.OpenAiStreamingChatModel;
import dev.langchain4j.model.openai.OpenAiStreamingResponseBuilder;
import dev.langchain4j.model.output.Response;
import java.net.Proxy;
import java.time.Duration;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static java.time.Duration.ofSeconds;
/**
 * Represents an OpenAI language model, hosted on Azure, such as text-davinci-003.
 * The LLM's response is streamed token by token and should be handled with {@link StreamingResponseHandler}.
 * However, it's recommended to use {@link OpenAiStreamingChatModel} instead,
 * as it offers more advanced features like function calling, multi-turn conversations, etc.
 * <p>
 * Mandatory parameters for initialization are: baseUrl, apiVersion and apiKey.
 * <p>
 * There are two primary authentication methods to access Azure OpenAI:
 * <p>
 * 1. API Key Authentication: For this type of authentication, HTTP requests must include the
 * API Key in the "api-key" HTTP header.
 * <p>
 * 2. Azure Active Directory Authentication: For this type of authentication, HTTP requests must include the
 * authentication/access token in the "Authorization" HTTP header.
 * <p>
 * <a href="https://learn.microsoft.com/en-us/azure/ai-services/openai/reference">More information</a>
 * <p>
 * Please note, that currently, only API Key authentication is supported by this class,
 * second authentication option will be supported later.
 */
public class AzureOpenAiStreamingLanguageModel implements StreamingLanguageModel, TokenCountEstimator {
    private final OpenAiClient client;
    private final Double temperature;
    private final Tokenizer tokenizer;
    public AzureOpenAiStreamingLanguageModel(String baseUrl,
                                             String apiVersion,
                                             String apiKey,
                                             Tokenizer tokenizer,
                                             Double temperature,
                                             Duration timeout,
                                             Proxy proxy,
```

```
                                            Boolean logRequests,
                                            Boolean logResponses) {
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .azureApiKey(apiKey)
                .apiVersion(apiVersion)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logStreamingResponses(logResponses)
                .build();
        this.temperature = getOrDefault(temperature, 0.7);
        this.tokenizer = tokenizer;
    }
    @Override
    public void generate(String prompt, StreamingResponseHandler<String>
handler) {
        CompletionRequest request = CompletionRequest.builder()
                .prompt(prompt)
                .temperature(temperature)
                .build();
        Integer inputTokenCount = tokenizer == null ? null :
tokenizer.estimateTokenCountInText(prompt);
        OpenAiStreamingResponseBuilder responseBuilder = new
OpenAiStreamingResponseBuilder(inputTokenCount);
        client.completion(request)
                .onPartialResponse(partialResponse -> {
                    responseBuilder.append(partialResponse);
                    String token = partialResponse.text();
                    if (token != null) {
                        handler.onNext(token);
                    }
                })
                .onComplete(() -> {
                    Response<AiMessage> response = responseBuilder.build();
                    handler.onComplete(Response.from(
                            response.content().text(),
                            response.tokenUsage(),
                            response.finishReason()
                    ));
                })
                .onError(handler::onError)
                .execute();
    }
    @Override
    public int estimateTokenCount(String prompt) {
        return tokenizer.estimateTokenCountInText(prompt);
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String baseUrl;
        private String apiVersion;
        private String apiKey;
        private Tokenizer tokenizer;
        private Double temperature;
```

```
        private Duration timeout;
        private Proxy proxy;
        private Boolean logRequests;
        private Boolean logResponses;
        /**
         * Sets the Azure OpenAI base URL. This is a mandatory parameter.
         *
         * @param baseUrl The Azure OpenAI base URL in the format: https://
{resource}.openai.azure.com/openai/deployments/{deployment}
         * @return builder
         */
        public Builder baseUrl(String baseUrl) {
            this.baseUrl = baseUrl;
            return this;
        }
        /**
         * Sets the Azure OpenAI API version. This is a mandatory parameter.
         *
         * @param apiVersion The Azure OpenAI api version in the format:
2023-05-15
         * @return builder
         */
        public Builder apiVersion(String apiVersion) {
            this.apiVersion = apiVersion;
            return this;
        }
        /**
         * Sets the Azure OpenAI API key. This is a mandatory parameter.
         *
         * @param apiKey The Azure OpenAI API key.
         * @return builder
         */
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        public Builder tokenizer(Tokenizer tokenizer) {
            this.tokenizer = tokenizer;
            return this;
        }
        public Builder temperature(Double temperature) {
            this.temperature = temperature;
            return this;
        }
        public Builder timeout(Duration timeout) {
            this.timeout = timeout;
            return this;
        }
        public Builder proxy(Proxy proxy) {
            this.proxy = proxy;
            return this;
        }
        public Builder logRequests(Boolean logRequests) {
            this.logRequests = logRequests;
            return this;
        }
        public Builder logResponses(Boolean logResponses) {
            this.logResponses = logResponses;
            return this;
        }
        public AzureOpenAiStreamingLanguageModel build() {
```

```
            return new AzureOpenAiStreamingLanguageModel(
                    baseUrl,
                    apiVersion,
                    apiKey,
                    tokenizer,
                    temperature,
                    timeout,
                    proxy,
                    logRequests,
                    logResponses
            );
        }
    }
}
```

langchain4j-bom\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-bom</artifactId>
    <packaging>pom</packaging>
    <name>LangChain4j BOM</name>
    <description>Bill of Materials pom for getting full, complete set of
compatible versions
        of LangChain4j components
    </description>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-core</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-spring-boot-starter</artifactId>
                <version>${project.version}</version>
            </dependency>
            <!-- model providers -->
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-open-ai</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-azure-open-ai</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-hugging-face</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-local-ai</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
```

```
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-vertex-ai</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-dashscope</artifactId>
                <version>${project.version}</version>
            </dependency>
            <!-- embedding stores -->
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-cassandra</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-chroma</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-elasticsearch</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-milvus</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-opensearch</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-pinecone</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-redis</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-vespa</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-weaviate</artifactId>
                <version>${project.version}</version>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

langchain4j-cassandra\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <artifactId>langchain4j-cassandra</artifactId>
    <name>LangChain4j integration with Cassandra and AstraDb</name>
    <description>Some dependencies have a "Public Domain" license</
description>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <properties>
        <astra-sdk.version>0.6.11</astra-sdk.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j</artifactId>
            <version>${parent.version}</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
        </dependency>
        <dependency>
            <groupId>com.datastax.astra</groupId>
            <artifactId>astra-sdk-vector</artifactId>
            <version>${astra-sdk.version}</version>
            <exclusions>
                <exclusion>
                    <groupId>ch.qos.logback</groupId>
                    <artifactId>logback-classic</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <!-- removing cve -->
        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20230618</version>
        </dependency>
        <dependency>
            <groupId>commons-beanutils</groupId>
            <artifactId>commons-beanutils</artifactId>
            <version>1.9.4</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
```

```
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <version>${assertj.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-open-ai</artifactId>
            <version>${parent.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.honton.chas</groupId>
                <artifactId>license-maven-plugin</artifactId>
                <configuration>
                    <!-- org.json:json has a "Public Domain" license -->
                    <skipCompliance>true</skipCompliance>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

langchain4j-cassandra\src\main\java\dev\langchain4j\store\embedding\cassandra\AstraDbEmbeddingConfiguration.java

```
package dev.langchain4j.store.embedding.cassandra;
import lombok.Builder;
import lombok.Getter;
import lombok.NonNull;
/**
 * Plain old Java Object (POJO) to hold the configuration for the CassandraEmbeddingStore.
 * Wrapping all arguments needed to initialize a store in a single object makes it easier to pass them around.
 * It also makes it easier to add new arguments in the future, without having to change the constructor of the store.
 * This is especially useful when the store is used in a pipeline, where the arguments are passed around multiple times.
 *
 * @see CassandraEmbeddingStore
 */
@Getter
@Builder
public class AstraDbEmbeddingConfiguration {
    /**
     * Represents the Api Key to interact with Astra DB
     *
     * @see <a href="https://docs.datastax.com/en/astra/docs/manage-application-tokens.html">Astra DB Api Key</a>
     */
    @NonNull
    private String token;
    /**
     * Represents the unique identifier for your database.
     */
    @NonNull
    private String databaseId;
    /**
     * Represents the region where your database is hosted. A database can be deployed
     * in multiple regions at the same time, and you can choose the region that is closest to your users.
     * If a database has a single region, it will be picked for you.
     */
    private String databaseRegion;
    /**
     * Represents the workspace name where you create your tables. One database can hold multiple keyspaces.
     * Best practice is to provide a keyspace for each application.
     */
    @NonNull
    protected String keyspace;
    /**
     * Represents the name of the table.
     */
    @NonNull
    protected String table;
    /**
     * Represents the dimension of the vector used to save the embeddings.
     */
    @NonNull
    protected Integer dimension;
```

```
    /**
     * Initialize the builder.
     *
     * @return cassandra embedding configuration builder
     */
    public static
AstraDbEmbeddingConfiguration.AstraDbEmbeddingConfigurationBuilder builder() {
        return new
AstraDbEmbeddingConfiguration.AstraDbEmbeddingConfigurationBuilder();
    }
    /**
     * Signature for the builder.
     */
    public static class AstraDbEmbeddingConfigurationBuilder {
    }
}
```

langchain4j-cassandra\src\main\java\dev\langchain4j\store\embedding\cassandra\AstraDbEmbeddingStore.java

```
package dev.langchain4j.store.embedding.cassandra;
import com.datastax.astra.sdk.AstraClient;
import com.datastax.oss.driver.api.core.CqlSession;
import com.dtsx.astra.sdk.cassio.MetadataVectorCassandraTable;
import dev.langchain4j.store.embedding.EmbeddingStore;
/**
 * Implementation of {@link EmbeddingStore} using Cassandra AstraDB.
 *
 * @see EmbeddingStore
 * @see MetadataVectorCassandraTable
 */
public class AstraDbEmbeddingStore extends CassandraEmbeddingStoreSupport {
    /**
     * Build the store from the configuration.
     *
     * @param config configuration
     */
    public AstraDbEmbeddingStore(AstraDbEmbeddingConfiguration config) {
        CqlSession cqlSession = AstraClient.builder()
                .withToken(config.getToken())
                .withCqlKeyspace(config.getKeyspace())
                .withDatabaseId(config.getDatabaseId())
                .withDatabaseRegion(config.getDatabaseRegion())
                .enableCql()
                .enableDownloadSecureConnectBundle()
                .build().cqlSession();
        embeddingTable = new MetadataVectorCassandraTable(cqlSession,
config.getKeyspace(), config.getTable(), config.getDimension());
    }
    public static Builder builder() {
        return new Builder();
    }
    /**
     * Syntax Sugar Builder.
     */
    public static class Builder {
        /**
         * Configuration built with the builder
         */
        private final
AstraDbEmbeddingConfiguration.AstraDbEmbeddingConfigurationBuilder conf;
        /**
         * Initialization
         */
        public Builder() {
            conf = AstraDbEmbeddingConfiguration.builder();
        }
        /**
         * Populating token.
         *
         * @param token token
         * @return current reference
         */
        public Builder token(String token) {
            conf.token(token);
            return this;
        }
```

```
        /**
         * Populating token.
         *
         * @param databaseId     database Identifier
         * @param databaseRegion database region
         * @return current reference
         */
        public Builder database(String databaseId, String databaseRegion) {
            conf.databaseId(databaseId);
            conf.databaseRegion(databaseRegion);
            return this;
        }
        /**
         * Populating model dimension.
         *
         * @param dimension model dimension
         * @return current reference
         */
        public Builder vectorDimension(int dimension) {
            conf.dimension(dimension);
            return this;
        }
        /**
         * Populating table name.
         *
         * @param keyspace keyspace name
         * @param table    table name
         * @return current reference
         */
        public Builder table(String keyspace, String table) {
            conf.keyspace(keyspace);
            conf.table(table);
            return this;
        }
        /**
         * Building the Store.
         *
         * @return store for Astra.
         */
        public AstraDbEmbeddingStore build() {
            return new AstraDbEmbeddingStore(conf.build());
        }
    }
}
```

langchain4j-cassandra\src\main\java\dev\langchain4j\store\embedding\cassandra\CassandraEmbeddingConfiguration.java

```
package dev.langchain4j.store.embedding.cassandra;
import lombok.Builder;
import lombok.Getter;
import lombok.NonNull;
import java.util.List;
/**
 * Plain old Java Object (POJO) to hold the configuration for the CassandraEmbeddingStore.
 * Wrapping all arguments needed to initialize a store in a single object makes it easier to pass them around.
 * It also makes it easier to add new arguments in the future, without having to change the constructor of the store.
 * This is especially useful when the store is used in a pipeline, where the arguments are passed around multiple times.
 *
 * @see CassandraEmbeddingStore
 */
@Getter
@Builder
public class CassandraEmbeddingConfiguration {
    /**
     * Default Cassandra Port.
     */
    public static Integer DEFAULT_PORT = 9042;
    // --- Connectivity Parameters ---
    /**
     * Represents the cassandra Contact points.
     */
    @NonNull
    private List<String> contactPoints;
    /**
     * Represent the local data center.
     */
    @NonNull
    private String localDataCenter;
    /**
     * Connection Port
     */
    @NonNull
    private Integer port;
    /**
     * (Optional) Represents the username to connect to the database.
     */
    private String userName;
    /**
     * (Optional) Represents the password to connect to the database.
     */
    private String password;
    /**
     * Represents the workspace name where you create your tables. One database can hold multiple keyspaces.
     * Best practice is to provide a keyspace for each application.
     */
    @NonNull
    protected String keyspace;
    /**
     * Represents the name of the table.
```

```
     */
    @NonNull
    protected String table;
    /**
     * Represents the dimension of the model use to create the embeddings.
The vector holding the embeddings
     * is a fixed size. The dimension of the vector is the dimension of the
model used to create the embeddings.
     */
    @NonNull
    protected Integer dimension;
    /**
     * Initialize the builder.
     *
     * @return cassandra embedding configuration buildesr
     */
    public static CassandraEmbeddingConfigurationBuilder builder() {
        return new CassandraEmbeddingConfigurationBuilder();
    }
    /**
     * Signature for the builder.
     */
    public static class CassandraEmbeddingConfigurationBuilder {
    }
}
```

langchain4j-cassandra\src\main\java\dev\langchain4j\store\embedding\cassandra\CassandraEmbeddingStore.java

```
package dev.langchain4j.store.embedding.cassandra;
import com.datastax.oss.driver.api.core.CqlSession;
import com.datastax.oss.driver.api.core.CqlSessionBuilder;
import com.datastax.oss.driver.api.querybuilder.SchemaBuilder;
import com.dtsx.astra.sdk.cassio.MetadataVectorCassandraTable;
import com.dtsx.astra.sdk.cassio.SimilarityMetric;
import dev.langchain4j.store.embedding.EmbeddingStore;
import java.net.InetSocketAddress;
import java.util.Arrays;
/**
 * Implementation of {@link EmbeddingStore} using Cassandra AstraDB.
 *
 * @see EmbeddingStore
 * @see MetadataVectorCassandraTable
 */
public class CassandraEmbeddingStore extends CassandraEmbeddingStoreSupport {
    /**
     * Build the store from the configuration.
     *
     * @param config configuration
     */
    public CassandraEmbeddingStore(CassandraEmbeddingConfiguration config) {
        CqlSessionBuilder sessionBuilder = createCqlSessionBuilder(config);
        createKeyspaceIfNotExist(sessionBuilder, config.getKeyspace());
        sessionBuilder.withKeyspace(config.getKeyspace());
        this.embeddingTable = new
MetadataVectorCassandraTable(sessionBuilder.build(),
                config.getKeyspace(), config.getTable(),
config.getDimension(), SimilarityMetric.COS);
    }
    /**
     * Build the cassandra session from the config. At the difference of
adminSession there
     * a keyspace attached to it.
     *
     * @param config current configuration
     * @return cassandra session
     */
    private CqlSessionBuilder
createCqlSessionBuilder(CassandraEmbeddingConfiguration config) {
        CqlSessionBuilder cqlSessionBuilder = CqlSession.builder();
        cqlSessionBuilder.withLocalDatacenter(config.getLocalDataCenter());
        if (config.getUserName() != null && config.getPassword() != null) {
            cqlSessionBuilder.withAuthCredentials(config.getUserName(),
config.getPassword());
        }
        config.getContactPoints().forEach(cp ->
                cqlSessionBuilder.addContactPoint(new InetSocketAddress(cp,
config.getPort())));
        return cqlSessionBuilder;
    }
    /**
     * Create the keyspace in cassandra Destination if not exist.
     */
    private void createKeyspaceIfNotExist(CqlSessionBuilder
cqlSessionBuilder, String keyspace) {
        try (CqlSession adminSession = cqlSessionBuilder.build()) {
```

```
            adminSession.execute(SchemaBuilder.createKeyspace(keyspace)
                    .ifNotExists()
                    .withSimpleStrategy(1)
                    .withDurableWrites(true)
                    .build());
        }
    }
    public static Builder builder() {
        return new Builder();
    }
    /**
     * Syntax Sugar Builder.
     */
    public static class Builder {
        /**
         * Configuration built with the builder
         */
        private final
CassandraEmbeddingConfiguration.CassandraEmbeddingConfigurationBuilder conf;
        /**
         * Initialization
         */
        public Builder() {
            conf = CassandraEmbeddingConfiguration.builder();
        }
        /**
         * Populating cassandra port.
         *
         * @param port port
         * @return current reference
         */
        public CassandraEmbeddingStore.Builder port(int port) {
            conf.port(port);
            return this;
        }
        /**
         * Populating cassandra contact points.
         *
         * @param hosts port
         * @return current reference
         */
        public CassandraEmbeddingStore.Builder contactPoints(String... hosts)
{
            conf.contactPoints(Arrays.asList(hosts));
            return this;
        }
        /**
         * Populating model dimension.
         *
         * @param dimension model dimension
         * @return current reference
         */
        public CassandraEmbeddingStore.Builder vectorDimension(int dimension)
{
            conf.dimension(dimension);
            return this;
        }
        /**
         * Populating datacenter.
         *
         * @param dc datacenter
```

```
         * @return current reference
         */
        public CassandraEmbeddingStore.Builder localDataCenter(String dc) {
            conf.localDataCenter(dc);
            return this;
        }
        /**
         * Populating table name.
         *
         * @param keyspace keyspace name
         * @param table    table name
         * @return current reference
         */
        public CassandraEmbeddingStore.Builder table(String keyspace, String
table) {
            conf.keyspace(keyspace);
            conf.table(table);
            return this;
        }
        /**
         * Building the Store.
         *
         * @return store for Astra.
         */
        public CassandraEmbeddingStore build() {
            return new CassandraEmbeddingStore(conf.build());
        }
    }
}
```

langchain4j-cassandra\src\main\java\dev\langchain4j\store\embedding\cassandra\CassandraEmbeddingStoreSupport.java

```
package dev.langchain4j.store.embedding.cassandra;
import com.dtsx.astra.sdk.cassio.MetadataVectorCassandraTable;
import com.dtsx.astra.sdk.cassio.SimilarityMetric;
import com.dtsx.astra.sdk.cassio.SimilaritySearchQuery;
import
com.dtsx.astra.sdk.cassio.SimilaritySearchQuery.SimilaritySearchQueryBuilder;
import com.dtsx.astra.sdk.cassio.SimilaritySearchResult;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import lombok.Getter;
import lombok.NonNull;
import java.util.ArrayList;
import java.util.List;
import static dev.langchain4j.internal.ValidationUtils.ensureBetween;
import static dev.langchain4j.internal.ValidationUtils.ensureGreaterThanZero;
import static java.util.stream.Collectors.toList;
/**
 * Support for CassandraEmbeddingStore with and Without Astra.
 */
@Getter
abstract class CassandraEmbeddingStoreSupport implements
EmbeddingStore<TextSegment> {
    /**
     * Represents an embedding table in Cassandra, it is a table with a
vector column.
     */
    protected MetadataVectorCassandraTable embeddingTable;
    /**
     * Add a new embedding to the store.
     * - the row id is generated
     * - text and metadata are not stored
     *
     * @param embedding representation of the list of floats
     * @return newly created row id
     */
    @Override
    public String add(@NonNull Embedding embedding) {
        return add(embedding, null);
    }
    /**
     * Add a new embedding to the store.
     * - the row id is generated
     * - text and metadata coming from the text Segment
     *
     * @param embedding   representation of the list of floats
     * @param textSegment text content and metadata
     * @return newly created row id
     */
    @Override
    public String add(@NonNull Embedding embedding, TextSegment textSegment) {
        MetadataVectorCassandraTable.Record record = new
MetadataVectorCassandraTable.Record(embedding.vectorAsList());
```

```
        if (textSegment != null) {
            record.setBody(textSegment.text());
            record.setMetadata(textSegment.metadata().asMap());
        }
        embeddingTable.put(record);
        return record.getRowId();
    }
    /**
     * Add a new embedding to the store.
     *
     * @param rowId     the row id
     * @param embedding representation of the list of floats
     */
    @Override
    public void add(@NonNull String rowId, @NonNull Embedding embedding) {
        embeddingTable.put(new MetadataVectorCassandraTable.Record(rowId,
embedding.vectorAsList()));
    }
    /**
     * Add multiple embeddings as a single action.
     *
     * @param embeddingList embeddings list
     * @return list of new row if (same order as the input)
     */
    @Override
    public List<String> addAll(List<Embedding> embeddingList) {
        return embeddingList.stream()
                .map(Embedding::vectorAsList)
                .map(MetadataVectorCassandraTable.Record::new)
                .peek(embeddingTable::putAsync)
                .map(MetadataVectorCassandraTable.Record::getRowId)
                .collect(toList());
    }
    /**
     * Add multiple embeddings as a single action.
     *
     * @param embeddingList   embeddings
     * @param textSegmentList text segments
     * @return list of new row if (same order as the input)
     */
    @Override
    public List<String> addAll(List<Embedding> embeddingList,
List<TextSegment> textSegmentList) {
        if (embeddingList == null || textSegmentList == null ||
embeddingList.size() != textSegmentList.size()) {
            throw new IllegalArgumentException("embeddingList and
textSegmentList must not be null and have the same size");
        }
        // Looping on both list with an index
        List<String> ids = new ArrayList<>();
        for (int i = 0; i < embeddingList.size(); i++) {
            ids.add(add(embeddingList.get(i), textSegmentList.get(i)));
        }
        return ids;
    }
    /**
     * Search for relevant.
     *
     * @param embedding  current embeddings
     * @param maxResults max number of result
     * @param minScore   threshold
```

```
     * @return list of matching elements
     */
    @Override
    public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
embedding, int maxResults, double minScore) {
        return embeddingTable
                .similaritySearch(SimilaritySearchQuery.builder()
                        .embeddings(embedding.vectorAsList())
                        .recordCount(ensureGreaterThanZero(maxResults,
"maxResults"))
                        .threshold(CosineSimilarity.fromRelevanceScore(ensureB
etween(minScore, 0, 1, "minScore")))
                        .distance(SimilarityMetric.COS)
                        .build())
                .stream()
                .map(CassandraEmbeddingStoreSupport::mapSearchResult)
                .collect(toList());
    }
    /**
     * Map Search result coming from Astra.
     *
     * @param record current record
     * @return search result
     */
    private static EmbeddingMatch<TextSegment>
mapSearchResult(SimilaritySearchResult<MetadataVectorCassandraTable.Record>
record) {
        return new EmbeddingMatch<>(
                // Score
                RelevanceScore.fromCosineSimilarity(record.getSimilarity()),
                // EmbeddingId : unique identifier
                record.getEmbedded().getRowId(),
                // Embeddings vector
                Embedding.from(record.getEmbedded().getVector()),
                // Text segment and metadata
                TextSegment.from(record.getEmbedded().getBody(), new
Metadata(record.getEmbedded().getMetadata())));
    }
    /**
     * Similarity Search ANN based on the embedding.
     *
     * @param embedding  vector
     * @param maxResults max number of results
     * @param minScore   score minScore
     * @param metadata   map key-value to build a metadata filter
     * @return list of matching results
     */
    public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
embedding, int maxResults, double minScore, Metadata metadata) {
        SimilaritySearchQueryBuilder builder = SimilaritySearchQuery.builder()
                .embeddings(embedding.vectorAsList())
                .recordCount(ensureGreaterThanZero(maxResults, "maxResults"))
                .threshold(CosineSimilarity.fromRelevanceScore(ensureBetween(m
inScore, 0, 1, "minScore")));
        if (metadata != null) {
            builder.metaData(metadata.asMap());
        }
        return embeddingTable
                .similaritySearch(builder.build())
                .stream()
                .map(CassandraEmbeddingStoreSupport::mapSearchResult)
```

```
                .collect(toList());
    }
}
```

langchain4j-cassandra\src\main\java\dev\langchain4j\store\memory\chat\cassandra\AstraDbChatMemoryStore.java

```
package dev.langchain4j.store.memory.chat.cassandra;
import com.datastax.astra.sdk.AstraClient;
/**
 * AstraDb is a version of Cassandra running in Saas Mode.
 * <p>
 * The initialization of the CQLSession will be done through an AstraClient
 */
public class AstraDbChatMemoryStore extends CassandraChatMemoryStore {
    /**
     * Constructor with default table name.
     *
     * @param token        token
     * @param dbId         database identifier
     * @param dbRegion     database region
     * @param keyspaceName keyspace name
     */
    public AstraDbChatMemoryStore(String token, String dbId, String dbRegion,
String keyspaceName) {
        this(token, dbId, dbRegion, keyspaceName, DEFAULT_TABLE_NAME);
    }
    /**
     * Constructor with explicit table name.
     *
     * @param token        token
     * @param dbId         database identifier
     * @param dbRegion     database region
     * @param keyspaceName keyspace name
     * @param tableName    table name
     */
    public AstraDbChatMemoryStore(String token, String dbId, String dbRegion,
String keyspaceName, String tableName) {
        super(AstraClient.builder()
                .withToken(token)
                .withCqlKeyspace(keyspaceName)
                .withDatabaseId(dbId)
                .withDatabaseRegion(dbRegion)
                .enableCql()
                .enableDownloadSecureConnectBundle()
                .build().cqlSession(), keyspaceName, tableName);
    }
}
```

langchain4j-cassandra\src\main\java\dev\langchain4j\store\memory\chat\cassandra\CassandraChatMemoryStore.java

```
package dev.langchain4j.store.memory.chat.cassandra;
import com.datastax.oss.driver.api.core.CqlSession;
import com.datastax.oss.driver.api.core.uuid.Uuids;
import com.dtsx.astra.sdk.cassio.ClusteredCassandraTable;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.ChatMessageDeserializer;
import dev.langchain4j.data.message.ChatMessageSerializer;
import dev.langchain4j.store.memory.chat.ChatMemoryStore;
import lombok.NonNull;
import lombok.extern.slf4j.Slf4j;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import static com.dtsx.astra.sdk.cassio.ClusteredCassandraTable.Record;
import static java.util.stream.Collectors.toList;
/**
 * Implementation of {@link ChatMemoryStore} using Astra DB Vector Search.
 * Table contains all chats. (default name is message_store). Each chat with multiple messages
 * is a partition.Message id is a time uuid.
 *
 * @see <a href="https://docs.datastax.com/en/astra-serverless/docs/vector-search/overview.html">Astra Vector Store Documentation</a>
 */
@Slf4j
public class CassandraChatMemoryStore implements ChatMemoryStore {
    /**
     * Default message store.
     */
    public static final String DEFAULT_TABLE_NAME = "message_store";
    /**
     * Message Table.
     */
    private final ClusteredCassandraTable messageTable;
    /**
     * Constructor for message store
     *
     * @param session      cassandra session
     * @param keyspaceName keyspace name
     * @param tableName    table name
     */
    public CassandraChatMemoryStore(CqlSession session, String keyspaceName, String tableName) {
        messageTable = new ClusteredCassandraTable(session, keyspaceName, tableName);
    }
    /**
     * Constructor for message store
     *
     * @param session      cassandra session
     * @param keyspaceName keyspace name
     */
    public CassandraChatMemoryStore(CqlSession session, String keyspaceName) {
        messageTable = new ClusteredCassandraTable(session, keyspaceName, DEFAULT_TABLE_NAME);
    }
    /**
```

```
     * {@inheritDoc}
     */
    @Override
    public List<ChatMessage> getMessages(@NonNull Object memoryId) {
        /*
         * RATIONAL:
         * In the cassandra table the order is explicitly put to DESC with
         * latest to come first (for long conversation for instance). Here we
ask
         * for the full history. Instead of changing the multi purpose table
         * we reverse the list.
         */
        List<ChatMessage> latestFirstList = messageTable
                .findPartition(getMemoryId(memoryId))
                .stream()
                .map(this::toChatMessage)
                .collect(toList());
        Collections.reverse(latestFirstList);
        return latestFirstList;
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public void updateMessages(@NonNull Object memoryId, @NonNull
List<ChatMessage> messages) {
        deleteMessages(memoryId);
        messageTable.upsertPartition(messages.stream()
                .map(record -> fromChatMessage(getMemoryId(memoryId), record))
                .collect(toList()));
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public void deleteMessages(@NonNull Object memoryId) {
        messageTable.deletePartition(getMemoryId(memoryId));
    }
    /**
     * Unmarshalling Cassandra row as a Message with proper sub-type.
     *
     * @param record cassandra record
     * @return chat message
     */
    private ChatMessage toChatMessage(@NonNull Record record) {
        try {
            return ChatMessageDeserializer.messageFromJson(record.getBody());
        } catch (Exception e) {
            log.error("Unable to parse message body", e);
            throw new IllegalArgumentException("Unable to parse message
body");
        }
    }
    /**
     * Serialize the {@link ChatMessage} as a Cassandra Row.
     *
     * @param memoryId    chat session identifier
     * @param chatMessage chat message
     * @return cassandra row.
     */
    private Record fromChatMessage(@NonNull String memoryId, @NonNull
```

```
ChatMessage chatMessage) {
        try {
            Record record = new Record();
            record.setRowId(Uuids.timeBased());
            record.setPartitionId(memoryId);
            record.setBody(ChatMessageSerializer.messageToJson(chatMessage));
            return record;
        } catch (Exception e) {
            log.error("Unable to parse message body", e);
            throw new IllegalArgumentException("Unable to parse message
body", e);
        }
    }
    private String getMemoryId(Object memoryId) {
        if (!(memoryId instanceof String)) {
            throw new IllegalArgumentException("memoryId must be a String");
        }
        return (String) memoryId;
    }
}
```

langchain4j-cassandra\src\test\java\dev\langchain4j\store\embedding\cassandra\AstraDbEmbeddingConfigurationTest.java

```
package dev.langchain4j.store.embedding.cassandra;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
/**
 * The configuration objects include a few validation rules.
 */
public class AstraDbEmbeddingConfigurationTest {
    @Test
    public void should_build_configuration_test() {
        AstraDbEmbeddingConfiguration config =
AstraDbEmbeddingConfiguration.builder()
                .token("token")
                .databaseId("dbId")
                .databaseRegion("dbRegion")
                .keyspace("ks")
                .dimension(20)
                .table("table")
                .build();
        assertNotNull(config);
        assertNotNull(config.getToken());
        assertNotNull(config.getDatabaseId());
        assertNotNull(config.getDatabaseRegion());
    }
    @Test
    public void should_error_if_no_table_test() {
        // Table is required
        NullPointerException exception =
assertThrows(NullPointerException.class,
                () -> AstraDbEmbeddingConfiguration.builder()
                        .token("token")
                        .databaseId("dbId")
                        .databaseRegion("dbRegion")
                        .keyspace("ks")
                        .dimension(20)
                        .build());
        assertEquals("table is marked non-null but is null",
exception.getMessage());
    }
    @Test
    public void should_error_if_no_keyspace_test() {
        // Table is required
        NullPointerException exception =
assertThrows(NullPointerException.class,
                () -> AstraDbEmbeddingConfiguration.builder()
                        .token("token")
                        .databaseId("dbId")
                        .databaseRegion("dbRegion")
                        .table("ks")
                        .dimension(20)
                        .build());
        assertEquals("keyspace is marked non-null but is null",
exception.getMessage());
    }
    @Test
    public void should_error_if_no_dimension_test() {
        // Table is required
        NullPointerException exception =
```

```
assertThrows(NullPointerException.class,
                () -> AstraDbEmbeddingConfiguration.builder()
                        .token("token")
                        .databaseId("dbId")
                        .databaseRegion("dbRegion")
                        .table("ks")
                        .keyspace("ks")
                        .build());
        assertEquals("dimension is marked non-null but is null",
exception.getMessage());
    }
    @Test
    public void should_error_if_no_token_test() {
        // Table is required
        NullPointerException exception =
assertThrows(NullPointerException.class,
                () -> AstraDbEmbeddingConfiguration.builder()
                        .databaseId("dbId")
                        .databaseRegion("dbRegion")
                        .table("ks")
                        .keyspace("ks")
                        .dimension(20)
                        .build());
        assertEquals("token is marked non-null but is null",
exception.getMessage());
    }
    @Test
    public void should_error_if_no_database_test() {
        // Table is required
        NullPointerException exception =
assertThrows(NullPointerException.class,
                () -> AstraDbEmbeddingConfiguration.builder()
                        .token("token")
                        .table("ks")
                        .keyspace("ks")
                        .dimension(20)
                        .build());
        assertEquals("databaseId is marked non-null but is null",
exception.getMessage());
    }
}
```

langchain4j-cassandra\src\test\java\dev\langchain4j\store\embedding\cassandra\AstraDbEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.cassandra;
import com.datastax.astra.sdk.AstraClient;
import com.datastax.oss.driver.api.core.CqlSession;
import com.dtsx.astra.sdk.utils.TestUtils;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.EnabledIfEnvironmentVariable;
import java.util.List;
import static com.dtsx.astra.sdk.utils.TestUtils.getAstraToken;
import static com.dtsx.astra.sdk.utils.TestUtils.setupDatabase;
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
/**
 * Testing implementation of Embedding Store using AstraDB.
 */
class AstraDbEmbeddingStoreTest {
    private static final String TEST_KEYSPACE = "langchain4j";
    private static final String TEST_INDEX = "test_embedding_store";
    /**
     * We want to trigger the test only if the expected variable
     * is present.
     */
    @Test
    @EnabledIfEnvironmentVariable(named = "ASTRA_DB_APPLICATION_TOKEN",
matches = "Astra.*")
    void testAddEmbeddingAndFindRelevant() {
        String astraToken = getAstraToken();
        String databaseId = setupDatabase("langchain4j", TEST_KEYSPACE);
        // Flush Table for test to be idempotent
        truncateTable(databaseId, TEST_KEYSPACE, TEST_INDEX);
        // Create the Store with the builder
        AstraDbEmbeddingStore astraDbEmbeddingStore = new
AstraDbEmbeddingStore(AstraDbEmbeddingConfiguration
                .builder()
                .token(astraToken)
                .databaseId(databaseId)
                .databaseRegion(TestUtils.TEST_REGION)
                .keyspace(TEST_KEYSPACE)
                .table(TEST_INDEX)
                .dimension(11)
                .build());
        Embedding embedding = Embedding.from(new float[]{9.9F, 4.5F, 3.5F,
1.3F, 1.7F, 5.7F, 6.4F, 5.5F, 8.2F, 9.3F, 1.5F});
        TextSegment textSegment = TextSegment.from("Text",
Metadata.from("Key", "Value"));
        String id = astraDbEmbeddingStore.add(embedding, textSegment);
        assertTrue(id != null && !id.isEmpty());
        Embedding refereceEmbedding = Embedding.from(new float[]{8.7F, 4.5F,
3.4F, 1.2F, 5.5F, 5.6F, 6.4F, 5.5F, 8.1F, 9.1F, 1.1F});
        List<EmbeddingMatch<TextSegment>> embeddingMatches =
astraDbEmbeddingStore.findRelevant(refereceEmbedding, 10);
        assertEquals(1, embeddingMatches.size());
```

```
        EmbeddingMatch<TextSegment> embeddingMatch = embeddingMatches.get(0);
        assertThat(embeddingMatch.score()).isBetween(0d, 1d);
        assertThat(embeddingMatch.embeddingId()).isEqualTo(id);
        assertThat(embeddingMatch.embedding()).isEqualTo(embedding);
        assertThat(embeddingMatch.embedded()).isEqualTo(textSegment);
    }
    private void truncateTable(String databaseId, String keyspace, String
table) {
        try (AstraClient astraClient = AstraClient.builder()
                    .withToken(getAstraToken())
                    .withCqlKeyspace(keyspace)
                    .withDatabaseId(databaseId)
                    .withDatabaseRegion(TestUtils.TEST_REGION)
                    .enableCql()
                    .enableDownloadSecureConnectBundle()
                    .build()) {
            astraClient.cqlSession()
                       .execute("TRUNCATE TABLE " + table);
        }
    }
}
```

langchain4j-cassandra\src\test\java\dev\langchain4j\store\embedding\cassandra\CassandraEmbeddingConfigurationTest.java

```
package dev.langchain4j.store.embedding.cassandra;
import org.junit.jupiter.api.Test;
import static dev.langchain4j.store.embedding.cassandra.CassandraEmbeddingConfiguration.DEFAULT_PORT;
import static java.util.Collections.singletonList;
import static org.junit.jupiter.api.Assertions.*;
public class CassandraEmbeddingConfigurationTest {
    @Test
    public void should_build_configuration_test() {
        CassandraEmbeddingConfiguration config = CassandraEmbeddingConfiguration.builder()
                .contactPoints(singletonList("localhost"))
                .port(DEFAULT_PORT)
                .keyspace("ks")
                .dimension(20)
                .table("table")
                .localDataCenter("dc1")
                .build();
        assertNotNull(config);
    }
    @Test
    public void should_error_if_no_datacenter_test() {
        // Table is required
        NullPointerException exception = assertThrows(NullPointerException.class,
                () -> CassandraEmbeddingConfiguration.builder()
                        .contactPoints(singletonList("localhost"))
                        .port(DEFAULT_PORT)
                        .keyspace("ks")
                        .dimension(20)
                        .table("table")
                        .build());
        assertEquals("localDataCenter is marked non-null but is null", exception.getMessage());
    }
    @Test
    public void should_error_if_no_table_test() {
        // Table is required
        NullPointerException exception = assertThrows(NullPointerException.class,
                () -> CassandraEmbeddingConfiguration.builder()
                        .contactPoints(singletonList("localhost"))
                        .port(DEFAULT_PORT)
                        .keyspace("ks")
                        .dimension(20)
                        .localDataCenter("dc1")
                        .build());
        assertEquals("table is marked non-null but is null", exception.getMessage());
    }
    @Test
    public void should_error_if_no_keyspace_test() {
        // Table is required
        NullPointerException exception = assertThrows(NullPointerException.class,
                () -> CassandraEmbeddingConfiguration.builder()
                        .contactPoints(singletonList("localhost"))
```

```
                        .port(DEFAULT_PORT)
                        .table("ks")
                        .dimension(20)
                        .localDataCenter("dc1")
                        .build());
        assertEquals("keyspace is marked non-null but is null",
exception.getMessage());
    }
    @Test
    public void should_error_if_no_dimension_test() {
        // Table is required
        NullPointerException exception =
assertThrows(NullPointerException.class,
                () -> CassandraEmbeddingConfiguration.builder()
                        .contactPoints(singletonList("localhost"))
                        .port(DEFAULT_PORT)
                        .table("ks")
                        .keyspace("ks")
                        .localDataCenter("dc1")
                        .build());
        assertEquals("dimension is marked non-null but is null",
exception.getMessage());
    }
    @Test
    public void should_error_if_no_contact_points_test() {
        // Table is required
        NullPointerException exception =
assertThrows(NullPointerException.class,
                () -> CassandraEmbeddingConfiguration.builder()
                        .port(DEFAULT_PORT)
                        .table("ks")
                        .keyspace("ks")
                        .dimension(20)
                        .localDataCenter("dc1")
                        .build());
        assertEquals("contactPoints is marked non-null but is null",
exception.getMessage());
    }
}
```

langchain4j-cassandra\src\test\java\dev\langchain4j\store\embedding\cassandra\CassandraEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.cassandra;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
/**
 * Work with Cassandra Embedding Store.
 */
class CassandraEmbeddingStoreTest {
    @Test
    @Disabled("To run this test, you must have a local Cassandra instance, a docker-compose is provided")
    public void testAddEmbeddingAndFindRelevant() {
        CassandraEmbeddingStore cassandraEmbeddingStore = initStore();
        Embedding embedding = Embedding.from(new float[]{9.9F, 4.5F, 3.5F, 1.3F, 1.7F, 5.7F, 6.4F, 5.5F, 8.2F, 9.3F, 1.5F});
        TextSegment textSegment = TextSegment.from("Text", Metadata.from("Key", "Value"));
        String id = cassandraEmbeddingStore.add(embedding, textSegment);
        assertTrue(id != null && !id.isEmpty());
        Embedding refereceEmbedding = Embedding.from(new float[]{8.7F, 4.5F, 3.4F, 1.2F, 5.5F, 5.6F, 6.4F, 5.5F, 8.1F, 9.1F, 1.1F});
        List<EmbeddingMatch<TextSegment>> embeddingMatches = cassandraEmbeddingStore.findRelevant(refereceEmbedding, 1);
        assertEquals(1, embeddingMatches.size());
        EmbeddingMatch<TextSegment> embeddingMatch = embeddingMatches.get(0);
        assertThat(embeddingMatch.score()).isBetween(0d, 1d);
        assertThat(embeddingMatch.embeddingId()).isEqualTo(id);
        assertThat(embeddingMatch.embedding()).isEqualTo(embedding);
        assertThat(embeddingMatch.embedded()).isEqualTo(textSegment);
    }
    private CassandraEmbeddingStore initStore() {
        return CassandraEmbeddingStore.builder()
                .contactPoints("127.0.0.1")
                .port(9042)
                .localDataCenter("datacenter1")
                .table("langchain4j", "table_" + randomUUID().replace("-", ""))
                .vectorDimension(11)
                .build();
    }
}
```

langchain4j-cassandra\src\test\java\dev\langchain4j\store\embedding\cassandra\SampleDocumentLoaderAndRagWithAstraTest.java

```
package dev.langchain4j.store.embedding.cassandra;
import com.dtsx.astra.sdk.utils.TestUtils;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.document.DocumentType;
import dev.langchain4j.data.document.FileSystemDocumentLoader;
import dev.langchain4j.data.document.splitter.DocumentSplitters;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.PromptTemplate;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.openai.OpenAiEmbeddingModel;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.EmbeddingStoreIngestor;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.EnabledIfEnvironmentVariable;
import java.io.File;
import java.nio.file.Path;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static com.dtsx.astra.sdk.utils.TestUtils.getAstraToken;
import static com.dtsx.astra.sdk.utils.TestUtils.setupDatabase;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static
dev.langchain4j.model.openai.OpenAiModelName.TEXT_EMBEDDING_ADA_002;
import static java.time.Duration.ofSeconds;
import static java.util.stream.Collectors.joining;
import static org.junit.jupiter.api.Assertions.assertNotNull;
class SampleDocumentLoaderAndRagWithAstraTest {
    @Test
    @EnabledIfEnvironmentVariable(named = "ASTRA_DB_APPLICATION_TOKEN",
matches = "Astra.*")
    @EnabledIfEnvironmentVariable(named = "OPENAI_API_KEY", matches = "sk.*")
    void shouldRagWithOpenAiAndAstra() {
        // Initialization
        String astraToken = getAstraToken();
        String databaseId = setupDatabase("langchain4j", "langchain4j");
        String openAIKey = System.getenv("OPENAI_API_KEY");
        // Given
        assertNotNull(openAIKey);
        assertNotNull(databaseId);
        assertNotNull(astraToken);
        // --- Ingesting documents ---
        // Parsing input file
        Path path = new File(getClass().getResource("/story-about-happy-
carrot.txt").getFile()).toPath();
        Document document = FileSystemDocumentLoader.loadDocument(path,
DocumentType.TXT);
        DocumentSplitter splitter = DocumentSplitters
```

```
                .recursive(100, 10, new OpenAiTokenizer(GPT_3_5_TURBO));
        // Embedding model (OpenAI)
        EmbeddingModel embeddingModel = OpenAiEmbeddingModel.builder()
                .apiKey(openAIKey)
                .modelName(TEXT_EMBEDDING_ADA_002)
                .timeout(ofSeconds(15))
                .logRequests(true)
                .logResponses(true)
                .build();
        // Embed the document and it in the store
        EmbeddingStore<TextSegment> embeddingStore =
AstraDbEmbeddingStore.builder()
                .token(astraToken)
                .database(databaseId, TestUtils.TEST_REGION)
                .table("langchain4j", "table_story")
                .vectorDimension(1536)
                .build();
        // Ingest method 2
        EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
                .documentSplitter(splitter)
                .embeddingModel(embeddingModel)
                .embeddingStore(embeddingStore)
                .build();
        ingestor.ingest(document);
        // --------- RAG -------------
        // Specify the question you want to ask the model
        String question = "Who is Charlie?";
        // Embed the question
        Response<Embedding> questionEmbedding =
embeddingModel.embed(question);
        // Find relevant embeddings in embedding store by semantic similarity
        // You can play with parameters below to find a sweet spot for your
specific use case
        int maxResults = 3;
        double minScore = 0.8;
        List<EmbeddingMatch<TextSegment>> relevantEmbeddings =
                embeddingStore.findRelevant(questionEmbedding.content(),
maxResults, minScore);
        // --------- Chat Template -------------
        // Create a prompt for the model that includes question and relevant
embeddings
        PromptTemplate promptTemplate = PromptTemplate.from(
                "Answer the following question to the best of your ability:\n"
                        + "\n"
                        + "Question:\n"
                        + "{{question}}\n"
                        + "\n"
                        + "Base your answer on the following information:\n"
                        + "{{information}}");
        String information = relevantEmbeddings.stream()
                .map(match -> match.embedded().text())
                .collect(joining("\n\n"));
        Map<String, Object> variables = new HashMap<>();
        variables.put("question", question);
        variables.put("information", information);
        Prompt prompt = promptTemplate.apply(variables);
        // Send the prompt to the OpenAI chat model
        ChatLanguageModel chatModel = OpenAiChatModel.builder()
                .apiKey(openAIKey)
                .modelName(GPT_3_5_TURBO)
                .temperature(0.7)
```

```
                .timeout(ofSeconds(15))
                .maxRetries(3)
                .logResponses(true)
                .logRequests(true)
                .build();
        Response<AiMessage> aiMessage =
chatModel.generate(prompt.toUserMessage());
        // See an answer from the model
        String answer = aiMessage.content().text();
        System.out.println(answer);
    }
}
```

langchain4j-cassandra\src\test\java\dev\langchain4j\store\memory\chat\cassandra\ChatMemoryStoreAstraTest.java

```
package dev.langchain4j.store.memory.chat.cassandra;
import com.datastax.astra.sdk.AstraClient;
import com.dtsx.astra.sdk.utils.TestUtils;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.TokenWindowChatMemory;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import dev.langchain4j.store.memory.chat.ChatMemoryStore;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.EnabledIfEnvironmentVariable;
import java.util.UUID;
import static com.dtsx.astra.sdk.utils.TestUtils.*;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertNotNull;
/**
 * Test Cassandra Chat Memory Store with a Saas DB.
 */
public class ChatMemoryStoreAstraTest {
    private static final String TEST_DATABASE = "langchain4j";
    private static final String TEST_KEYSPACE = "langchain4j";
    @Test
    @EnabledIfEnvironmentVariable(named = "ASTRA_DB_APPLICATION_TOKEN",
matches = "Astra.*")
    @EnabledIfEnvironmentVariable(named = "OPENAI_API_KEY", matches = "sk.*")
    void chatMemoryAstraTest() {
        // Initialization
        String astraToken = getAstraToken();
        String databaseId = setupDatabase(TEST_DATABASE, TEST_KEYSPACE);
        // Given
        assertNotNull(databaseId);
        assertNotNull(astraToken);
        // Flush Table before test
        truncateTable(databaseId, TEST_KEYSPACE,
CassandraChatMemoryStore.DEFAULT_TABLE_NAME);
        // When
        ChatMemoryStore chatMemoryStore =
                new AstraDbChatMemoryStore(astraToken, databaseId,
TEST_REGION, "langchain4j");
        // When
        String chatSessionId = "chat-" + UUID.randomUUID();
        ChatMemory chatMemory = TokenWindowChatMemory.builder()
                .chatMemoryStore(chatMemoryStore)
                .id(chatSessionId)
                .maxTokens(300, new OpenAiTokenizer(GPT_3_5_TURBO))
                .build();
        // When
        UserMessage userMessage = userMessage("I will ask you a few question
about ff4j.");
        chatMemory.add(userMessage);
        AiMessage aiMessage = aiMessage("Sure, go ahead!");
        chatMemory.add(aiMessage);
        // Then
```

```
        assertThat(chatMemory.messages()).containsExactly(userMessage,
aiMessage);
    }
    private void truncateTable(String databaseId, String keyspace, String
table) {
        try (AstraClient astraClient = AstraClient.builder()
                .withToken(getAstraToken())
                .withCqlKeyspace(keyspace)
                .withDatabaseId(databaseId)
                .withDatabaseRegion(TestUtils.TEST_REGION)
                .enableCql()
                .enableDownloadSecureConnectBundle()
                .build()) {
            astraClient.cqlSession()
                    .execute("TRUNCATE TABLE " + table);
        }
    }
}
```

langchain4j-cassandra\src\test\resources\story-about-happy-carrot.txt

Once upon a time in the town of VeggieVille, there lived a cheerful carrot named Charlie.
Charlie was a radiant carrot, always beaming with joy and positivity.
His vibrant orange skin and lush green top were a sight to behold, but it was his infectious laughter and warm personality that really set him apart.
Charlie had a diverse group of friends, each a vegetable with their own unique characteristics.
There was Bella the blushing beetroot, always ready with a riddle or two; Timmy the timid tomato, a gentle soul with a heart of gold; and Percy the prankster potato, whose jokes always brought a smile to everyone's faces.
Despite their differences, they shared a close bond, their friendship as robust as their natural goodness.
Their lives were filled with delightful adventures, from playing hide-and-seek amidst the leafy lettuce to swimming in the dewy droplets that pooled on the cabbage leaves.
Their favorite place, though, was the sunlit corner of the vegetable patch, where they would bask in the warmth of the sun, share stories, and have hearty laughs.
One day, a bunch of pesky caterpillars invaded VeggieVille.
The vegetables were terrified, fearing they would be nibbled to nothingness.
But Charlie, with his usual sunny disposition, had an idea.
He proposed they host a grand feast for the caterpillars, with the juiciest leaves from the outskirts of the town.
Charlie's optimism was contagious, and his friends eagerly joined in to prepare the feast.
When the caterpillars arrived, they were pleasantly surprised.
They enjoyed the feast and were so impressed with the vegetables' hospitality that they promised not to trouble VeggieVille again.
In return, they agreed to help pollinate the flowers, contributing to a more lush and vibrant VeggieVille.
Charlie's idea had saved the day, but he humbly attributed the success to their teamwork and friendship.
They celebrated their victory with a grand party, filled with laughter, dance, and merry games.
That night, under the twinkling stars, they made a pact to always stand by each other, come what may.
From then on, the story of the happy carrot and his friends spread far and wide, a tale of friendship, unity, and positivity.
Charlie, Bella, Timmy, and Percy continued to live their joyful lives, their laughter echoing through VeggieVille.
And so, the tale of the happy carrot and his friends serves as a reminder that no matter the challenge, with optimism, teamwork, and a bit of creativity, anything is possible.

langchain4j-chroma\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-chroma</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Chroma</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.squareup.retrofit2</groupId>
            <artifactId>retrofit</artifactId>
        </dependency>
        <dependency>
            <groupId>com.squareup.retrofit2</groupId>
            <artifactId>converter-gson</artifactId>
        </dependency>
        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-chroma\src\main\java\dev\langchain4j\store\embedding\chroma\AddEmbeddingsRequest.java

```
package dev.langchain4j.store.embedding.chroma;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
class AddEmbeddingsRequest {
    private final List<String> ids;
    private final List<float[]> embeddings;
    private final List<String> documents;
    private final List<Map<String, String>> metadatas;
    public AddEmbeddingsRequest(Builder builder) {
        this.ids = builder.ids;
        this.embeddings = builder.embeddings;
        this.documents = builder.documents;
        this.metadatas = builder.metadatas;
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private List<String> ids = new ArrayList<>();
        private List<float[]> embeddings = new ArrayList<>();
        private List<String> documents = new ArrayList<>();
        private List<Map<String, String>> metadatas = new ArrayList<>();
        public Builder ids(List<String> ids) {
            if (ids != null) {
                this.ids = ids;
            }
            return this;
        }
        public Builder embeddings(List<float[]> embeddings) {
            if (embeddings != null) {
                this.embeddings = embeddings;
            }
            return this;
        }
        public Builder documents(List<String> documents) {
            if (documents != null) {
                this.documents = documents;
            }
            return this;
        }
        public Builder metadatas(List<Map<String, String>> metadatas) {
            if (metadatas != null) {
                this.metadatas = metadatas;
            }
            return this;
        }
        AddEmbeddingsRequest build() {
            return new AddEmbeddingsRequest(this);
        }
    }
}
```

langchain4j-chroma\src\main\java\dev\langchain4j\store\embedding\chroma\ChromaApi.java

```
package dev.langchain4j.store.embedding.chroma;
import retrofit2.Call;
import retrofit2.http.*;
interface ChromaApi {
    @GET("/api/v1/collections/{collection_name}")
    @Headers({"Content-Type: application/json"})
    Call<Collection> collection(@Path("collection_name") String
collectionName);
    @POST("/api/v1/collections")
    @Headers({"Content-Type: application/json"})
    Call<Collection> createCollection(@Body CreateCollectionRequest
createCollectionRequest);
    @POST("/api/v1/collections/{collection_id}/add")
    @Headers({"Content-Type: application/json"})
    Call<Boolean> addEmbeddings(@Path("collection_id") String collectionId,
@Body AddEmbeddingsRequest embedding);
    @POST("/api/v1/collections/{collection_id}/query")
    @Headers({"Content-Type: application/json"})
    Call<QueryResponse> queryCollection(@Path("collection_id") String
collectionId, @Body QueryRequest queryRequest);
}
```

langchain4j-chroma\src\main\java\dev\langchain4j\store\embedding\chroma\ChromaClient.java

```
package dev.langchain4j.store.embedding.chroma;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import okhttp3.OkHttpClient;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
import java.io.IOException;
import java.time.Duration;
import static com.google.gson.FieldNamingPolicy.LOWER_CASE_WITH_UNDERSCORES;
class ChromaClient {
    private final ChromaApi chromaApi;
    ChromaClient(String baseUrl, Duration timeout) {
        OkHttpClient okHttpClient = new OkHttpClient.Builder()
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .build();
        Gson gson = new GsonBuilder()
                .setFieldNamingPolicy(LOWER_CASE_WITH_UNDERSCORES)
                .create();
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(baseUrl)
                .client(okHttpClient)
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build();
        this.chromaApi = retrofit.create(ChromaApi.class);
    }
    Collection createCollection(CreateCollectionRequest
createCollectionRequest) {
        try {
            Response<Collection> response =
chromaApi.createCollection(createCollectionRequest).execute();
            if (response.isSuccessful()) {
                return response.body();
            } else {
                throw toException(response);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    Collection collection(String collectionName) {
        try {
            Response<Collection> response =
chromaApi.collection(collectionName).execute();
            if (response.isSuccessful()) {
                return response.body();
            } else {
                // if collection is not present, Chroma returns: Status - 500
                return null;
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

```
    boolean addEmbeddings(String collectionId, AddEmbeddingsRequest
addEmbeddingsRequest) {
        try {
            Response<Boolean> retrofitResponse =
chromaApi.addEmbeddings(collectionId, addEmbeddingsRequest)
                    .execute();
            if (retrofitResponse.isSuccessful()) {
                return Boolean.TRUE.equals(retrofitResponse.body());
            } else {
                throw toException(retrofitResponse);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    QueryResponse queryCollection(String collectionId, QueryRequest
queryRequest) {
        try {
            Response<QueryResponse> retrofitResponse =
chromaApi.queryCollection(collectionId, queryRequest)
                    .execute();
            if (retrofitResponse.isSuccessful()) {
                return retrofitResponse.body();
            } else {
                throw toException(retrofitResponse);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private static RuntimeException toException(Response<?> response) throws
IOException {
        int code = response.code();
        String body = response.errorBody().string();
        String errorMessage = String.format("status code: %s; body: %s",
code, body);
        return new RuntimeException(errorMessage);
    }
}
```

langchain4j-chroma\src\main\java\dev\langchain4j\store\embedding\chroma\ChromaEmbeddingStore.java

```
package dev.langchain4j.store.embedding.chroma;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.time.Duration.ofSeconds;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.toList;
/**
 * Represents a store for embeddings using the Chroma backend.
 * Always uses cosine distance as the distance metric.
 */
public class ChromaEmbeddingStore implements EmbeddingStore<TextSegment> {
    private final ChromaClient chromaClient;
    private final String collectionId;
    /**
     * Initializes a new instance of ChromaEmbeddingStore with the specified parameters.
     *
     * @param baseUrl        The base URL of the Chroma service.
     * @param collectionName The name of the collection in the Chroma service. If not specified, "default" will be used.
     * @param timeout        The timeout duration for the Chroma client. If not specified, 5 seconds will be used.
     */
    public ChromaEmbeddingStore(String baseUrl, String collectionName, Duration timeout) {
        collectionName = getOrDefault(collectionName, "default");
        this.chromaClient = new ChromaClient(baseUrl, getOrDefault(timeout, ofSeconds(5)));
        Collection collection = chromaClient.collection(collectionName);
        if (collection == null) {
            Collection createdCollection = chromaClient.createCollection(new CreateCollectionRequest(collectionName));
            collectionId = createdCollection.id();
        } else {
            collectionId = collection.id();
        }
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String baseUrl;
        private String collectionName;
        private Duration timeout;
        /**
         * @param baseUrl The base URL of the Chroma service.
         * @return builder
         */
```

```
        public Builder baseUrl(String baseUrl) {
            this.baseUrl = baseUrl;
            return this;
        }
        /**
         * @param collectionName The name of the collection in the Chroma
service. If not specified, "default" will be used.
         * @return builder
         */
        public Builder collectionName(String collectionName) {
            this.collectionName = collectionName;
            return this;
        }
        /**
         * @param timeout The timeout duration for the Chroma client. If not
specified, 5 seconds will be used.
         * @return builder
         */
        public Builder timeout(Duration timeout) {
            this.timeout = timeout;
            return this;
        }
        public ChromaEmbeddingStore build() {
            return new ChromaEmbeddingStore(this.baseUrl,
this.collectionName, this.timeout);
        }
    }
    @Override
    public String add(Embedding embedding) {
        String id = randomUUID();
        add(id, embedding);
        return id;
    }
    @Override
    public void add(String id, Embedding embedding) {
        addInternal(id, embedding, null);
    }
    @Override
    public String add(Embedding embedding, TextSegment textSegment) {
        String id = randomUUID();
        addInternal(id, embedding, textSegment);
        return id;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings) {
        List<String> ids = embeddings.stream()
                .map(embedding -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, null);
        return ids;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
textSegments) {
        List<String> ids = embeddings.stream()
                .map(embedding -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, textSegments);
        return ids;
    }
    private void addInternal(String id, Embedding embedding, TextSegment
```

```
textSegment) {
        addAllInternal(singletonList(id), singletonList(embedding),
textSegment == null ? null : singletonList(textSegment));
    }
    private void addAllInternal(List<String> ids, List<Embedding> embeddings,
List<TextSegment> textSegments) {
        AddEmbeddingsRequest addEmbeddingsRequest =
AddEmbeddingsRequest.builder()
                .embeddings(embeddings.stream()
                        .map(Embedding::vector)
                        .collect(toList()))
                .ids(ids)
                .metadatas(textSegments == null
                        ? null
                        : textSegments.stream()
                        .map(TextSegment::metadata)
                        .map(Metadata::asMap)
                        .collect(toList()))
                .documents(textSegments == null
                        ? null
                        : textSegments.stream()
                        .map(TextSegment::text)
                        .collect(toList()))
                .build();
        chromaClient.addEmbeddings(collectionId, addEmbeddingsRequest);
    }
    @Override
    public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
referenceEmbedding, int maxResults, double minScore) {
        QueryRequest queryRequest = new
QueryRequest(referenceEmbedding.vectorAsList(), maxResults);
        QueryResponse queryResponse =
chromaClient.queryCollection(collectionId, queryRequest);
        List<EmbeddingMatch<TextSegment>> matches =
toEmbeddingMatches(queryResponse);
        return matches.stream()
                .filter(match -> match.score() >= minScore)
                .collect(toList());
    }
    private static List<EmbeddingMatch<TextSegment>>
toEmbeddingMatches(QueryResponse queryResponse) {
        List<EmbeddingMatch<TextSegment>> embeddingMatches = new
ArrayList<>();
        for (int i = 0; i < queryResponse.ids().get(0).size(); i++) {
            double score =
distanceToScore(queryResponse.distances().get(0).get(i));
            String embeddingId = queryResponse.ids().get(0).get(i);
            Embedding embedding =
Embedding.from(queryResponse.embeddings().get(0).get(i));
            TextSegment textSegment = toTextSegment(queryResponse, i);
            embeddingMatches.add(new EmbeddingMatch<>(score, embeddingId,
embedding, textSegment));
        }
        return embeddingMatches;
    }
    /**
     * By default, cosine distance will be used. For details: <a
href="https://docs.trychroma.com/usage-guide"></a>
     * Converts a cosine distance in the range [0, 2] to a score in the range
[0, 1].
     *
```

```
     * @param distance The distance value.
     * @return The converted score.
     */
    private static double distanceToScore(double distance) {
        return 1 - (distance / 2);
    }
    private static TextSegment toTextSegment(QueryResponse queryResponse, int
i) {
        String text = queryResponse.documents().get(0).get(i);
        Map<String, String> metadata =
queryResponse.metadatas().get(0).get(i);
        return text == null ? null : TextSegment.from(text, metadata ==
null ? new Metadata() : new Metadata(metadata));
    }
}
```

langchain4j-chroma\src\main\java\dev\langchain4j\store\embedding\chroma\Collection.java

```
package dev.langchain4j.store.embedding.chroma;
import java.util.Map;
class Collection {
    private String id;
    private String name;
    private Map<String, String> metadata;
    public String id() {
        return id;
    }
    public String name() {
        return name;
    }
    public Map<String, String> metadata() {
        return metadata;
    }
}
```

langchain4j-chroma\src\main\java\dev\langchain4j\store\embedding\chroma\CreateCollectionRequest.java

```
package dev.langchain4j.store.embedding.chroma;
import java.util.HashMap;
import java.util.Map;
class CreateCollectionRequest {
    private final String name;
    private final Map<String, String> metadata;
    /**
     * Currently, cosine distance is always used as the distance method for chroma implementation
     */
    CreateCollectionRequest(String name) {
        this.name = name;
        HashMap<String, String> metadata = new HashMap<>();
        metadata.put("hnsw:space", "cosine");
        this.metadata = metadata;
    }
}
```

langchain4j-chroma\src\main\java\dev\langchain4j\store\embedding\chroma\QueryRequest.java

```
package dev.langchain4j.store.embedding.chroma;
import java.util.List;
import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
class QueryRequest {
    private final List<List<Float>> queryEmbeddings;
    private final int nResults;
    private final List<String> include = asList("metadatas", "documents",
"distances", "embeddings");
    public QueryRequest(List<Float> queryEmbedding, int nResults) {
        this.queryEmbeddings = singletonList(queryEmbedding);
        this.nResults = nResults;
    }
}
```

langchain4j-chroma\src\main\java\dev\langchain4j\store\embedding\chroma\QueryResponse.java

```
package dev.langchain4j.store.embedding.chroma;
import java.util.List;
import java.util.Map;
class QueryResponse {
    private List<List<String>> ids;
    private List<List<List<Float>>> embeddings;
    private List<List<String>> documents;
    private List<List<Map<String, String>>> metadatas;
    private List<List<Double>> distances;
    public List<List<String>> ids() {
        return ids;
    }
    public List<List<List<Float>>> embeddings() {
        return embeddings;
    }
    public List<List<String>> documents() {
        return documents;
    }
    public List<List<Map<String, String>>> metadatas() {
        return metadatas;
    }
    public List<List<Double>> distances() {
        return distances;
    }
}
```

langchain4j-chroma\src\test\java\dev\langchain4j\store\embedding\chroma\ChromaEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.chroma;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
@Disabled("needs Chroma running locally")
class ChromaEmbeddingStoreTest {
    /**
     * First ensure you have Chroma running locally. If not, then:
     * - Execute "docker pull ghcr.io/chroma-core/chroma:0.4.6"
     * - Execute "docker run -d -p 8000:8000 ghcr.io/chroma-core/chroma:0.4.6"
     * - Wait until Chroma is ready to serve (may take a few minutes)
     */
    private final EmbeddingStore<TextSegment> embeddingStore =
ChromaEmbeddingStore.builder()
            .baseUrl("http://localhost:8000")
            .collectionName(randomUUID())
            .build();
    private final EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    @Test
    void should_add_embedding() {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
```

```
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_embedding_with_segment_with_metadata() {
        TextSegment segment = TextSegment.from(randomUUID(),
Metadata.from("test-key", "test-value"));
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
```

```
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
        );
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
```

```
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
referenceEmbedding)),
                withPercentage(1)
        );
    }
}
```

langchain4j-core\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-core</artifactId>
    <packaging>jar</packaging>
    <name>langchain4j-core</name>
    <description>Core classes and interfaces of langchain4j</description>
    <dependencies>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
        </dependency>
        <dependency>
            <groupId>com.github.spullara.mustache.java</groupId>
            <artifactId>compiler</artifactId>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache License, Version 2.0</name>
            <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
        </license>
    </licenses>
</project>
```

langchain4j-core\src\main\java\dev\langchain4j\agent\tool\JsonSchemaProperty.java

```
package dev.langchain4j.agent.tool;
import java.util.Objects;
import static dev.langchain4j.internal.Utils.quoted;
public class JsonSchemaProperty {
    public static final JsonSchemaProperty STRING = type("string");
    public static final JsonSchemaProperty INTEGER = type("integer");
    public static final JsonSchemaProperty NUMBER = type("number");
    public static final JsonSchemaProperty OBJECT = type("object");
    public static final JsonSchemaProperty ARRAY = type("array");
    public static final JsonSchemaProperty BOOLEAN = type("boolean");
    public static final JsonSchemaProperty NULL = type("null");
    private final String key;
    private final Object value;
    public JsonSchemaProperty(String key, Object value) {
        this.key = key;
        this.value = value;
    }
    public String key() {
        return key;
    }
    public Object value() {
        return value;
    }
    @Override
    public boolean equals(Object another) {
        if (this == another) return true;
        return another instanceof JsonSchemaProperty
                && equalTo((JsonSchemaProperty) another);
    }
    private boolean equalTo(JsonSchemaProperty another) {
        return Objects.equals(key, another.key)
                && Objects.equals(value, another.value);
    }
    @Override
    public int hashCode() {
        int h = 5381;
        h += (h << 5) + Objects.hashCode(key);
        h += (h << 5) + Objects.hashCode(value);
        return h;
    }
    @Override
    public String toString() {
        return "JsonSchemaProperty {"
                + " key = " + quoted(key)
                + ", value = " + value
                + " }";
    }
    public static JsonSchemaProperty from(String key, Object value) {
        return new JsonSchemaProperty(key, value);
    }
    public static JsonSchemaProperty property(String key, Object value) {
        return from(key, value);
    }
    public static JsonSchemaProperty type(String value) {
        return from("type", value);
    }
    public static JsonSchemaProperty description(String value) {
```

```
        return from("description", value);
    }
    public static JsonSchemaProperty enums(String... enumValues) {
        return from("enum", enumValues);
    }
    public static JsonSchemaProperty enums(Object... enumValues) {
        for (Object enumValue : enumValues) {
            if (!enumValue.getClass().isEnum()) {
                throw new RuntimeException("Value " +
enumValue.getClass().getName() + " should be enum");
            }
        }
        return from("enum", enumValues);
    }
    public static JsonSchemaProperty enums(Class<?> enumClass) {
        if (!enumClass.isEnum()) {
            throw new RuntimeException("Class " + enumClass.getName() + "
should be enum");
        }
        return from("enum", enumClass.getEnumConstants());
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\agent\tool\P.java

```
package dev.langchain4j.agent.tool;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
/**
 * Parameter of a Tool
 */
@Retention(RUNTIME)
@Target({PARAMETER})
public @interface P {
    /**
     * Description of a parameter
     */
    String value();
}
```

langchain4j-core\src\main\java\dev\langchain4j\agent\tool\Tool.java

```
package dev.langchain4j.agent.tool;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
/**
 * Java methods annotated with @Tool are considered tools that language model
can use.
 */
@Retention(RUNTIME)
@Target({METHOD})
public @interface Tool {
    /**
     * Name of the tool. If not provided, method name will be used.
     */
    String name() default "";
    /**
     * Description of the tool.
     * It should be clear and descriptive to allow language model to
understand the tool's purpose and its intended use.
     */
    String[] value() default "";
}
```

langchain4j-core\src\main\java\dev\langchain4j\agent\tool\ToolExecutionRequest.java

```
package dev.langchain4j.agent.tool;
import static dev.langchain4j.internal.Utils.quoted;
import java.util.Objects;
public class ToolExecutionRequest {
    private final String name;
    private final String arguments;
    private ToolExecutionRequest(Builder builder) {
        this.name = builder.name;
        this.arguments = builder.arguments;
    }
    public String name() {
        return name;
    }
    public String arguments() {
        return arguments;
    }
    @Override
    public boolean equals(Object another) {
        if (this == another) return true;
        return another instanceof ToolExecutionRequest
                && equalTo((ToolExecutionRequest) another);
    }
    private boolean equalTo(ToolExecutionRequest another) {
        return Objects.equals(name, another.name)
                && Objects.equals(arguments, another.arguments);
    }
    @Override
    public int hashCode() {
        int h = 5381;
        h += (h << 5) + Objects.hashCode(name);
        h += (h << 5) + Objects.hashCode(arguments);
        return h;
    }
    @Override
    public String toString() {
        return "ToolExecutionRequest {"
                + " name = " + quoted(name)
                + ", arguments = " + quoted(arguments)
                + " }";
    }
    public static Builder builder() {
        return new Builder();
    }
    public static final class Builder {
        private String name;
        private String arguments;
        private Builder() {
        }
        public Builder name(String name) {
            this.name = name;
            return this;
        }
        public Builder arguments(String arguments) {
            this.arguments = arguments;
            return this;
        }
        public ToolExecutionRequest build() {
```

```
            return new ToolExecutionRequest(this);
        }
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\agent\tool\ToolExecutionRequestUtil.java

```
package dev.langchain4j.agent.tool;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import java.lang.reflect.Type;
import java.util.Map;
public class ToolExecutionRequestUtil {
    public static final Gson GSON = new Gson();
    public static final Type MAP_TYPE = new TypeToken<Map<String, Object>>() {
    }.getType();
    public static <T> T argument(ToolExecutionRequest toolExecutionRequest,
String name) {
        Map<String, Object> arguments =
argumentsAsMap(toolExecutionRequest.arguments()); // TODO cache
        return (T) arguments.get(name);
    }
    public static Map<String, Object> argumentsAsMap(String arguments) {
        return GSON.fromJson(arguments, MAP_TYPE);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\agent\tool\ToolParameters.java

```
package dev.langchain4j.agent.tool;
import java.util.*;
import static dev.langchain4j.internal.Utils.quoted;
public class ToolParameters {
    private final String type;
    private final Map<String, Map<String, Object>> properties;
    private final List<String> required;
    private ToolParameters(Builder builder) {
        this.type = builder.type;
        this.properties = builder.properties;
        this.required = builder.required;
    }
    public String type() {
        return type;
    }
    public Map<String, Map<String, Object>> properties() {
        return properties;
    }
    public List<String> required() {
        return required;
    }
    @Override
    public boolean equals(Object another) {
        if (this == another) return true;
        return another instanceof ToolParameters
                && equalTo((ToolParameters) another);
    }
    private boolean equalTo(ToolParameters another) {
        return Objects.equals(type, another.type)
                && Objects.equals(properties, another.properties)
                && Objects.equals(required, another.required);
    }
    @Override
    public int hashCode() {
        int h = 5381;
        h += (h << 5) + Objects.hashCode(type);
        h += (h << 5) + Objects.hashCode(properties);
        h += (h << 5) + Objects.hashCode(required);
        return h;
    }
    @Override
    public String toString() {
        return "ToolParameters {"
                + " type = " + quoted(type)
                + ", properties = " + properties
                + ", required = " + required
                + " }";
    }
    public static Builder builder() {
        return new Builder();
    }
    public static final class Builder {
        private String type = "object";
        private Map<String, Map<String, Object>> properties = new HashMap<>();
        private List<String> required = new ArrayList<>();
        private Builder() {
        }
        public Builder type(String type) {
```

```
            this.type = type;
            return this;
        }
        public Builder properties(Map<String, Map<String, Object>>
properties) {
            this.properties = properties;
            return this;
        }
        public Builder required(List<String> required) {
            this.required = required;
            return this;
        }
        public ToolParameters build() {
            return new ToolParameters(this);
        }
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\agent\tool\ToolSpecification.java

```
package dev.langchain4j.agent.tool;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
import static dev.langchain4j.internal.Utils.quoted;
import static java.util.Arrays.asList;
public class ToolSpecification {
    private final String name;
    private final String description;
    private final ToolParameters parameters;
    private ToolSpecification(Builder builder) {
        this.name = builder.name;
        this.description = builder.description;
        this.parameters = builder.parameters;
    }
    public String name() {
        return name;
    }
    public String description() {
        return description;
    }
    public ToolParameters parameters() {
        return parameters;
    }
    @Override
    public boolean equals(Object another) {
        if (this == another) return true;
        return another instanceof ToolSpecification
                && equalTo((ToolSpecification) another);
    }
    private boolean equalTo(ToolSpecification another) {
        return Objects.equals(name, another.name)
                && Objects.equals(description, another.description)
                && Objects.equals(parameters, another.parameters);
    }
    @Override
    public int hashCode() {
        int h = 5381;
        h += (h << 5) + Objects.hashCode(name);
        h += (h << 5) + Objects.hashCode(description);
        h += (h << 5) + Objects.hashCode(parameters);
        return h;
    }
    @Override
    public String toString() {
        return "ToolSpecification {"
                + " name = " + quoted(name)
                + ", description = " + quoted(description)
                + ", parameters = " + parameters
                + " }";
    }
    public static Builder builder() {
        return new Builder();
    }
    public static final class Builder {
        private String name;
        private String description;
```

```
        private ToolParameters parameters;
        private Builder() {
        }
        public Builder name(String name) {
            this.name = name;
            return this;
        }
        public Builder description(String description) {
            this.description = description;
            return this;
        }
        public Builder parameters(ToolParameters parameters) {
            this.parameters = parameters;
            return this;
        }
        public Builder addParameter(String name, JsonSchemaProperty...
jsonSchemaProperties) {
            return addParameter(name, asList(jsonSchemaProperties));
        }
        public Builder addParameter(String name, Iterable<JsonSchemaProperty>
jsonSchemaProperties) {
            addOptionalParameter(name, jsonSchemaProperties);
            this.parameters.required().add(name);
            return this;
        }
        public Builder addOptionalParameter(String name,
JsonSchemaProperty... jsonSchemaProperties) {
            return addOptionalParameter(name, asList(jsonSchemaProperties));
        }
        public Builder addOptionalParameter(String name,
Iterable<JsonSchemaProperty> jsonSchemaProperties) {
            if (this.parameters == null) {
                this.parameters = ToolParameters.builder().build();
            }
            Map<String, Object> jsonSchemaPropertiesMap = new HashMap<>();
            for (JsonSchemaProperty jsonSchemaProperty :
jsonSchemaProperties) {
                jsonSchemaPropertiesMap.put(jsonSchemaProperty.key(),
jsonSchemaProperty.value());
            }
            this.parameters.properties().put(name, jsonSchemaPropertiesMap);
            return this;
        }
        public ToolSpecification build() {
            return new ToolSpecification(this);
        }
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\agent\tool\ToolSpecifications.java

```
package dev.langchain4j.agent.tool;
import java.lang.reflect.Method;
import java.lang.reflect.Parameter;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.List;
import java.util.Objects;
import java.util.Set;
import static dev.langchain4j.agent.tool.JsonSchemaProperty.*;
import static dev.langchain4j.internal.Utils.isNullOrBlank;
import static java.util.Arrays.stream;
import static java.util.stream.Collectors.toList;
public class ToolSpecifications {
    public static List<ToolSpecification> toolSpecificationsFrom(Object
objectWithTools) {
        return stream(objectWithTools.getClass().getDeclaredMethods())
                .filter(method -> method.isAnnotationPresent(Tool.class))
                .map(ToolSpecifications::toolSpecificationFrom)
                .collect(toList());
    }
    public static ToolSpecification toolSpecificationFrom(Method method) {
        Tool annotation = method.getAnnotation(Tool.class);
        String name = isNullOrBlank(annotation.name()) ? method.getName() :
annotation.name();
        String description = String.join("\n", annotation.value());
        ToolSpecification.Builder builder = ToolSpecification.builder()
                .name(name)
                .description(description);
        for (Parameter parameter : method.getParameters()) {
            builder.addParameter(parameter.getName(),
toJsonSchemaProperties(parameter));
        }
        return builder.build();
    }
    private static Iterable<JsonSchemaProperty>
toJsonSchemaProperties(Parameter parameter) {
        Class<?> type = parameter.getType();
        P annotation = parameter.getAnnotation(P.class);
        JsonSchemaProperty description = annotation == null ? null :
description(annotation.value());
        if (type == String.class) {
            return removeNulls(STRING, description);
        }
        if (type == boolean.class || type == Boolean.class) {
            return removeNulls(BOOLEAN, description);
        }
        if (type == byte.class || type == Byte.class
                || type == short.class || type == Short.class
                || type == int.class || type == Integer.class
                || type == long.class || type == Long.class
                || type == BigInteger.class) {
            return removeNulls(INTEGER, description);
        }
        // TODO put constraints on min and max?
        if (type == float.class || type == Float.class
                || type == double.class || type == Double.class
                || type == BigDecimal.class) {
```

```
            return removeNulls(NUMBER, description);
        }
        if (type.isArray()
                || type == List.class
                || type == Set.class) { // TODO something else?
            return removeNulls(ARRAY, description); // TODO provide type of
array?
        }
        if (type.isEnum()) {
            return removeNulls(STRING, enums((Object[])
type.getEnumConstants()), description);
        }
        return removeNulls(OBJECT, description); // TODO provide internals
    }
    private static Iterable<JsonSchemaProperty>
removeNulls(JsonSchemaProperty... properties) {
        return stream(properties)
                .filter(Objects::nonNull)
                .collect(toList());
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\chain\Chain.java

```
package dev.langchain4j.chain;
public interface Chain<Input, Output> {
    Output execute(Input input);
}
```

langchain4j-core\src\main\java\dev\langchain4j\classification\TextClassifier.java

```
package dev.langchain4j.classification;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.segment.TextSegment;
import java.util.List;
/**
 * Classifies given text according to specified enum.
 *
 * @param <E> Enum that is the result of classification.
 */
public interface TextClassifier<E extends Enum<E>> {
    /**
     * Classify the given text.
     *
     * @param text Text to classify.
     * @return A list of classification categories.
     */
    List<E> classify(String text);
    /**
     * Classify the given {@link TextSegment}.
     *
     * @param textSegment {@link TextSegment} to classify.
     * @return A list of classification categories.
     */
    default List<E> classify(TextSegment textSegment) {
        return classify(textSegment.text());
    }
    /**
     * Classify the given {@link Document}.
     *
     * @param document {@link Document} to classify.
     * @return A list of classification categories.
     */
    default List<E> classify(Document document) {
        return classify(document.text());
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\code\CodeExecutionEngine.java

```
package dev.langchain4j.code;
public interface CodeExecutionEngine {
    String execute(String code);
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\document\Document.java

```
package dev.langchain4j.data.document;
import dev.langchain4j.data.segment.TextSegment;
import java.util.Objects;
import static dev.langchain4j.internal.Utils.quoted;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Represents an unstructured piece of text that usually corresponds to a
content of a single file.
 * This text could originate from various sources such as a text file, PDF,
DOCX, or a web page (HTML).
 * Each document may have associated metadata including its source, owner,
creation date, etc.
 */
public class Document {
    public static final String DOCUMENT_TYPE = "document_type";
    public static final String FILE_NAME = "file_name";
    public static final String ABSOLUTE_DIRECTORY_PATH =
"absolute_directory_path";
    public static final String URL = "url";
    private final String text;
    private final Metadata metadata;
    public Document(String text, Metadata metadata) {
        this.text = ensureNotBlank(text, "text");
        this.metadata = ensureNotNull(metadata, "metadata");
    }
    public String text() {
        return text;
    }
    public Metadata metadata() {
        return metadata;
    }
    public String metadata(String key) {
        return metadata.get(key);
    }
    public TextSegment toTextSegment() {
        return TextSegment.from(text, metadata.copy().add("index", 0));
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Document that = (Document) o;
        return Objects.equals(this.text, that.text)
                && Objects.equals(this.metadata, that.metadata);
    }
    @Override
    public int hashCode() {
        return Objects.hash(text, metadata);
    }
    @Override
    public String toString() {
        return "Document {" +
                " text = " + quoted(text) +
                " metadata = " + metadata.asMap() +
                " }";
    }
    public static Document from(String text) {
```

```
        return new Document(text, new Metadata());
    }
    public static Document from(String text, Metadata metadata) {
        return new Document(text, metadata);
    }
    public static Document document(String text) {
        return from(text);
    }
    public static Document document(String text, Metadata metadata) {
        return from(text, metadata);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\document\DocumentParser.java

```
package dev.langchain4j.data.document;
import java.io.InputStream;
/**
 * Defines the interface for parsing an InputStream into a Document.
 * Different document types require specialized parsing logic.
 */
public interface DocumentParser {
    /**
     * Parses an InputStream into a Document.
     * The specific implementation of this method will depend on the type of
the document being parsed.
     *
     * @param inputStream The InputStream that contains the content of the
document.
     * @return The parsed Document.
     */
    Document parse(InputStream inputStream);
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\document\DocumentSource.java

```
package dev.langchain4j.data.document;
import java.io.IOException;
import java.io.InputStream;
/**
 * Defines the interface for a Document source.
 * Documents can be loaded from various sources such as the file system, HTTP, FTP, etc.
 */
public interface DocumentSource {
    /**
     * Provides an InputStream to read the content of the document.
     * This method can be implemented to read from various sources like a local file or a network connection.
     *
     * @return An InputStream from which the document content can be read.
     * @throws IOException If an I/O error occurs while creating the InputStream.
     */
    InputStream inputStream() throws IOException;
    /**
     * Returns the metadata associated with the source of the document.
     * This could include details such as the source location, date of creation, owner, etc.
     *
     * @return A Metadata object containing information associated with the source of the document.
     */
    Metadata metadata();
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\document\DocumentSplitter.java

```
package dev.langchain4j.data.document;
import dev.langchain4j.data.segment.TextSegment;
import java.util.List;
import static java.util.stream.Collectors.toList;
/**
 * Defines the interface for splitting a document into text segments.
 * This is necessary as LLMs have a limited context window, making it
impossible to send the entire document at once.
 * Therefore, the document should first be split into segments, and only the
relevant segments should be sent to LLM.
 */
public interface DocumentSplitter {
    /**
     * Splits a single Document into a list of TextSegment objects.
     * The metadata is typically copied from the document and enriched with
segment-specific information,
     * such as position in the document, page number, etc.
     *
     * @param document The Document to be split.
     * @return A list of TextSegment objects derived from the input Document.
     */
    List<TextSegment> split(Document document);
    /**
     * Splits a list of Documents into a list of TextSegment objects.
     * This is a convenience method that calls the split method for each
Document in the list.
     *
     * @param documents The list of Documents to be split.
     * @return A list of TextSegment objects derived from the input Documents.
     */
    default List<TextSegment> splitAll(List<Document> documents) {
        return documents.stream()
                .flatMap(document -> split(document).stream())
                .collect(toList());
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\document\DocumentTransformer.java

```
package dev.langchain4j.data.document;
import java.util.List;
import java.util.Objects;
import static java.util.stream.Collectors.toList;
/**
 * Defines the interface for transforming a {@link Document}.
 * Implementations can perform a variety of tasks such as transforming, filtering, enriching, etc.
 */
public interface DocumentTransformer {
    /**
     * Transforms a provided document.
     *
     * @param document The document to be transformed.
     * @return The transformed document, or null if the document should be filtered out.
     */
    Document transform(Document document);
    /**
     * Transforms all the provided documents.
     *
     * @param documents A list of documents to be transformed.
     * @return A list of transformed documents. The length of this list may be shorter or longer than the original list. Returns an empty list if all documents were filtered out.
     */
    default List<Document> transformAll(List<Document> documents) {
        return documents.stream()
                .map(this::transform)
                .filter(Objects::nonNull)
                .collect(toList());
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\document\Metadata.java

```
package dev.langchain4j.data.document;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Represents metadata of a Document or a TextSegment.
 * The metadata is stored in a key-value map, where both keys and values are
strings.
 * For a Document, the metadata could include information such as the source,
creation date,
 * owner, or any other relevant details.
 * For a TextSegment, in addition to metadata copied from a document, it can
also include segment-specific information,
 * such as the page number, the position of the segment within the document,
chapter, etc.
 */
public class Metadata {
    private final Map<String, String> metadata;
    public Metadata() {
        this(new HashMap<>());
    }
    public Metadata(Map<String, String> metadata) {
        this.metadata = ensureNotNull(metadata, "metadata");
    }
    public String get(String key) {
        return metadata.get(key);
    }
    public Metadata add(String key, Object value) {
        this.metadata.put(key, value.toString());
        return this;
    }
    public Metadata remove(String key) {
        this.metadata.remove(key);
        return this;
    }
    public Metadata copy() {
        return new Metadata(new HashMap<>(metadata));
    }
    public Map<String, String> asMap() {
        return new HashMap<>(metadata);
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Metadata that = (Metadata) o;
        return Objects.equals(this.metadata, that.metadata);
    }
    @Override
    public int hashCode() {
        return Objects.hash(metadata);
    }
    @Override
    public String toString() {
        return "Metadata {" +
                " metadata = " + metadata +
                " }";
```

```
    }
    public static Metadata from(String key, Object value) {
        return new Metadata().add(key, value);
    }
    public static Metadata from(Map<String, String> metadata) {
        return new Metadata(metadata);
    }
    public static Metadata metadata(String key, Object value) {
        return from(key, value);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\embedding\Embedding.java

```
package dev.langchain4j.data.embedding;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Represents a dense vector embedding of a text.
 * This class encapsulates a float array that captures the "meaning" or
semantic information of the text.
 * Texts with similar meanings will have their vectors located close to each
other in the embedding space.
 * The embeddings are typically created by embedding models.
 * @see dev.langchain4j.model.embedding.EmbeddingModel
 */
public class Embedding {
    private final float[] vector;
    public Embedding(float[] vector) {
        this.vector = ensureNotNull(vector, "vector");
    }
    public float[] vector() {
        return vector;
    }
    public List<Float> vectorAsList() {
        List<Float> list = new ArrayList<>(vector.length);
        for (float f : vector) {
            list.add(f);
        }
        return list;
    }
    public int dimensions() {
        return vector.length;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Embedding that = (Embedding) o;
        return Arrays.equals(this.vector, that.vector);
    }
    @Override
    public int hashCode() {
        return Arrays.hashCode(vector);
    }
    @Override
    public String toString() {
        return "Embedding {" +
                " vector = " + Arrays.toString(vector) +
                " }";
    }
    public static Embedding from(float[] vector) {
        return new Embedding(vector);
    }
    public static Embedding from(List<Float> vector) {
        float[] array = new float[vector.size()];
        for (int i = 0; i < vector.size(); i++) {
            array[i] = vector.get(i);
        }
        return new Embedding(array);
```

```
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\AiMessage.java

```
package dev.langchain4j.data.message;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import java.util.Objects;
import static dev.langchain4j.data.message.ChatMessageType.AI;
import static dev.langchain4j.internal.Utils.quoted;
/**
 * Represents a response message from an AI (language model).
 * The message can contain either a textual response or a request to execute
a tool.
 * In the case of tool execution, the response to this message should be a
{@link ToolExecutionResultMessage}.
 */
public class AiMessage extends ChatMessage {
    private final ToolExecutionRequest toolExecutionRequest;
    public AiMessage(String text) {
        this(text, null);
    }
    public AiMessage(String text, ToolExecutionRequest toolExecutionRequest) {
        super(text);
        this.toolExecutionRequest = toolExecutionRequest;
    }
    public ToolExecutionRequest toolExecutionRequest() {
        return toolExecutionRequest;
    }
    @Override
    public ChatMessageType type() {
        return AI;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        AiMessage that = (AiMessage) o;
        return Objects.equals(this.text, that.text)
                && Objects.equals(this.toolExecutionRequest,
that.toolExecutionRequest);
    }
    @Override
    public int hashCode() {
        return Objects.hash(text, toolExecutionRequest);
    }
    @Override
    public String toString() {
        return "AiMessage {" +
                " text = " + quoted(text) +
                " toolExecutionRequest = " + toolExecutionRequest +
                " }";
    }
    public static AiMessage from(String text) {
        return new AiMessage(text);
    }
    public static AiMessage from(ToolExecutionRequest toolExecutionRequest) {
        return new AiMessage(null, toolExecutionRequest);
    }
    public static AiMessage aiMessage(String text) {
        return from(text);
    }
    public static AiMessage aiMessage(ToolExecutionRequest
```

```
toolExecutionRequest) {
        return from(toolExecutionRequest);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\ChatMessage.java

```
package dev.langchain4j.data.message;
public abstract class ChatMessage {
    protected final String text;
    ChatMessage(String text) {
        this.text = text;
    }
    public String text() {
        return text;
    }
    public abstract ChatMessageType type();
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\ChatMessageDeserializer.java

```
package dev.langchain4j.data.message;
import com.google.gson.reflect.TypeToken;
import java.util.List;
import static dev.langchain4j.data.message.ChatMessageSerializer.GSON;
import static java.util.Collections.emptyList;
public class ChatMessageDeserializer {
    /**
     * Deserializes a JSON string into a {@link ChatMessage}.
     *
     * @param json The JSON string representing a chat message.
     * @return A {@link ChatMessage} deserialized from the provided JSON
string.
     * @see ChatMessageSerializer For details on serialization.
     */
    public static ChatMessage messageFromJson(String json) {
        return GSON.fromJson(json, ChatMessage.class);
    }
    /**
     * Deserializes a JSON string into a list of {@link ChatMessage}.
     *
     * @param json The JSON string containing an array of chat messages.
     * @return A list of {@link ChatMessage} deserialized from the provided
JSON string.
     * @see ChatMessageSerializer For details on serialization.
     */
    public static List<ChatMessage> messagesFromJson(String json) {
        List<ChatMessage> messages = GSON.fromJson(json, new
TypeToken<List<ChatMessage>>() {
        }.getType());
        return messages == null ? emptyList() : messages;
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\ChatMessageSerializer.java

```
package dev.langchain4j.data.message;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import java.util.List;
public class ChatMessageSerializer {
    static final Gson GSON = new GsonBuilder()
            .registerTypeAdapter(ChatMessage.class, new
GsonChatMessageAdapter())
            .registerTypeAdapter(SystemMessage.class, new
GsonChatMessageAdapter())
            .registerTypeAdapter(UserMessage.class, new
GsonChatMessageAdapter())
            .registerTypeAdapter(AiMessage.class, new
GsonChatMessageAdapter())
            .registerTypeAdapter(ToolExecutionResultMessage.class, new
GsonChatMessageAdapter())
            .create();
    /**
     * Serializes a chat message into a JSON string.
     *
     * @param message Chat message to be serialized.
     * @return A JSON string with the contents of the message.
     * @see ChatMessageDeserializer For details on deserialization.
     */
    public static String messageToJson(ChatMessage message) {
        return GSON.toJson(message);
    }
    /**
     * Serializes a list of chat messages into a JSON string.
     *
     * @param messages The list of chat messages to be serialized.
     * @return A JSON string representing provided chat messages.
     * @see ChatMessageDeserializer For details on deserialization.
     */
    public static String messagesToJson(List<ChatMessage> messages) {
        return GSON.toJson(messages);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\ChatMessageType.java

```
package dev.langchain4j.data.message;
import static dev.langchain4j.internal.Exceptions.illegalArgument;
public enum ChatMessageType {
    SYSTEM,
    USER,
    AI,
    TOOL_EXECUTION_RESULT;
    public static Class<? extends ChatMessage> classOf(ChatMessageType type) {
        switch (type) {
            case SYSTEM:
                return SystemMessage.class;
            case USER:
                return UserMessage.class;
            case AI:
                return AiMessage.class;
            case TOOL_EXECUTION_RESULT:
                return ToolExecutionResultMessage.class;
            default:
                throw illegalArgument("Unknown ChatMessageType: %s", type);
        }
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\GsonChatMessageAdapter.java

```
package dev.langchain4j.data.message;
import com.google.gson.*;
import java.lang.reflect.Type;
class GsonChatMessageAdapter implements JsonDeserializer<ChatMessage>,
JsonSerializer<ChatMessage> {
    private static final Gson GSON = new Gson();
    private static final String CHAT_MESSAGE_TYPE = "type"; // do not change,
will break backward compatibility!
    @Override
    public JsonElement serialize(ChatMessage chatMessage, Type ignored,
JsonSerializationContext context) {
        JsonObject messageJsonObject =
GSON.toJsonTree(chatMessage).getAsJsonObject();
        messageJsonObject.addProperty(CHAT_MESSAGE_TYPE,
chatMessage.type().toString());
        return messageJsonObject;
    }
    @Override
    public ChatMessage deserialize(JsonElement messageJsonElement, Type
ignored, JsonDeserializationContext context) throws JsonParseException {
        String chatMessageTypeString =
messageJsonElement.getAsJsonObject().get(CHAT_MESSAGE_TYPE).getAsString();
        ChatMessageType chatMessageType =
ChatMessageType.valueOf(chatMessageTypeString);
        return GSON.fromJson(messageJsonElement,
ChatMessageType.classOf(chatMessageType));
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\SystemMessage.java

```
package dev.langchain4j.data.message;
import java.util.Objects;
import static dev.langchain4j.data.message.ChatMessageType.SYSTEM;
import static dev.langchain4j.internal.Utils.quoted;
/**
 * Represents a system message, typically defined by a developer.
 * This type of message usually provides instructions regarding the AI's
actions, such as its behavior or response style.
 */
public class SystemMessage extends ChatMessage {
    public SystemMessage(String text) {
        super(text);
    }
    @Override
    public ChatMessageType type() {
        return SYSTEM;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        SystemMessage that = (SystemMessage) o;
        return Objects.equals(this.text, that.text);
    }
    @Override
    public int hashCode() {
        return Objects.hash(text);
    }
    @Override
    public String toString() {
        return "SystemMessage {" +
                " text = " + quoted(text) +
                " }";
    }
    public static SystemMessage from(String text) {
        return new SystemMessage(text);
    }
    public static SystemMessage systemMessage(String text) {
        return from(text);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\ToolExecutionResultMessage.java

```
package dev.langchain4j.data.message;
import java.util.Objects;
import static
dev.langchain4j.data.message.ChatMessageType.TOOL_EXECUTION_RESULT;
import static dev.langchain4j.internal.Utils.quoted;
/**
 * Represents the result of a tool execution. Tool execution requests come
from a previous AiMessage.
 */
public class ToolExecutionResultMessage extends ChatMessage {
    private final String toolName;
    public ToolExecutionResultMessage(String toolName, String
toolExecutionResult) {
        super(toolExecutionResult);
        this.toolName = toolName;
    }
    public String toolName() {
        return toolName;
    }
    @Override
    public ChatMessageType type() {
        return TOOL_EXECUTION_RESULT;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        ToolExecutionResultMessage that = (ToolExecutionResultMessage) o;
        return Objects.equals(this.toolName, that.toolName)
                && Objects.equals(this.text, that.text);
    }
    @Override
    public int hashCode() {
        return Objects.hash(toolName, text);
    }
    @Override
    public String toString() {
        return "ToolExecutionResultMessage {" +
                " toolName = " + quoted(toolName) +
                " text = " + quoted(text) +
                " }";
    }
    public static ToolExecutionResultMessage from(String toolName, String
toolExecutionResult) {
        return new ToolExecutionResultMessage(toolName, toolExecutionResult);
    }
    public static ToolExecutionResultMessage
toolExecutionResultMessage(String toolName, String toolExecutionResult) {
        return from(toolName, toolExecutionResult);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\message\UserMessage.java

```
package dev.langchain4j.data.message;
import java.util.Objects;
import static dev.langchain4j.data.message.ChatMessageType.USER;
import static dev.langchain4j.internal.Utils.quoted;
/**
 * Represents a message from a user, typically an end user of the application.
 */
public class UserMessage extends ChatMessage {
    private final String name;
    public UserMessage(String text) {
        this(null, text);
    }
    public UserMessage(String name, String text) {
        super(text);
        this.name = name;
    }
    public String name() {
        return name;
    }
    @Override
    public ChatMessageType type() {
        return USER;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        UserMessage that = (UserMessage) o;
        return Objects.equals(this.name, that.name)
                && Objects.equals(this.text, that.text);
    }
    @Override
    public int hashCode() {
        return Objects.hash(name, text);
    }
    @Override
    public String toString() {
        return "UserMessage {" +
                " name = " + quoted(name) +
                " text = " + quoted(text) +
                " }";
    }
    public static UserMessage from(String text) {
        return new UserMessage(text);
    }
    public static UserMessage from(String name, String text) {
        return new UserMessage(name, text);
    }
    public static UserMessage userMessage(String text) {
        return from(text);
    }
    public static UserMessage userMessage(String name, String text) {
        return from(name, text);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\segment\TextSegment.java

```
package dev.langchain4j.data.segment;
import dev.langchain4j.data.document.Metadata;
import java.util.Objects;
import static dev.langchain4j.internal.Utils.quoted;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Represents a semantically meaningful segment (chunk/piece/fragment) of a
larger entity such as a document or chat conversation.
 * This might be a sentence, a paragraph, or any other discrete unit of text
that carries meaning.
 * This class encapsulates a piece of text and its associated metadata.
 */
public class TextSegment {
    private final String text;
    private final Metadata metadata;
    public TextSegment(String text, Metadata metadata) {
        this.text = ensureNotBlank(text, "text");
        this.metadata = ensureNotNull(metadata, "metadata");
    }
    public String text() {
        return text;
    }
    public Metadata metadata() {
        return metadata;
    }
    public String metadata(String key) {
        return metadata.get(key);
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        TextSegment that = (TextSegment) o;
        return Objects.equals(this.text, that.text)
                && Objects.equals(this.metadata, that.metadata);
    }
    @Override
    public int hashCode() {
        return Objects.hash(text, metadata);
    }
    @Override
    public String toString() {
        return "TextSegment {" +
                " text = " + quoted(text) +
                " metadata = " + metadata.asMap() +
                " }";
    }
    public static TextSegment from(String text) {
        return new TextSegment(text, new Metadata());
    }
    public static TextSegment from(String text, Metadata metadata) {
        return new TextSegment(text, metadata);
    }
    public static TextSegment textSegment(String text) {
        return from(text);
    }
    public static TextSegment textSegment(String text, Metadata metadata) {
```

```
        return from(text, metadata);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\data\segment\TextSegmentTransformer.java

```
package dev.langchain4j.data.segment;
import java.util.List;
import java.util.Objects;
import static java.util.stream.Collectors.toList;
/**
 * Defines the interface for transforming a {@link TextSegment}.
 * Implementations can perform a variety of tasks such as transforming,
filtering, enriching, etc.
 */
public interface TextSegmentTransformer {
    /**
     * Transforms a provided segment.
     *
     * @param segment The segment to be transformed.
     * @return The transformed segment, or null if the segment should be
filtered out.
     */
    TextSegment transform(TextSegment segment);
    /**
     * Transforms all the provided segments.
     *
     * @param segments A list of segments to be transformed.
     * @return A list of transformed segments. The length of this list may be
shorter or longer than the original list. Returns an empty list if all
segments were filtered out.
     */
    default List<TextSegment> transformAll(List<TextSegment> segments) {
        return segments.stream()
                .map(this::transform)
                .filter(Objects::nonNull)
                .collect(toList());
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\internal\Exceptions.java

```
package dev.langchain4j.internal;
public class Exceptions {
    public static IllegalArgumentException illegalArgument(String format,
Object... args) {
        return new IllegalArgumentException(String.format(format, args));
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\internal\Json.java

```
package dev.langchain4j.internal;
import static java.time.format.DateTimeFormatter.ISO_LOCAL_DATE;
import static java.time.format.DateTimeFormatter.ISO_LOCAL_DATE_TIME;
import com.google.gson.*;
import com.google.gson.stream.JsonWriter;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.time.LocalDate;
import java.time.LocalDateTime;
public class Json {
  private static final Gson GSON = new GsonBuilder()
    .setPrettyPrinting()
    .registerTypeAdapter(
      LocalDate.class,
      (JsonSerializer<LocalDate>) (localDate, type, context) -> new
JsonPrimitive(localDate.format(ISO_LOCAL_DATE))
    )
    .registerTypeAdapter(
      LocalDate.class,
      (JsonDeserializer<LocalDate>) (json, type, context) ->
LocalDate.parse(json.getAsString(), ISO_LOCAL_DATE)
    )
    .registerTypeAdapter(
      LocalDateTime.class,
      (JsonSerializer<LocalDateTime>) (localDateTime, type, context) ->
        new JsonPrimitive(localDateTime.format(ISO_LOCAL_DATE_TIME))
    )
    .registerTypeAdapter(
      LocalDateTime.class,
      (JsonDeserializer<LocalDateTime>) (json, type, context) ->
        LocalDateTime.parse(json.getAsString(), ISO_LOCAL_DATE_TIME)
    )
    .create();
  public static String toJson(Object o) {
    return GSON.toJson(o);
  }
  public static <T> T fromJson(String json, Class<T> type) {
    return GSON.fromJson(json, type);
  }
  public static InputStream toInputStream(Object o, Class type) throws
IOException {
    try (
      ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
      OutputStreamWriter outputStreamWriter = new
OutputStreamWriter(byteArrayOutputStream, StandardCharsets.UTF_8);
      JsonWriter jsonWriter = new JsonWriter(outputStreamWriter)
    ) {
      GSON.toJson(o, type, jsonWriter);
      jsonWriter.flush();
      return new ByteArrayInputStream(byteArrayOutputStream.toByteArray());
    }
  }
}
```

langchain4j-core\src\main\java\dev\langchain4j\internal\RetryUtils.java

```
package dev.langchain4j.internal;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.concurrent.Callable;
import static java.lang.String.format;
public class RetryUtils {
    private static final Logger log =
LoggerFactory.getLogger(RetryUtils.class);
    /**
     * This method attempts to execute a given action up to a specified
number of times with a 1-second delay.
     * If the action fails on all attempts, it throws a RuntimeException.
     *
     * @param action      The action to be executed.
     * @param maxAttempts The maximum number of attempts to execute the
action.
     * @return The result of the action if it is successful.
     * @throws RuntimeException if the action fails on all attempts.
     */
    public static <T> T withRetry(Callable<T> action, int maxAttempts) {
        for (int attempt = 1; attempt <= maxAttempts; attempt++) {
            try {
                return action.call();
            } catch (Exception e) {
                if (attempt == maxAttempts) {
                    throw new RuntimeException(e);
                }
                log.warn(format("Exception was thrown on attempt %s of %s",
attempt, maxAttempts), e);
                try {
                    Thread.sleep(1000); // TODO make configurable
                } catch (InterruptedException ignored) {
                }
            }
        }
        throw new RuntimeException("Failed after " + maxAttempts + "
attempts");
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\internal\Utils.java

```
package dev.langchain4j.internal;
import static java.nio.charset.StandardCharsets.UTF_8;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Collection;
import java.util.UUID;
public class Utils {
  public static <T> T getOrDefault(T value, T defaultValue) {
    return value != null ? value : defaultValue;
  }
  public static boolean isNullOrBlank(String string) {
    return string == null || string.trim().isEmpty();
  }
  public static boolean isNotNullOrBlank(String string) {
    return !isNullOrBlank(string);
  }
  public static boolean areNotNullOrBlank(String... strings) {
    if (strings == null || strings.length == 0) {
      return false;
    }
    for (String string : strings) {
      if (isNullOrBlank(string)) {
        return false;
      }
    }
    return true;
  }
  public static boolean isCollectionEmpty(Collection<?> collection) {
    return collection == null || collection.isEmpty();
  }
  public static String repeat(String string, int times) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < times; i++) {
      sb.append(string);
    }
    return sb.toString();
  }
  public static String randomUUID() {
    return UUID.randomUUID().toString();
  }
  public static String generateUUIDFrom(String input) {
    try {
      byte[] hashBytes =
MessageDigest.getInstance("SHA-256").digest(input.getBytes(UTF_8));
      StringBuilder sb = new StringBuilder();
      for (byte b : hashBytes) sb.append(String.format("%02x", b));
      return UUID.nameUUIDFromBytes(sb.toString().getBytes(UTF_8)).toString();
    } catch (NoSuchAlgorithmException e) {
      throw new IllegalArgumentException(e);
    }
  }
  public static String quoted(String string) {
    if (string == null) {
      return "null";
    }
    return "\"" + string + "\"";
  }
  public static String firstChars(String string, int numberOfChars) {
```

```
    if (string == null) {
      return null;
    }
    return string.length() > numberOfChars ? string.substring(0,
numberOfChars) : string;
  }
}
```

langchain4j-core\src\main\java\dev\langchain4j\internal\ValidationUtils.java

```
package dev.langchain4j.internal;
import java.util.Collection;
import static dev.langchain4j.internal.Exceptions.illegalArgument;
public class ValidationUtils {
    public static <T> T ensureNotNull(T object, String name) {
        if (object == null) {
            throw illegalArgument("%s cannot be null", name);
        }
        return object;
    }
    public static <T extends Collection<?>> T ensureNotEmpty(T collection,
String name) {
        if (collection == null || collection.isEmpty()) {
            throw illegalArgument("%s cannot be null or empty", name);
        }
        return collection;
    }
    public static String ensureNotBlank(String string, String name) {
        if (string == null || string.trim().isEmpty()) {
            throw illegalArgument("%s cannot be null or blank", name);
        }
        return string;
    }
    public static void ensureTrue(boolean expression, String msg) {
        if (!expression) {
            throw illegalArgument(msg);
        }
    }
    public static int ensureGreaterThanZero(Integer i, String name) {
        if (i == null || i <= 0) {
            throw illegalArgument("%s must be greater than zero, but is: %s",
name, i);
        }
        return i;
    }
    public static double ensureBetween(Double d, double min, double max,
String name) {
        if (d == null || d < min || d > max) {
            throw illegalArgument("%s must be between %s and %s, but is: %s",
name, min, max, d);
        }
        return d;
    }
    public static int ensureBetween(Integer i, int min, int max, String name)
{
        if (i == null || i < min || i > max) {
            throw illegalArgument("%s must be between %s and %s, but is: %s",
name, min, max, i);
        }
        return i;
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\memory\ChatMemory.java

```
package dev.langchain4j.memory;
import dev.langchain4j.data.message.ChatMessage;
import java.util.List;
/**
 * Represents the memory (history) of a chat conversation.
 * Since language models do not keep the state of the conversation, it is
necessary to provide all previous messages
 * on every interaction with the language model.
 * {@link ChatMemory} helps with keeping track of the conversation and
ensuring that messages fit within language model's context window.
 */
public interface ChatMemory {
    /**
     * @return The ID of the {@link ChatMemory}.
     */
    Object id();
    /**
     * Adds a message to the chat memory.
     *
     * @param message The {@link ChatMessage} to add.
     */
    void add(ChatMessage message);
    /**
     * Retrieves messages from the chat memory.
     * Depending on the implementation, it may not return all previously
added messages,
     * but rather a subset, a summary, or a combination thereof.
     *
     * @return A list of {@link ChatMessage} objects that represent the
current state of the chat memory.
     */
    List<ChatMessage> messages();
    /**
     * Clears the chat memory.
     */
    void clear();
}
```

langchain4j-core\src\main\java\dev\langchain4j\MightChangeInTheFuture.java

```
package dev.langchain4j;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.*;
/**
 * Indicates that a class/constructor/method is experimental and might change
in the future.
 */
@Target({TYPE, CONSTRUCTOR, METHOD})
public @interface MightChangeInTheFuture {
    String value();
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\chat\ChatLanguageModel.java

```
package dev.langchain4j.model.chat;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.model.output.Response;
import java.util.List;
import static java.util.Arrays.asList;
/**
 * Represents a language model that has a chat interface.
 */
public interface ChatLanguageModel {
    /**
     * Generates a response from the model based on a message from a user.
     * This is a convenience method that receives the message from a user as
a String
     * and returns only the generated response.
     *
     * @param userMessage The message from the user.
     * @return The response generated by the model.
     */
    default String generate(String userMessage) {
        return generate(UserMessage.from(userMessage)).content().text();
    }
    /**
     * Generates a response from the model based on a sequence of messages.
     * Typically, the sequence contains messages in the following order:
     * System (optional) - User - AI - User - AI - User ...
     *
     * @param messages An array of messages.
     * @return The response generated by the model.
     */
    default Response<AiMessage> generate(ChatMessage... messages) {
        return generate(asList(messages));
    }
    /**
     * Generates a response from the model based on a sequence of messages.
     * Typically, the sequence contains messages in the following order:
     * System (optional) - User - AI - User - AI - User ...
     *
     * @param messages A list of messages.
     * @return The response generated by the model.
     */
    Response<AiMessage> generate(List<ChatMessage> messages);
    /**
     * Generates a response from the model based on a list of messages and a
list of tool specifications.
     * The response may either be a text message or a request to execute one
of the specified tools.
     * Typically, the list contains messages in the following order:
     * System (optional) - User - AI - User - AI - User ...
     *
     * @param messages           A list of messages.
     * @param toolSpecifications A list of tools that the model is allowed to
execute.
     *                           The model autonomously decides whether to
use any of these tools.
```

```
     * @return The response generated by the model.
     * {@link AiMessage} can contain either a textual response or a request
to execute one of the tools.
     */
    Response<AiMessage> generate(List<ChatMessage> messages,
List<ToolSpecification> toolSpecifications);
    /**
     * Generates a response from the model based on a list of messages and a
single tool specification.
     * The model is forced to execute the specified tool.
     * Typically, the list contains messages in the following order:
     * System (optional) - User - AI - User - AI - User ...
     *
     * @param messages          A list of messages.
     * @param toolSpecification The specification of a tool that must be
executed.
     *                          The model is forced to execute this tool.
     * @return The response generated by the model.
     * {@link AiMessage} contains a request to execute the specified tool.
     */
    Response<AiMessage> generate(List<ChatMessage> messages,
ToolSpecification toolSpecification);
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\chat\StreamingChatLanguageModel.java

```
package dev.langchain4j.model.chat;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import java.util.List;
import static java.util.Collections.singletonList;
/**
 * Represents a language model that has a chat interface and can stream a response one token at a time.
 */
public interface StreamingChatLanguageModel {
    /**
     * Generates a response from the model based on a message from a user.
     *
     * @param userMessage The message from the user.
     * @param handler     The handler for streaming the response.
     */
    default void generate(String userMessage, StreamingResponseHandler<AiMessage> handler) {
        generate(singletonList(UserMessage.from(userMessage)), handler);
    }
    /**
     * Generates a response from the model based on a sequence of messages.
     * Typically, the sequence contains messages in the following order:
     * System (optional) - User - AI - User - AI - User ...
     *
     * @param messages A list of messages.
     * @param handler  The handler for streaming the response.
     */
    void generate(List<ChatMessage> messages, StreamingResponseHandler<AiMessage> handler);
    /**
     * Generates a response from the model based on a list of messages and a list of tool specifications.
     * The response may either be a text message or a request to execute one of the specified tools.
     * Typically, the list contains messages in the following order:
     * System (optional) - User - AI - User - AI - User ...
     *
     * @param messages           A list of messages.
     * @param toolSpecifications A list of tools that the model is allowed to execute.
     *                           The model autonomously decides whether to use any of these tools.
     * @param handler            The handler for streaming the response.
     *                           {@link AiMessage} can contain either a textual response or a request to execute one of the tools.
     */
    void generate(List<ChatMessage> messages, List<ToolSpecification> toolSpecifications, StreamingResponseHandler<AiMessage> handler);
    void generate(List<ChatMessage> messages, ToolSpecification toolSpecification, StreamingResponseHandler<AiMessage> handler);
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\chat\TokenCountEstimator.java

```
package dev.langchain4j.model.chat;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.input.Prompt;
import java.util.List;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static java.util.Collections.singletonList;
/**
 * Represents an interface for estimating the count of tokens in various text types such as a text, message, prompt, text segment, etc.
 * This can be useful when it's necessary to know in advance the cost of processing a specified text by the LLM.
 */
public interface TokenCountEstimator {
    default int estimateTokenCount(String text) {
        return estimateTokenCount(userMessage(text));
    }
    default int estimateTokenCount(UserMessage userMessage) {
        return estimateTokenCount(singletonList(userMessage));
    }
    default int estimateTokenCount(Prompt prompt) {
        return estimateTokenCount(prompt.text());
    }
    default int estimateTokenCount(TextSegment textSegment) {
        return estimateTokenCount(textSegment.text());
    }
    int estimateTokenCount(List<ChatMessage> messages);
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\embedding\EmbeddingModel.java

```
package dev.langchain4j.model.embedding;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.output.Response;
import java.util.List;
import static java.util.Collections.singletonList;
/**
 * Represents a model that can convert a given text into an embedding (vector
representation of the text).
 */
public interface EmbeddingModel {
    default Response<Embedding> embed(String text) {
        return embed(TextSegment.from(text));
    }
    default Response<Embedding> embed(TextSegment textSegment) {
        Response<List<Embedding>> response =
embedAll(singletonList(textSegment));
        return Response.from(
                response.content().get(0),
                response.tokenUsage(),
                response.finishReason()
        );
    }
    Response<List<Embedding>> embedAll(List<TextSegment> textSegments);
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\embedding\TokenCountEstimator.java

```
package dev.langchain4j.model.embedding;
import dev.langchain4j.data.segment.TextSegment;
import java.util.List;
/**
 * Represents an interface for estimating the count of tokens in various texts, text segments, etc.
 * This can be useful when it's necessary to know in advance the cost of processing a specified text by the LLM.
 */
public interface TokenCountEstimator {
    int estimateTokenCount(String text);
    default int estimateTokenCount(TextSegment textSegment) {
        return estimateTokenCount(textSegment.text());
    }
    default int estimateTokenCount(List<TextSegment> textSegments) {
        int tokenCount = 0;
        for (TextSegment textSegment : textSegments) {
            tokenCount += estimateTokenCount(textSegment);
        }
        return tokenCount;
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\input\MustachePromptTemplateFactory.java

```
package dev.langchain4j.model.input;
import com.github.mustachejava.DefaultMustacheFactory;
import com.github.mustachejava.Mustache;
import com.github.mustachejava.MustacheFactory;
import dev.langchain4j.spi.prompt.PromptTemplateFactory;
import java.io.StringReader;
import java.io.StringWriter;
import java.util.Map;
class MustachePromptTemplateFactory implements PromptTemplateFactory {
    private static final MustacheFactory MUSTACHE_FACTORY = new
DefaultMustacheFactory();
    @Override
    public MustacheTemplate create(PromptTemplateFactory.Input input) {
        StringReader stringReader = new StringReader(input.getTemplate());
        return new MustacheTemplate(MUSTACHE_FACTORY.compile(stringReader,
input.getName()));
    }
    static class MustacheTemplate implements Template {
        private final Mustache mustache;
        MustacheTemplate(Mustache mustache) {
            this.mustache = mustache;
        }
        public String render(Map<String, Object> vars) {
            StringWriter writer = new StringWriter();
            mustache.execute(writer, vars);
            return writer.toString();
        }
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\input\Prompt.java

```
package dev.langchain4j.model.input;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.data.message.UserMessage;
import java.util.Objects;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static dev.langchain4j.data.message.SystemMessage.systemMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.internal.Utils.quoted;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
/**
 * Represents a prompt (an input text sent to the LLM).
 * A prompt usually contains instructions, contextual information, end-user
input, etc.
 * A Prompt is typically created by applying one or multiple values to a
PromptTemplate.
 */
public class Prompt {
    private final String text;
    public Prompt(String text) {
        this.text = ensureNotBlank(text, "text");
    }
    public String text() {
        return text;
    }
    public SystemMessage toSystemMessage() {
        return systemMessage(text);
    }
    public UserMessage toUserMessage() {
        return userMessage(text);
    }
    public AiMessage toAiMessage() {
        return aiMessage(text);
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Prompt that = (Prompt) o;
        return Objects.equals(this.text, that.text);
    }
    @Override
    public int hashCode() {
        return Objects.hash(text);
    }
    @Override
    public String toString() {
        return "Prompt {" +
                " text = " + quoted(text) +
                " }";
    }
    public static Prompt from(String text) {
        return new Prompt(text);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\input\PromptTemplate.java

```
package dev.langchain4j.model.input;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
import static java.util.Collections.singletonMap;
import dev.langchain4j.spi.ServiceHelper;
import dev.langchain4j.spi.prompt.PromptTemplateFactory;
import java.time.Clock;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.HashMap;
import java.util.Map;
/**
 * Represents a template of a prompt that can be reused multiple times.
 * A template typically contains one or more variables (placeholders) defined as {{variable_name}} that are
 * replaced with actual values to produce a Prompt.
 * Special variables {{current_date}}, {{current_time}}, and {{current_date_time}} are automatically
 * filled with LocalDate.now(), LocalTime.now(), and LocalDateTime.now() respectively.
 * This class uses the Mustache templating engine under the hood, so all Mustache syntax and features are supported.
 */
public class PromptTemplate {
    private static final PromptTemplateFactory FACTORY = factory();
    private static PromptTemplateFactory factory() {
        for (PromptTemplateFactory factory : ServiceHelper.loadFactories(PromptTemplateFactory.class)) {
            return factory;
        }
        // fallback to the default
        return new MustachePromptTemplateFactory();
    }
    private final PromptTemplateFactory.Template template;
    private final Clock clock;
    public PromptTemplate(String template) {
        this(template, Clock.systemDefaultZone());
    }
    PromptTemplate(String template, Clock clock) {
        this.template = FACTORY.create(new PromptTemplateFactory.Input() {
            @Override
            public String getTemplate() {
                return template;
            }
            @Override
            public String getName() {
                return "template";
            }
        });
        this.clock = ensureNotNull(clock, "clock");
    }
    /**
     * Applies a value to a template containing a single variable. The single variable should have the name {{it}}.
     *
     * @param value The value that will be injected in place of the {{it}} placeholder in the template.
```

```
     * @return A Prompt object where the {{it}} placeholder in the template
has been replaced by the provided value.
     */
    public Prompt apply(Object value) {
        return apply(singletonMap("it", value));
    }
    /**
     * Applies multiple values to a template containing multiple variables.
     *
     * @param variables A map of variable names to values that will be
injected in place of the corresponding placeholders in the template.
     * @return A Prompt object where the placeholders in the template have
been replaced by the provided values.
     */
    public Prompt apply(Map<String, Object> variables) {
        return
Prompt.from(template.render(injectDateTimeVariables(variables)));
    }
    private Map<String, Object> injectDateTimeVariables(Map<String, Object>
variables) {
        Map<String, Object> variablesCopy = new HashMap<>(variables);
        variablesCopy.put("current_date", LocalDate.now(clock));
        variablesCopy.put("current_time", LocalTime.now(clock));
        variablesCopy.put("current_date_time", LocalDateTime.now(clock));
        return variablesCopy;
    }
    public static PromptTemplate from(String template) {
        return new PromptTemplate(template);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\input\structured\DefaultStructuredPromptFactory.java

```
package dev.langchain4j.model.input.structured;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.ToNumberPolicy;
import com.google.gson.reflect.TypeToken;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.PromptTemplate;
import dev.langchain4j.spi.prompt.structured.StructuredPromptFactory;
import java.util.Map;
public class DefaultStructuredPromptFactory implements
StructuredPromptFactory {
    private static final Gson GSON = new GsonBuilder().setObjectToNumberStrate
gy(ToNumberPolicy.LONG_OR_DOUBLE).create();
    public Prompt toPrompt(Object structuredPrompt) {
        validate(structuredPrompt);
        StructuredPrompt annotation =
structuredPrompt.getClass().getAnnotation(StructuredPrompt.class);
        String promptTemplateString = String.join(annotation.delimiter(),
annotation.value());
        PromptTemplate promptTemplate =
PromptTemplate.from(promptTemplateString);
        Map<String, Object> variables = extractVariables(structuredPrompt);
        return promptTemplate.apply(variables);
    }
    private void validate(Object structuredPrompt) {
        if (structuredPrompt == null) {
            throw new IllegalArgumentException("Structured prompt cannot be
null");
        }
        String structuredPromptClassName =
structuredPrompt.getClass().getName();
        StructuredPrompt structuredPromptAnnotation =
structuredPrompt.getClass().getAnnotation(StructuredPrompt.class);
        if (structuredPromptAnnotation == null) {
            throw new IllegalArgumentException(
                    String.format(
                            "%s should be annotated with @StructuredPrompt to
be used as a structured prompt",
                            structuredPromptClassName
                    )
            );
        }
    }
    private static Map<String, Object> extractVariables(Object
structuredPrompt) {
        String json = GSON.toJson(structuredPrompt);
        TypeToken<Map<String, Object>> mapType = new TypeToken<Map<String,
Object>>() {};
        return GSON.fromJson(json, mapType);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\input\structured\StructuredPrompt.java

```
package dev.langchain4j.model.input.structured;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
@Target(TYPE)
@Retention(RUNTIME)
public @interface StructuredPrompt {
    /**
     * Prompt template can be defined in one line or multiple lines.
     * If the template is defined in multiple lines, the lines will be joined with a delimiter defined below.
     */
    String[] value();
    String delimiter() default "\n";
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\input\structured\StructuredPromptProcessor.java

```
package dev.langchain4j.model.input.structured;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.spi.ServiceHelper;
import dev.langchain4j.spi.prompt.structured.StructuredPromptFactory;
public class StructuredPromptProcessor {
    private static final StructuredPromptFactory FACTORY = factory();
    private static StructuredPromptFactory factory() {
        for (StructuredPromptFactory factory :
ServiceHelper.loadFactories(StructuredPromptFactory.class)) {
            return factory;
        }
        // fallback to the default
        return new DefaultStructuredPromptFactory();
    }
    public static Prompt toPrompt(Object structuredPrompt) {
        return FACTORY.toPrompt(structuredPrompt);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\language\LanguageModel.java

```
package dev.langchain4j.model.language;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.output.Response;
/**
 * Represents a language model that has a simple text interface (as opposed to a chat interface).
 * It is recommended to use the {@link dev.langchain4j.model.chat.ChatLanguageModel} instead,
 * as it offers more features.
 */
public interface LanguageModel {
    Response<String> generate(String prompt);
    default Response<String> generate(Prompt prompt) {
        return generate(prompt.text());
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\language\StreamingLanguageModel.java

```
package dev.langchain4j.model.language;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.input.Prompt;
/**
 * Represents a language model that has a simple text interface (as opposed
to a chat interface)
 * and can stream a response one token at a time.
 * It is recommended to use the {@link
dev.langchain4j.model.chat.StreamingChatLanguageModel} instead,
 * as it offers more features.
 */
public interface StreamingLanguageModel {
    void generate(String prompt, StreamingResponseHandler<String> handler);
    default void generate(Prompt prompt, StreamingResponseHandler<String>
handler) {
        generate(prompt.text(), handler);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\language\TokenCountEstimator.java

```
package dev.langchain4j.model.language;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.input.Prompt;
/**
 * Represents an interface for estimating the count of tokens in various text types such as a text, prompt, text segment, etc.
 * This can be useful when it's necessary to know in advance the cost of processing a specified text by the LLM.
 */
public interface TokenCountEstimator {
    int estimateTokenCount(String text);
    default int estimateTokenCount(Prompt prompt) {
        return estimateTokenCount(prompt.text());
    }
    default int estimateTokenCount(TextSegment textSegment) {
        return estimateTokenCount(textSegment.text());
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\moderation\Moderation.java

```
package dev.langchain4j.model.moderation;
import java.util.Objects;
import static dev.langchain4j.internal.Utils.quoted;
public class Moderation {
    private final boolean flagged;
    private final String flaggedText;
    public Moderation() {
        this.flagged = false;
        this.flaggedText = null;
    }
    public Moderation(String flaggedText) {
        this.flagged = true;
        this.flaggedText = flaggedText;
    }
    public boolean flagged() {
        return flagged;
    }
    public String flaggedText() {
        return flaggedText;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Moderation that = (Moderation) o;
        return this.flagged == that.flagged
                && Objects.equals(this.flaggedText, that.flaggedText);
    }
    @Override
    public int hashCode() {
        return Objects.hash(flagged, flaggedText);
    }
    @Override
    public String toString() {
        return "Moderation {" +
                " flagged = " + flagged +
                ", flaggedText = " + quoted(flaggedText) +
                " }";
    }
    public static Moderation flagged(String flaggedText) {
        return new Moderation(flaggedText);
    }
    public static Moderation notFlagged() {
        return new Moderation();
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\moderation\ModerationModel.java

```
package dev.langchain4j.model.moderation;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.output.Response;
import java.util.List;
public interface ModerationModel {
    Response<Moderation> moderate(String text);
    Response<Moderation> moderate(Prompt prompt);
    Response<Moderation> moderate(ChatMessage message);
    Response<Moderation> moderate(List<ChatMessage> messages);
    Response<Moderation> moderate(TextSegment textSegment);
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\output\FinishReason.java

```
package dev.langchain4j.model.output;
public enum FinishReason {
    STOP,
    LENGTH,
    TOOL_EXECUTION,
    CONTENT_FILTER
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\output\OutputParser.java

```
package dev.langchain4j.model.output;
public interface OutputParser<T> {
    T parse(String text);
    String formatInstructions();
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\output\Response.java

```
package dev.langchain4j.model.output;
import java.util.Objects;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Represents the response from various types of models, including language,
chat, embedding, and moderation models.
 * This class encapsulates the generated content, token usage statistics, and
finish reason.
 *
 * @param <T> The type of content generated by the model.
 */
public class Response<T> {
    private final T content;
    private final TokenUsage tokenUsage;
    private final FinishReason finishReason;
    public Response(T content) {
        this(content, null, null);
    }
    public Response(T content, TokenUsage tokenUsage, FinishReason
finishReason) {
        this.content = ensureNotNull(content, "content");
        this.tokenUsage = tokenUsage;
        this.finishReason = finishReason;
    }
    public T content() {
        return content;
    }
    public TokenUsage tokenUsage() {
        return tokenUsage;
    }
    public FinishReason finishReason() {
        return finishReason;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Response<?> that = (Response<?>) o;
        return Objects.equals(this.content, that.content)
                && Objects.equals(this.tokenUsage, that.tokenUsage)
                && Objects.equals(this.finishReason, that.finishReason);
    }
    @Override
    public int hashCode() {
        return Objects.hash(content, tokenUsage, finishReason);
    }
    @Override
    public String toString() {
        return "Response {" +
                " content = " + content +
                ", tokenUsage = " + tokenUsage +
                ", finishReason = " + finishReason +
                " }";
    }
    public static <T> Response<T> from(T content) {
        return new Response<>(content);
    }
    public static <T> Response<T> from(T content, TokenUsage tokenUsage) {
```

```
        return new Response<>(content, tokenUsage, null);
    }
    public static <T> Response<T> from(T content, TokenUsage tokenUsage,
FinishReason finishReason) {
        return new Response<>(content, tokenUsage, finishReason);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\output\structured\Description.java

```
package dev.langchain4j.model.output.structured;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
@Target(FIELD)
@Retention(RUNTIME)
public @interface Description {
    /**
     * The description can be defined in one line or multiple lines.
     * If the description is defined in multiple lines, the lines will be
joined with a space (" ") automatically.
     */
    String[] value();
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\output\TokenUsage.java

```
package dev.langchain4j.model.output;
import java.util.Objects;
import static dev.langchain4j.internal.Utils.getOrDefault;
public class TokenUsage {
    private final Integer inputTokenCount;
    private final Integer outputTokenCount;
    private final Integer totalTokenCount;
    public TokenUsage() {
        this(null);
    }
    public TokenUsage(Integer inputTokenCount) {
        this(inputTokenCount, null);
    }
    public TokenUsage(Integer inputTokenCount, Integer outputTokenCount) {
        this(inputTokenCount, outputTokenCount, sum(inputTokenCount,
outputTokenCount));
    }
    public TokenUsage(Integer inputTokenCount, Integer outputTokenCount,
Integer totalTokenCount) {
        this.inputTokenCount = inputTokenCount;
        this.outputTokenCount = outputTokenCount;
        this.totalTokenCount = totalTokenCount;
    }
    public Integer inputTokenCount() {
        return inputTokenCount;
    }
    public Integer outputTokenCount() {
        return outputTokenCount;
    }
    public Integer totalTokenCount() {
        return totalTokenCount;
    }
    public TokenUsage add(TokenUsage that) {
        return new TokenUsage(
                sum(this.inputTokenCount, that.inputTokenCount),
                sum(this.outputTokenCount, that.outputTokenCount),
                sum(this.totalTokenCount, that.totalTokenCount)
        );
    }
    private static Integer sum(Integer first, Integer second) {
        if (first == null && second == null) {
            return null;
        }
        return getOrDefault(first, 0) + getOrDefault(second, 0);
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        TokenUsage that = (TokenUsage) o;
        return Objects.equals(this.inputTokenCount, that.inputTokenCount)
                && Objects.equals(this.outputTokenCount,
that.outputTokenCount)
                && Objects.equals(this.totalTokenCount, that.totalTokenCount);
    }
    @Override
    public int hashCode() {
        return Objects.hash(inputTokenCount, outputTokenCount,
```

```
totalTokenCount);
    }
    @Override
    public String toString() {
        return "TokenUsage {" +
                " inputTokenCount = " + inputTokenCount +
                ", outputTokenCount = " + outputTokenCount +
                ", totalTokenCount = " + totalTokenCount +
                " }";
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\StreamingResponseHandler.java

```
package dev.langchain4j.model;
import dev.langchain4j.model.output.Response;
public interface StreamingResponseHandler<T> {
    /**
     * Invoked each time the language model generates a new token in a textual response.
     * If the model executes a tool instead, this method will not be invoked; {@link #onComplete} will be invoked instead.
     *
     * @param token The newly generated token, which is a part of the complete response.
     */
    void onNext(String token);
    /**
     * Invoked when the language model has finished streaming a response.
     * If the model executes a tool, it is accessible via {@link dev.langchain4j.data.message.AiMessage#toolExecutionRequest()}.
     *
     * @param response The complete response generated by the language model.
     *                 For textual responses, it contains all tokens from {@link #onNext} concatenated.
     */
    default void onComplete(Response<T> response) {
    }
    /**
     * This method is invoked when an error occurs during streaming.
     *
     * @param error The error that occurred
     */
    void onError(Throwable error);
}
```

langchain4j-core\src\main\java\dev\langchain4j\model\Tokenizer.java

```
package dev.langchain4j.model;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.ChatMessage;
import java.util.ArrayList;
import java.util.List;
import static
dev.langchain4j.agent.tool.ToolSpecifications.toolSpecificationsFrom;
import static java.util.Collections.singletonList;
public interface Tokenizer {
    int estimateTokenCountInText(String text);
    int estimateTokenCountInMessage(ChatMessage message);
    int estimateTokenCountInMessages(Iterable<ChatMessage> messages);
    default int estimateTokenCountInTools(Object objectWithTools) {
        return estimateTokenCountInTools(singletonList(objectWithTools));
    }
    default int estimateTokenCountInTools(Iterable<Object> objectsWithTools) {
        List<ToolSpecification> toolSpecifications = new ArrayList<>();
        objectsWithTools.forEach(objectWithTools ->

toolSpecifications.addAll(toolSpecificationsFrom(objectWithTools)));
        return estimateTokenCountInToolSpecifications(toolSpecifications);
    }
    default int estimateTokenCountInToolSpecification(ToolSpecification
toolSpecification) {
        return
estimateTokenCountInToolSpecifications(singletonList(toolSpecification));
    }
    int estimateTokenCountInToolSpecifications(Iterable<ToolSpecification>
toolSpecifications);
}
```

langchain4j-core\src\main\java\dev\langchain4j\retriever\Retriever.java

```
package dev.langchain4j.retriever;
import java.util.List;
public interface Retriever<T> {
    List<T> findRelevant(String text);
}
```

langchain4j-core\src\main\java\dev\langchain4j\spi\prompt\PromptTemplateFactory.java

```
package dev.langchain4j.spi.prompt;
import java.util.Map;
public interface PromptTemplateFactory {
    Template create(Input input);
    interface Input {
        String getTemplate();
        String getName();
    }
    interface Template {
        String render(Map<String, Object> vars);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\spi\prompt\structured\StructuredPromptFactory.java

```
package dev.langchain4j.spi.prompt.structured;
import dev.langchain4j.model.input.Prompt;
public interface StructuredPromptFactory {
    Prompt toPrompt(Object structuredPrompt);
}
```

langchain4j-core\src\main\java\dev\langchain4j\spi\ServiceHelper.java

```
package dev.langchain4j.spi;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.ServiceLoader;
public class ServiceHelper {
  public static <T> Collection<T> loadFactories(Class<T> clazz) {
    return loadFactories(clazz, null);
  }
  public static <T> Collection<T> loadFactories(Class<T> clazz, ClassLoader
classLoader) {
    List<T> list = new ArrayList<>();
    ServiceLoader<T> factories;
    if (classLoader != null) {
      factories = ServiceLoader.load(clazz, classLoader);
    } else {
      // this is equivalent to:
      // ServiceLoader.load(clazz, TCCL);
      factories = ServiceLoader.load(clazz);
    }
    if (factories.iterator().hasNext()) {
      factories.iterator().forEachRemaining(list::add);
      return list;
    } else {
      // By default ServiceLoader.load uses the TCCL, this may not be enough
in environment dealing with
      // classloaders differently such as OSGi. So we should try to use the
classloader having loaded this
      // class. In OSGi it would be the bundle exposing vert.x and so have
access to all its classes.
      factories = ServiceLoader.load(clazz,
ServiceHelper.class.getClassLoader());
      if (factories.iterator().hasNext()) {
        factories.iterator().forEachRemaining(list::add);
        return list;
      } else {
        return Collections.emptyList();
      }
    }
  }
}
```

langchain4j-core\src\main\java\dev\langchain4j\store\embedding\CosineSimilarity.java

```
package dev.langchain4j.store.embedding;
import dev.langchain4j.data.embedding.Embedding;
import static dev.langchain4j.internal.Exceptions.illegalArgument;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
public class CosineSimilarity {
    /**
     * Calculates cosine similarity between two vectors.
     * <p>
     * Cosine similarity measures the cosine of the angle between two
vectors, indicating their directional similarity.
     * It produces a value in the range:
     * <p>
     * -1 indicates vectors are diametrically opposed (opposite directions).
     * <p>
     * 0 indicates vectors are orthogonal (no directional similarity).
     * <p>
     * 1 indicates vectors are pointing in the same direction (but not
necessarily of the same magnitude).
     * <p>
     * Not to be confused with cosine distance ([0..2]), which quantifies how
different two vectors are.
     *
     * @param embeddingA first embedding vector
     * @param embeddingB second embedding vector
     * @return cosine similarity in the range [-1..1]
     */
    public static double between(Embedding embeddingA, Embedding embeddingB) {
        ensureNotNull(embeddingA, "embeddingA");
        ensureNotNull(embeddingB, "embeddingB");
        float[] vectorA = embeddingA.vector();
        float[] vectorB = embeddingB.vector();
        if (vectorA.length != vectorB.length) {
            throw illegalArgument("Length of vector a (%s) must be equal to
the length of vector b (%s)",
                    vectorA.length, vectorB.length);
        }
        double dotProduct = 0.0;
        double normA = 0.0;
        double normB = 0.0;
        for (int i = 0; i < vectorA.length; i++) {
            dotProduct += vectorA[i] * vectorB[i];
            normA += vectorA[i] * vectorA[i];
            normB += vectorB[i] * vectorB[i];
        }
        return dotProduct / (Math.sqrt(normA) * Math.sqrt(normB));
    }
    /**
     * Converts relevance score into cosine similarity.
     *
     * @param relevanceScore Relevance score in the range [0..1] where 0 is
not relevant and 1 is relevant.
     * @return Cosine similarity in the range [-1..1] where -1 is not
relevant and 1 is relevant.
     */
    public static double fromRelevanceScore(double relevanceScore) {
        return relevanceScore * 2 - 1;
    }
```

```
}
```

langchain4j-core\src\main\java\dev\langchain4j\store\embedding\EmbeddingMatch.java

```
package dev.langchain4j.store.embedding;
import dev.langchain4j.data.embedding.Embedding;
import java.util.Objects;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
/**
 * Represents a matched embedding along with its relevance score (derivative
of cosine distance), ID, and original embedded content.
 *
 * @param <Embedded> The class of the object that has been embedded.
Typically, it is {@link dev.langchain4j.data.segment.TextSegment}.
 */
public class EmbeddingMatch<Embedded> {
    private final Double score;
    private final String embeddingId;
    private final Embedding embedding;
    private final Embedded embedded;
    public EmbeddingMatch(Double score, String embeddingId, Embedding
embedding, Embedded embedded) {
        this.score = ensureNotNull(score, "score");
        this.embeddingId = ensureNotBlank(embeddingId, "embeddingId");
        this.embedding = embedding;
        this.embedded = embedded;
    }
    /**
     * Returns the relevance score (derivative of cosine distance) of this
embedding compared to
     * a reference embedding during a search.
     * The current implementation assumes that the embedding store uses
cosine distance when comparing embeddings.
     *
     * @return Relevance score, ranging from 0 (not relevant) to 1 (highly
relevant).
     */
    public Double score() {
        return score;
    }
    /**
     * @return The ID of the embedding assigned when adding this embedding to
the store.
     */
    public String embeddingId() {
        return embeddingId;
    }
    /**
     * @return The embedding that has been matched.
     */
    public Embedding embedding() {
        return embedding;
    }
    /**
     * @return The original content that was embedded. Typically, this is a
{@link dev.langchain4j.data.segment.TextSegment}.
     */
    public Embedded embedded() {
        return embedded;
    }
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        EmbeddingMatch<?> that = (EmbeddingMatch<?>) o;
        return Objects.equals(this.score, that.score)
                && Objects.equals(this.embeddingId, that.embeddingId)
                && Objects.equals(this.embedding, that.embedding)
                && Objects.equals(this.embedded, that.embedded);
    }
    @Override
    public int hashCode() {
        return Objects.hash(score, embeddingId, embedding, embedded);
    }
    @Override
    public String toString() {
        return "EmbeddingMatch {" +
                " score = " + score +
                ", embedded = " + embedded +
                ", embeddingId = " + embeddingId +
                ", embedding = " + embedding +
                " }";
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\store\embedding\EmbeddingStore.java

```
package dev.langchain4j.store.embedding;
import dev.langchain4j.data.embedding.Embedding;
import java.util.List;
/**
 * Represents a store for embeddings, also known as a vector database.
 *
 * @param <Embedded> The class of the object that has been embedded.
Typically, this is {@link dev.langchain4j.data.segment.TextSegment}.
 */
public interface EmbeddingStore<Embedded> {
    /**
     * Adds a given embedding to the store.
     *
     * @param embedding The embedding to be added to the store.
     * @return The auto-generated ID associated with the added embedding.
     */
    String add(Embedding embedding);
    /**
     * Adds a given embedding to the store.
     *
     * @param id        The unique identifier for the embedding to be added.
     * @param embedding The embedding to be added to the store.
     */
    void add(String id, Embedding embedding);
    /**
     * Adds a given embedding and the corresponding content that has been
embedded to the store.
     *
     * @param embedding The embedding to be added to the store.
     * @param embedded  Original content that was embedded.
     * @return The auto-generated ID associated with the added embedding.
     */
    String add(Embedding embedding, Embedded embedded);
    /**
     * Adds multiple embeddings to the store.
     *
     * @param embeddings A list of embeddings to be added to the store.
     * @return A list of auto-generated IDs associated with the added
embeddings.
     */
    List<String> addAll(List<Embedding> embeddings);
    /**
     * Adds multiple embeddings and their corresponding contents that have
been embedded to the store.
     *
     * @param embeddings A list of embeddings to be added to the store.
     * @param embedded   A list of original contents that were embedded.
     * @return A list of auto-generated IDs associated with the added
embeddings.
     */
    List<String> addAll(List<Embedding> embeddings, List<Embedded> embedded);
    /**
     * Finds the most relevant (closest in space) embeddings to the provided
reference embedding.
     * By default, minScore is set to 0, which means that the results may
include embeddings with low relevance.
     *
```

```
     * @param referenceEmbedding The embedding used as a reference. Returned
embeddings should be relevant (closest) to this one.
     * @param maxResults         The maximum number of embeddings to be
returned.
     * @return A list of embedding matches.
     * Each embedding match includes a relevance score (derivative of cosine
distance),
     * ranging from 0 (not relevant) to 1 (highly relevant).
     */
    default List<EmbeddingMatch<Embedded>> findRelevant(Embedding
referenceEmbedding, int maxResults) {
        return findRelevant(referenceEmbedding, maxResults, 0);
    }
    /**
     * Finds the most relevant (closest in space) embeddings to the provided
reference embedding.
     *
     * @param referenceEmbedding The embedding used as a reference. Returned
embeddings should be relevant (closest) to this one.
     * @param maxResults         The maximum number of embeddings to be
returned.
     * @param minScore           The minimum relevance score, ranging from 0
to 1 (inclusive).
     *                           Only embeddings with a score of this value
or higher will be returned.
     * @return A list of embedding matches.
     * Each embedding match includes a relevance score (derivative of cosine
distance),
     * ranging from 0 (not relevant) to 1 (highly relevant).
     */
    List<EmbeddingMatch<Embedded>> findRelevant(Embedding referenceEmbedding,
int maxResults, double minScore);
}
```

langchain4j-core\src\main\java\dev\langchain4j\store\embedding\RelevanceScore.java

```
package dev.langchain4j.store.embedding;
public class RelevanceScore {
    /**
     * Converts cosine similarity into relevance score.
     *
     * @param cosineSimilarity Cosine similarity in the range [-1..1] where
-1 is not relevant and 1 is relevant.
     * @return Relevance score in the range [0..1] where 0 is not relevant
and 1 is relevant.
     */
    public static double fromCosineSimilarity(double cosineSimilarity) {
        return (cosineSimilarity + 1) / 2;
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\store\memory\chat\ChatMemoryStore.java

```
package dev.langchain4j.store.memory.chat;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.ChatMessageDeserializer;
import dev.langchain4j.data.message.ChatMessageSerializer;
import dev.langchain4j.memory.ChatMemory;
import java.util.List;
/**
 * Represents a store for the {@link ChatMemory} state.
 * Allows for flexibility in terms of where and how chat memory is stored.
 * Currently, the only implementation available is {@link
InMemoryChatMemoryStore}. We are in the process of adding
 * ready implementations for popular stores like SQL DBs, document stores,
etc.
 * In the meantime, you can implement this interface to connect to any
storage of your choice.
 */
public interface ChatMemoryStore {
    /**
     * Retrieves messages for a specified chat memory.
     *
     * @param memoryId The ID of the chat memory.
     * @return List of messages for the specified chat memory. Must not be
null. Can be deserialized from JSON using {@link ChatMessageDeserializer}.
     */
    List<ChatMessage> getMessages(Object memoryId);
    /**
     * Updates messages for a specified chat memory.
     *
     * @param memoryId The ID of the chat memory.
     * @param messages List of messages for the specified chat memory, that
represent the current state of the {@link ChatMemory}.
     *                 Can be serialized to JSON using {@link
ChatMessageSerializer}.
     */
    void updateMessages(Object memoryId, List<ChatMessage> messages);
    /**
     * Deletes all messages for a specified chat memory.
     *
     * @param memoryId The ID of the chat memory.
     */
    void deleteMessages(Object memoryId);
}
```

langchain4j-core\src\main\java\dev\langchain4j\store\memory\chat\InMemoryChatMemoryStore.java

```
package dev.langchain4j.store.memory.chat;
import dev.langchain4j.data.message.ChatMessage;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
/**
 * Implementation of {@link ChatMemoryStore} that stores state of {@link dev.langchain4j.memory.ChatMemory} (chat messages) in-memory.
 * <p>
 * This storage mechanism is transient and does not persist data across application restarts.
 */
public class InMemoryChatMemoryStore implements ChatMemoryStore {
    private final Map<Object, List<ChatMessage>> messagesByMemoryId = new ConcurrentHashMap<>();
    @Override
    public List<ChatMessage> getMessages(Object memoryId) {
        return messagesByMemoryId.computeIfAbsent(memoryId, ignored -> new ArrayList<>());
    }
    @Override
    public void updateMessages(Object memoryId, List<ChatMessage> messages) {
        messagesByMemoryId.put(memoryId, messages);
    }
    @Override
    public void deleteMessages(Object memoryId) {
        messagesByMemoryId.remove(memoryId);
    }
}
```

langchain4j-core\src\main\java\dev\langchain4j\WillChangeSoon.java

```
package dev.langchain4j;
import java.lang.annotation.Target;
import static java.lang.annotation.ElementType.*;
/**
 * Indicates that a class/constructor/method is planned to change soon.
 */
@Target({TYPE, CONSTRUCTOR, METHOD})
public @interface WillChangeSoon {
    String value();
}
```

langchain4j-core\src\test\java\dev\langchain4j\data\message\ChatMessageSerializerTest.java

```
package dev.langchain4j.data.message;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static
dev.langchain4j.data.message.ChatMessageDeserializer.messageFromJson;
import static
dev.langchain4j.data.message.ChatMessageDeserializer.messagesFromJson;
import static dev.langchain4j.data.message.SystemMessage.systemMessage;
import static dev.langchain4j.data.message.ToolExecutionResultMessage.toolExec
utionResultMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static java.util.Arrays.asList;
import static java.util.Collections.emptyList;
import static java.util.Collections.singletonList;
import static org.assertj.core.api.Assertions.assertThat;
class ChatMessageSerializerTest {
    @Test
    void should_serialize_and_deserialize_chat_message() {
        UserMessage message = userMessage("hello");
        String json = ChatMessageSerializer.messageToJson(message);
        ChatMessage deserializedMessage = messageFromJson(json);
        assertThat(deserializedMessage).isEqualTo(message);
    }
    @Test
    void should_serialize_and_deserialize_empty_list() {
        List<ChatMessage> messages = emptyList();
        String json = ChatMessageSerializer.messagesToJson(messages);
        List<ChatMessage> deserializedMessages = messagesFromJson(json);
        assertThat(deserializedMessages).isEmpty();
    }
    @Test
    void should_deserialize_null_as_empty_list() {
        assertThat(messagesFromJson(null)).isEmpty();
    }
    @Test
    void should_serialize_and_deserialize_list_with_one_message() {
        List<ChatMessage> messages = singletonList(userMessage("hello"));
        String json = ChatMessageSerializer.messagesToJson(messages);
        assertThat(json).isEqualTo("[{\"text\":\"hello\",\"type\":
\"USER\"}]");
        List<ChatMessage> deserializedMessages = messagesFromJson(json);
        assertThat(deserializedMessages).isEqualTo(messages);
    }
    @Test
    void should_serialize_and_deserialize_list_with_all_types_of_messages() {
        List<ChatMessage> messages = asList(
                systemMessage("Hello from system"),
                userMessage("Hello from user"),
                userMessage("Klaus", "Hello from Klaus"),
                aiMessage("Hello from AI"),
                aiMessage(ToolExecutionRequest.builder()
                        .name("calculator")
                        .arguments("{}")
                        .build()),
                toolExecutionResultMessage("calculator", "4")
```

```
        );
        String json = ChatMessageSerializer.messagesToJson(messages);
        assertThat(json).isEqualTo("[" +
                "{\"text\":\"Hello from system\",\"type\":\"SYSTEM\"}," +
                "{\"text\":\"Hello from user\",\"type\":\"USER\"}," +
                "{\"name\":\"Klaus\",\"text\":\"Hello from Klaus\",\"type\":
\"USER\"}," +
                "{\"text\":\"Hello from AI\",\"type\":\"AI\"}," +
                "{\"toolExecutionRequest\":{\"name\":\"calculator\",
\"arguments\":\"{}\"},\"type\":\"AI\"}," +
                "{\"toolName\":\"calculator\",\"text\":\"4\",\"type\":
\"TOOL_EXECUTION_RESULT\"}" +
                "]");
        List<ChatMessage> deserializedMessages = messagesFromJson(json);
        assertThat(deserializedMessages).isEqualTo(messages);
    }
}
```

langchain4j-core\src\test\java\dev\langchain4j\internal\JsonTest.java

```
package dev.langchain4j.internal;
import static org.assertj.core.api.Assertions.assertThat;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import com.google.gson.annotations.SerializedName;
import org.junit.jupiter.api.Test;
class JsonTest {
  @Test
  void conversionToJsonAndFromJsonWorks() {
    TestData testData = new TestData();
    testData.setSampleDate(LocalDate.of(2023, 1, 15));
    testData.setSampleDateTime(LocalDateTime.of(2023, 1, 15, 10, 20));
    testData.setSomeValue("value");
    String json = Json.toJson(testData);
    assertThat(json)
      .isEqualTo(
        "{\n" +
        "  \"sampleDate\": \"2023-01-15\",\n" +
        "  \"sampleDateTime\": \"2023-01-15T10:20:00\",\n" +
        "  \"some_value\": \"value\"\n" +
        "}"
      );
    TestData deserializedData = Json.fromJson(json, TestData.class);
    assertThat(deserializedData.getSampleDate()).isEqualTo(testData.getSampleD
ate());
    assertThat(deserializedData.getSampleDateTime()).isEqualTo(testData.getSam
pleDateTime());
    assertThat(deserializedData.getSomeValue()).isEqualTo(testData.getSomeValu
e());
  }
  @Test
  void toInputStreamWorksForList() throws IOException {
    List<TestObject> testObjects = Arrays.asList(
            new TestObject("John", LocalDate.of(2021, 8, 17),
LocalDateTime.of(2021, 8, 17, 14, 20)),
            new TestObject("Jane", LocalDate.of(2021, 8, 16),
LocalDateTime.of(2021, 8, 16, 13, 19))
    );
    String expectedJson = "[{" +
            "\"name\":\"John\"," +
            "\"date\":\"2021-08-17\"," +
            "\"dateTime\":\"2021-08-17T14:20:00\"" +
            "},{" +
            "\"name\":\"Jane\"," +
            "\"date\":\"2021-08-16\"," +
            "\"dateTime\":\"2021-08-16T13:19:00\"" +
            "}]";
    InputStream inputStream = Json.toInputStream(testObjects, List.class);
    try (BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream))) {
      String resultJson =
```

```
bufferedReader.lines().collect(Collectors.joining());
      assertThat(resultJson).isEqualTo(expectedJson);
    }
  }
  private static class TestObject {
    private final String name;
    private final LocalDate date;
    private final LocalDateTime dateTime;
    public TestObject(String name, LocalDate date, LocalDateTime dateTime) {
      this.name = name;
      this.date = date;
      this.dateTime = dateTime;
    }
  }
  private static class TestData {
    private LocalDate sampleDate;
    private LocalDateTime sampleDateTime;
    @SerializedName("some_value")
    private String someValue;
    LocalDate getSampleDate() {
      return sampleDate;
    }
    void setSampleDate(LocalDate sampleDate) {
      this.sampleDate = sampleDate;
    }
    LocalDateTime getSampleDateTime() {
      return sampleDateTime;
    }
    void setSampleDateTime(LocalDateTime sampleDateTime) {
      this.sampleDateTime = sampleDateTime;
    }
    String getSomeValue() {
      return someValue;
    }
    void setSomeValue(String someValue) {
      this.someValue = someValue;
    }
  }
}
```

langchain4j-core\src\test\java\dev\langchain4j\internal\UtilsTest.java

```
package dev.langchain4j.internal;
import static dev.langchain4j.internal.Utils.quoted;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatExceptionOfType;
import java.util.UUID;
import java.util.stream.Stream;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
class UtilsTest {
  @Test
  void randomUUIDWorks() {
    String uuid1 = Utils.randomUUID();
    String uuid2 = Utils.randomUUID();
    assertThat(uuid1).isNotNull().isNotEmpty();
    assertThat(uuid2).isNotNull().isNotEmpty();
    // Checking if the two generated UUIDs are not the same
    assertThat(uuid1).isNotEqualTo(uuid2);
    // Validate if the returned string is in the UUID format
    UUID.fromString(uuid1);
    UUID.fromString(uuid2);
  }
  @Test
  void generateUUIDFromTextWorks() {
    String input1 = "Hello";
    String input2 = "World";
    String uuidFromInput1 = Utils.generateUUIDFrom(input1);
    String uuidFromInput2 = Utils.generateUUIDFrom(input2);
    assertThat(uuidFromInput1).isNotNull().isNotEmpty();
    assertThat(uuidFromInput2).isNotNull().isNotEmpty();
    // Different inputs should produce different UUIDs
    assertThat(uuidFromInput1).isNotEqualTo(uuidFromInput2);
    // Validate if the returned string is in the UUID format
    UUID.fromString(uuidFromInput1);
    UUID.fromString(uuidFromInput2);
    // Test if hashing is consistent for the same input
    assertThat(Utils.generateUUIDFrom(input1)).isEqualTo(uuidFromInput1);
  }
  @Test
  void generateUUIDFromEmptyInputWorks() {
    String uuidFromEmptyInput = Utils.generateUUIDFrom("");
    assertThat(uuidFromEmptyInput).isNotNull().isNotEmpty();
    // Validate if the returned string is in the UUID format
    UUID.fromString(uuidFromEmptyInput);
  }
  @Test
  void generateUUIDFromNullInputWorks() {
    assertThatExceptionOfType(NullPointerException.class).isThrownBy(() ->
Utils.generateUUIDFrom(null));
  }
  @MethodSource
  @ParameterizedTest
  void test_quoted(String string, String expected) {
    assertThat(quoted(string)).isEqualTo(expected);
  }
  static Stream<Arguments> test_quoted() {
    return Stream.of(
```

```
      Arguments.of(null, "null"),
      Arguments.of("", "\"\""),
      Arguments.of(" ", "\" \""),
      Arguments.of("hello", "\"hello\"")
    );
  }
}
```

langchain4j-core\src\test\java\dev\langchain4j\internal\ValidationUtilsTest.java

```
package dev.langchain4j.internal;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.NullSource;
import org.junit.jupiter.params.provider.ValueSource;
import static dev.langchain4j.internal.ValidationUtils.ensureBetween;
import static dev.langchain4j.internal.ValidationUtils.ensureGreaterThanZero;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
class ValidationUtilsTest {
    @ParameterizedTest
    @ValueSource(ints = {1, Integer.MAX_VALUE})
    void should_not_throw_when_greater_than_0(Integer i) {
        ensureGreaterThanZero(i, "integer");
    }
    @ParameterizedTest
    @NullSource
    @ValueSource(ints = {Integer.MIN_VALUE, 0})
    void should_throw_when_when_not_greater_than_0(Integer i) {
        assertThatThrownBy(() -> ensureGreaterThanZero(i, "integer"))
                .isExactlyInstanceOf(IllegalArgumentException.class)
                .hasMessage("integer must be greater than zero, but is: " +
i);
    }
    @ParameterizedTest
    @ValueSource(doubles = {0.0, 0.5, 1.0})
    void should_not_throw_when_between(Double d) {
        ensureBetween(d, 0.0, 1.0, "test");
    }
    @ParameterizedTest
    @NullSource
    @ValueSource(doubles = {-0.1, 1.1})
    void should_throw_when_not_between(Double d) {
        assertThatThrownBy(() -> ensureBetween(d, 0.0, 1.0, "test"))
                .isExactlyInstanceOf(IllegalArgumentException.class)
                .hasMessage("test must be between 0.0 and 1.0, but is: " + d);
    }
}
```

langchain4j-core\src\test\java\dev\langchain4j\model\input\PromptTemplateTest.java

```
package dev.langchain4j.model.input;
import org.junit.jupiter.api.Test;
import java.time.*;
import java.util.HashMap;
import java.util.Map;
import static org.assertj.core.api.Assertions.assertThat;
class PromptTemplateTest {
    @Test
    void should_create_prompt_from_template_with_single_variable() {
        PromptTemplate promptTemplate = PromptTemplate.from("My name is
{{it}}.");
        Prompt prompt = promptTemplate.apply("Klaus");
        assertThat(prompt.text()).isEqualTo("My name is Klaus.");
    }
    @Test
    void should_create_prompt_from_template_with_multiple_variables() {
        PromptTemplate promptTemplate = PromptTemplate.from("My name is
{{name}} {{surname}}.");
        Map<String, Object> variables = new HashMap<>();
        variables.put("name", "Klaus");
        variables.put("surname", "Heißler");
        Prompt prompt = promptTemplate.apply(variables);
        assertThat(prompt.text()).isEqualTo("My name is Klaus Heißler.");
    }
    @Test
    void should_provide_date_automatically() {
        PromptTemplate promptTemplate = PromptTemplate.from("My name is
{{it}} and today is {{current_date}}");
        Prompt prompt = promptTemplate.apply("Klaus");
        assertThat(prompt.text()).isEqualTo("My name is Klaus and today is "
+ LocalDate.now());
    }
    @Test
    void should_provide_time_automatically() {
        Clock clock = Clock.fixed(Instant.now(), ZoneOffset.UTC);
        PromptTemplate promptTemplate = new PromptTemplate("My name is {{it}}
and now is {{current_time}}", clock);
        Prompt prompt = promptTemplate.apply("Klaus");
        assertThat(prompt.text()).isEqualTo("My name is Klaus and now is " +
LocalTime.now(clock));
    }
    @Test
    void should_provide_date_and_time_automatically() {
        Clock clock = Clock.fixed(Instant.now(), ZoneOffset.UTC);
        PromptTemplate promptTemplate = new PromptTemplate("My name is {{it}}
and now is {{current_date_time}}", clock);
        Prompt prompt = promptTemplate.apply("Klaus");
        assertThat(prompt.text()).isEqualTo("My name is Klaus and now is " +
LocalDateTime.now(clock));
    }
}
```

langchain4j-core\src\test\java\dev\langchain4j\model\input\structured\StructuredPromptProcessorTest.java

```
package dev.langchain4j.model.input.structured;
import static
dev.langchain4j.model.input.structured.StructuredPromptProcessor.toPrompt;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import dev.langchain4j.model.input.Prompt;
import java.util.List;
import org.junit.jupiter.api.Test;
class StructuredPromptProcessorTest {
  @StructuredPrompt("Hello, my name is {{name}}")
  static class Greeting {
    private String name;
  }
  @Test
  void test_prompt_with_single_variable() {
    Greeting structuredPrompt = new Greeting();
    structuredPrompt.name = "Klaus";
    Prompt prompt = toPrompt(structuredPrompt);
    assertThat(prompt.text()).isEqualTo("Hello, my name is Klaus");
  }
  @StructuredPrompt(
    {
      "Suggest tasty {{dish}} recipes that can be prepared in
{{maxPreparationTime}} minutes.",
      "I have only {{ingredients}} in my fridge.",
    }
  )
  static class SuggestRecipes {
    private String dish;
    private int maxPreparationTime;
    private List<String> ingredients;
  }
  @Test
  void test_prompt_with_multiple_variables() {
    SuggestRecipes structuredPrompt = new SuggestRecipes();
    structuredPrompt.dish = "salad";
    structuredPrompt.maxPreparationTime = 5;
    structuredPrompt.ingredients = asList("Tomato", "Cucumber", "Onion");
    Prompt prompt = toPrompt(structuredPrompt);
    assertThat(prompt.text())
      .isEqualTo(
        "Suggest tasty salad recipes that can be prepared in 5 minutes.\nI
have only [Tomato, Cucumber, Onion] in my fridge."
      );
  }
  @StructuredPrompt(
    "Example of numbers with floating point: {{nDouble}}, {{nFloat}} and
whole numbers: {{nInt}}, {{nShort}}, {{nLong}}"
  )
  static class VariousNumbers {
    private double nDouble;
    private float nFloat;
    private int nInt;
    private short nShort;
    private long nLong;
  }
  @Test
```

```
  void test_prompt_with_various_number_types() {
    VariousNumbers numbers = new VariousNumbers();
    numbers.nDouble = 17.15;
    numbers.nFloat = 1;
    numbers.nInt = 2;
    numbers.nShort = 10;
    numbers.nLong = 12;
    Prompt prompt = toPrompt(numbers);
    assertThat(prompt.text())
      .isEqualTo("Example of numbers with floating point: 17.15, 1.0 and
whole numbers: 2, 10, 12");
  }
}
```

langchain4j-core\src\test\java\dev\langchain4j\store\embedding\CosineSimilarityTest.java

```
package dev.langchain4j.store.embedding;
import dev.langchain4j.data.embedding.Embedding;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
class CosineSimilarityTest {
    @Test
    void should_calculate_cosine_similarity() {
        Embedding embeddingA = Embedding.from(new float[]{1, 1, 1});
        Embedding embeddingB = Embedding.from(new float[]{-1, -1, -1});
        assertThat(CosineSimilarity.between(embeddingA,
embeddingA)).isCloseTo(1, withPercentage(1));
        assertThat(CosineSimilarity.between(embeddingA,
embeddingB)).isCloseTo(-1, withPercentage(1));
    }
    @Test
    void should_convert_relevance_score_into_cosine_similarity() {
        assertThat(CosineSimilarity.fromRelevanceScore(0)).isEqualTo(-1);
        assertThat(CosineSimilarity.fromRelevanceScore(0.5)).isEqualTo(0);
        assertThat(CosineSimilarity.fromRelevanceScore(1)).isEqualTo(1);
    }
}
```

langchain4j-core\src\test\java\dev\langchain4j\store\embedding\RelevanceScoreTest.java

```
package dev.langchain4j.store.embedding;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class RelevanceScoreTest {
    @Test
    void should_convert_cosine_similarity_into_relevance_score() {
        assertThat(RelevanceScore.fromCosineSimilarity(-1)).isEqualTo(0);
        assertThat(RelevanceScore.fromCosineSimilarity(0)).isEqualTo(0.5);
        assertThat(RelevanceScore.fromCosineSimilarity(1)).isEqualTo(1);
    }
}
```

langchain4j-dashscope\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-dashscope</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with DashScope</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dashscope-sdk-java</artifactId>
            <version>2.5.0</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-dashscope\src\main\java\dev\langchain4j\model\dashscope\QwenChatModel.java

```
package dev.langchain4j.model.dashscope;
import com.alibaba.dashscope.aigc.generation.Generation;
import com.alibaba.dashscope.aigc.generation.GenerationResult;
import com.alibaba.dashscope.aigc.generation.models.QwenParam;
import com.alibaba.dashscope.exception.InputRequiredException;
import com.alibaba.dashscope.exception.NoApiKeyException;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.internal.Utils;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.output.Response;
import java.util.List;
import static com.alibaba.dashscope.aigc.generation.models.QwenParam.ResultFormat.MESSAGE;
import static dev.langchain4j.model.dashscope.QwenHelper.*;
public class QwenChatModel implements ChatLanguageModel {
    protected final String apiKey;
    protected final String modelName;
    protected final Double topP;
    protected final Integer topK;
    protected final Boolean enableSearch;
    protected final Integer seed;
    protected final Generation generation;
    protected QwenChatModel(String apiKey,
                            String modelName,
                            Double topP,
                            Integer topK,
                            Boolean enableSearch,
                            Integer seed) {
        this.apiKey = apiKey;
        this.modelName = modelName;
        this.topP = topP;
        this.topK = topK;
        this.enableSearch = enableSearch;
        this.seed = seed;
        this.generation = new Generation();
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages) {
        try {
            QwenParam param = QwenParam.builder()
                    .apiKey(apiKey)
                    .model(modelName)
                    .topP(topP)
                    .topK(topK)
                    .enableSearch(enableSearch)
                    .seed(seed)
                    .messages(toQwenMessages(messages))
                    .resultFormat(MESSAGE)
                    .build();
            GenerationResult generationResult = generation.call(param);
            String answer = answerFrom(generationResult);
            return Response.from(AiMessage.from(answer),
                    tokenUsageFrom(generationResult),
finishReasonFrom(generationResult));
        } catch (NoApiKeyException | InputRequiredException e) {
```

```
            throw new RuntimeException(e);
        }
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
List<ToolSpecification> toolSpecifications) {
        throw new IllegalArgumentException("Tools are currently not supported
for qwen models");
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
ToolSpecification toolSpecification) {
        throw new IllegalArgumentException("Tools are currently not supported
for qwen models");
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        protected String apiKey;
        protected String modelName;
        protected Double topP;
        protected Integer topK;
        protected Boolean enableSearch;
        protected Integer seed;
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        public Builder modelName(String modelName) {
            this.modelName = modelName;
            return this;
        }
        public Builder topP(Double topP) {
            this.topP = topP;
            return this;
        }
        public Builder topK(Integer topK) {
            this.topK = topK;
            return this;
        }
        public Builder enableSearch(Boolean enableSearch) {
            this.enableSearch = enableSearch;
            return this;
        }
        public Builder seed(Integer seed) {
            this.seed = seed;
            return this;
        }
        protected void ensureOptions() {
            if (Utils.isNullOrBlank(apiKey)) {
                throw new IllegalArgumentException("DashScope api key must be
defined. It can be generated here: https://dashscope.console.aliyun.com/
apiKey");
            }
            modelName = Utils.isNullOrBlank(modelName) ?
QwenModelName.QWEN_PLUS : modelName;
            enableSearch = enableSearch != null && enableSearch;
        }
        public QwenChatModel build() {
            ensureOptions();
```

```
            return new QwenChatModel(apiKey, modelName, topP, topK,
enableSearch, seed);
        }
    }
}
```

langchain4j-dashscope\src\main\java\dev\langchain4j\model\dashscope\QwenEmbeddingModel.java

```
package dev.langchain4j.model.dashscope;
import com.alibaba.dashscope.embeddings.*;
import com.alibaba.dashscope.exception.NoApiKeyException;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.internal.Utils;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import static
com.alibaba.dashscope.embeddings.TextEmbeddingParam.TextType.DOCUMENT;
import static
com.alibaba.dashscope.embeddings.TextEmbeddingParam.TextType.QUERY;
import static java.util.Collections.singletonList;
public class QwenEmbeddingModel implements EmbeddingModel {
    public static final String TYPE_KEY = "type";
    public static final String TYPE_QUERY = "query";
    public static final String TYPE_DOCUMENT = "document";
    private final String apiKey;
    private final String modelName;
    private final TextEmbedding embedding;
    public QwenEmbeddingModel(String apiKey, String modelName) {
        this.apiKey = apiKey;
        this.modelName = modelName;
        this.embedding = new TextEmbedding();
    }
    private boolean containsDocuments(List<TextSegment> textSegments) {
        return textSegments.stream()
                .map(TextSegment::metadata)
                .map(metadata -> metadata.get(TYPE_KEY))
                .filter(TYPE_DOCUMENT::equalsIgnoreCase)
                .anyMatch(Utils::isNullOrBlank);
    }
    private boolean containsQueries(List<TextSegment> textSegments) {
        return textSegments.stream()
                .map(TextSegment::metadata)
                .map(metadata -> metadata.get(TYPE_KEY))
                .filter(TYPE_QUERY::equalsIgnoreCase)
                .anyMatch(Utils::isNullOrBlank);
    }
    private Response<List<Embedding>> embedTexts(List<TextSegment>
textSegments, TextEmbeddingParam.TextType textType) {
        TextEmbeddingParam param = TextEmbeddingParam.builder()
                .apiKey(apiKey)
                .model(modelName)
                .textType(textType)
                .texts(textSegments.stream()
                        .map(TextSegment::text)
                        .collect(Collectors.toList()))
                .build();
        try {
            TextEmbeddingResult generationResult = embedding.call(param);
```

```
            // total_tokens are the same as input_tokens in the embedding
model
            TokenUsage usage = new
TokenUsage(generationResult.getUsage().getTotalTokens());
            List<Embedding> embeddings = Optional.of(generationResult)
                    .map(TextEmbeddingResult::getOutput)
                    .map(TextEmbeddingOutput::getEmbeddings)
                    .orElse(Collections.emptyList())
                    .stream()
                    .map(TextEmbeddingResultItem::getEmbedding)
                    .map(doubleList ->
doubleList.stream().map(Double::floatValue).collect(Collectors.toList()))
                    .map(Embedding::from)
                    .collect(Collectors.toList());
            return Response.from(embeddings, usage);
        } catch (NoApiKeyException e) {
            throw new RuntimeException(e);
        }
    }
    @Override
    public Response<List<Embedding>> embedAll(List<TextSegment> textSegments)
{
        boolean queries = containsQueries(textSegments);
        if (!queries) {
            // default all documents
            return embedTexts(textSegments, DOCUMENT);
        } else {
            boolean documents = containsDocuments(textSegments);
            if (!documents) {
                return embedTexts(textSegments, QUERY);
            } else {
                // This is a mixed collection of queries and documents. Embed
one by one.
                List<Embedding> embeddings = new
ArrayList<>(textSegments.size());
                Integer tokens = null;
                for (TextSegment textSegment : textSegments) {
                    Response<List<Embedding>> result;
                    if
(TYPE_QUERY.equalsIgnoreCase(textSegment.metadata(TYPE_KEY))) {
                        result = embedTexts(singletonList(textSegment),
QUERY);
                    } else {
                        result = embedTexts(singletonList(textSegment),
DOCUMENT);
                    }
                    embeddings.addAll(result.content());
                    if (result.tokenUsage() == null) {
                        continue;
                    }
                    if (tokens == null) {
                        tokens = result.tokenUsage().inputTokenCount();
                    } else {
                        tokens += result.tokenUsage().inputTokenCount();
                    }
                }
                return Response.from(embeddings, new TokenUsage(tokens));
            }
        }
    }
    public static Builder builder() {
```

```
        return new Builder();
    }
    public static class Builder {
        private String apiKey;
        private String modelName;
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        public Builder modelName(String modelName) {
            this.modelName = modelName;
            return this;
        }
        protected void ensureOptions() {
            if (Utils.isNullOrBlank(apiKey)) {
                throw new IllegalArgumentException("DashScope api key must be
defined. It can be generated here: https://dashscope.console.aliyun.com/
apiKey");
            }
            modelName = Utils.isNullOrBlank(modelName) ?
QwenModelName.TEXT_EMBEDDING_V1 : modelName;
        }
        public QwenEmbeddingModel build() {
            ensureOptions();
            return new QwenEmbeddingModel(apiKey, modelName);
        }
    }
}
```

langchain4j-dashscope\src\main\java\dev\langchain4j\model\dashscope\QwenHelper.java

```
package dev.langchain4j.model.dashscope;
import com.alibaba.dashscope.aigc.generation.GenerationOutput;
import com.alibaba.dashscope.aigc.generation.GenerationOutput.Choice;
import com.alibaba.dashscope.aigc.generation.GenerationResult;
import com.alibaba.dashscope.common.Message;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.model.output.FinishReason;
import dev.langchain4j.model.output.TokenUsage;
import java.util.List;
import java.util.Optional;
import static com.alibaba.dashscope.common.Role.*;
import static dev.langchain4j.model.output.FinishReason.LENGTH;
import static dev.langchain4j.model.output.FinishReason.STOP;
import static java.util.stream.Collectors.toList;
class QwenHelper {
    static List<Message> toQwenMessages(List<ChatMessage> messages) {
        return messages.stream()
                .map(QwenHelper::toQwenMessage)
                .collect(toList());
    }
    static Message toQwenMessage(ChatMessage message) {
        return Message.builder()
                .role(roleFrom(message))
                .content(message.text())
                .build();
    }
    static String roleFrom(ChatMessage message) {
        if (message instanceof AiMessage) {
            return ASSISTANT.getValue();
        } else if (message instanceof SystemMessage) {
            return SYSTEM.getValue();
        } else {
            return USER.getValue();
        }
    }
    static String answerFrom(GenerationResult result) {
        return Optional.of(result)
                .map(GenerationResult::getOutput)
                .map(GenerationOutput::getChoices)
                .filter(choices -> !choices.isEmpty())
                .map(choices -> choices.get(0))
                .map(Choice::getMessage)
                .map(Message::getContent)
                // Compatible with some older models.
                .orElseGet(() -> Optional.of(result)
                        .map(GenerationResult::getOutput)
                        .map(GenerationOutput::getText)
                        .orElse(""));
    }
    static TokenUsage tokenUsageFrom(GenerationResult result) {
        return Optional.of(result)
                .map(GenerationResult::getUsage)
                .map(usage -> new TokenUsage(usage.getInputTokens(),
usage.getOutputTokens()))
                .orElse(new TokenUsage(null, null));
```

```
    }
    static FinishReason finishReasonFrom(GenerationResult result) {
        String finishReason = Optional.of(result)
                .map(GenerationResult::getOutput)
                .map(GenerationOutput::getChoices)
                .filter(choices -> !choices.isEmpty())
                .map(choices -> choices.get(0))
                .map(Choice::getFinishReason)
                .orElse("");
        switch (finishReason) {
            case "stop":
                return STOP;
            case "length":
                return LENGTH;
            default:
                return null;
        }
    }
}
```

langchain4j-dashscope\src\main\java\dev\langchain4j\model\dashscope\QwenLanguageModel.java

```
package dev.langchain4j.model.dashscope;
import com.alibaba.dashscope.aigc.generation.Generation;
import com.alibaba.dashscope.aigc.generation.GenerationResult;
import com.alibaba.dashscope.aigc.generation.models.QwenParam;
import com.alibaba.dashscope.exception.InputRequiredException;
import com.alibaba.dashscope.exception.NoApiKeyException;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.output.Response;
import static
com.alibaba.dashscope.aigc.generation.models.QwenParam.ResultFormat.MESSAGE;
import static dev.langchain4j.internal.Utils.isNullOrBlank;
import static dev.langchain4j.model.dashscope.QwenHelper.*;
import static dev.langchain4j.model.dashscope.QwenModelName.QWEN_PLUS;
public class QwenLanguageModel implements LanguageModel {
    protected final String apiKey;
    protected final String modelName;
    protected final Double topP;
    protected final Integer topK;
    protected final Boolean enableSearch;
    protected final Integer seed;
    protected final Generation generation;
    public QwenLanguageModel(String apiKey,
                             String modelName,
                             Double topP,
                             Integer topK,
                             Boolean enableSearch,
                             Integer seed) {
        this.apiKey = apiKey;
        this.modelName = modelName;
        this.topP = topP;
        this.topK = topK;
        this.enableSearch = enableSearch;
        this.seed = seed;
        this.generation = new Generation();
    }
    @Override
    public Response<String> generate(String prompt) {
        try {
            QwenParam param = QwenParam.builder()
                    .apiKey(apiKey)
                    .model(modelName)
                    .topP(topP)
                    .topK(topK)
                    .enableSearch(enableSearch)
                    .seed(seed)
                    .prompt(prompt)
                    .resultFormat(MESSAGE)
                    .build();
            GenerationResult generationResult = generation.call(param);
            return Response.from(answerFrom(generationResult),
                    tokenUsageFrom(generationResult),
finishReasonFrom(generationResult));
        } catch (NoApiKeyException | InputRequiredException e) {
            throw new RuntimeException(e);
        }
    }
    public static Builder builder() {
```

```
        return new Builder();
    }
    public static class Builder {
        protected String apiKey;
        protected String modelName;
        protected Double topP;
        protected Integer topK;
        protected Boolean enableSearch;
        protected Integer seed;
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        public Builder modelName(String modelName) {
            this.modelName = modelName;
            return this;
        }
        public Builder topP(Double topP) {
            this.topP = topP;
            return this;
        }
        public Builder topK(Integer topK) {
            this.topK = topK;
            return this;
        }
        public Builder enableSearch(Boolean enableSearch) {
            this.enableSearch = enableSearch;
            return this;
        }
        public Builder seed(Integer seed) {
            this.seed = seed;
            return this;
        }
        protected void ensureOptions() {
            if (isNullOrBlank(apiKey)) {
                throw new IllegalArgumentException("DashScope api key must be
defined. It can be generated here: https://dashscope.console.aliyun.com/
apiKey");
            }
            modelName = isNullOrBlank(modelName) ? QWEN_PLUS : modelName;
            enableSearch = enableSearch != null && enableSearch;
        }
        public QwenLanguageModel build() {
            ensureOptions();
            return new QwenLanguageModel(apiKey, modelName, topP, topK,
enableSearch, seed);
        }
    }
}
```

langchain4j-dashscope\src\main\java\dev\langchain4j\model\dashscope\QwenModelName.java

```
package dev.langchain4j.model.dashscope;
/**
 * The LLMs provided by Alibaba Cloud, performs better than most LLMs in Asia languages.
 */
public class QwenModelName {
    // Use with QwenChatModel and QwenLanguageModel
    public static final String QWEN_TURBO = "qwen-turbo";  // Qwen base model, 4k context.
    public static final String QWEN_PLUS = "qwen-plus";  // Qwen plus model, 8k context.
    public static final String QWEN_SPARK_V1 = "qwen-spark-v1";  // Qwen sft for ai character scene, 4k context.
    public static final String QWEN_SPARK_V2 = "qwen-spark-v2";  // Qwen sft for ai character scene, 8k context.
    public static final String QWEN_7B_CHAT = "qwen-7b-chat";  // Qwen open sourced 7-billion-parameters version
    public static final String QWEN_14B_CHAT = "qwen-14b-chat";  // Qwen open sourced 14-billion-parameters version
    // Use with QwenEmbeddingModel
    public static final String TEXT_EMBEDDING_V1 = "text-embedding-v1";
}
```

langchain4j-dashscope\src\main\java\dev\langchain4j\model\dashscope\QwenStreamingChatModel.java

```
package dev.langchain4j.model.dashscope;
import com.alibaba.dashscope.aigc.generation.GenerationResult;
import com.alibaba.dashscope.aigc.generation.models.QwenParam;
import com.alibaba.dashscope.common.ResultCallback;
import com.alibaba.dashscope.exception.InputRequiredException;
import com.alibaba.dashscope.exception.NoApiKeyException;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import java.util.List;
import static
com.alibaba.dashscope.aigc.generation.models.QwenParam.ResultFormat.MESSAGE;
import static dev.langchain4j.model.dashscope.QwenHelper.toQwenMessages;
public class QwenStreamingChatModel extends QwenChatModel implements
StreamingChatLanguageModel {
    public QwenStreamingChatModel(String apiKey,
                                  String modelName,
                                  Double topP,
                                  Integer topK,
                                  Boolean enableSearch,
                                  Integer seed) {
        super(apiKey, modelName, topP, topK, enableSearch, seed);
    }
    @Override
    public void generate(List<ChatMessage> messages,
StreamingResponseHandler<AiMessage> handler) {
        try {
            QwenParam param = QwenParam.builder()
                    .apiKey(apiKey)
                    .model(modelName)
                    .topP(topP)
                    .topK(topK)
                    .enableSearch(enableSearch)
                    .seed(seed)
                    .messages(toQwenMessages(messages))
                    .resultFormat(MESSAGE)
                    .build();
            QwenStreamingResponseBuilder responseBuilder = new
QwenStreamingResponseBuilder();
            generation.streamCall(param, new
ResultCallback<GenerationResult>() {
                @Override
                public void onEvent(GenerationResult result) {
                    String delta = responseBuilder.append(result);
                    if (delta != null) {
                        handler.onNext(delta);
                    }
                }
                @Override
                public void onComplete() {
                    handler.onComplete(responseBuilder.build());
                }
                @Override
                public void onError(Exception e) {
                    handler.onError(e);
```

```
                }
            });
        } catch (NoApiKeyException | InputRequiredException e) {
            throw new RuntimeException(e);
        }
    }
    @Override
    public void generate(List<ChatMessage> messages,
                         List<ToolSpecification> toolSpecifications,
                         StreamingResponseHandler<AiMessage> handler) {
        throw new IllegalArgumentException("Tools are currently not supported
for qwen models");
    }
    @Override
    public void generate(List<ChatMessage> messages,
                         ToolSpecification toolSpecification,
                         StreamingResponseHandler<AiMessage> handler) {
        throw new IllegalArgumentException("Tools are currently not supported
for qwen models");
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder extends QwenChatModel.Builder {
        public Builder apiKey(String apiKey) {
            return (Builder) super.apiKey(apiKey);
        }
        public Builder modelName(String modelName) {
            return (Builder) super.modelName(modelName);
        }
        public Builder topP(Double topP) {
            return (Builder) super.topP(topP);
        }
        public Builder topK(Integer topK) {
            return (Builder) super.topK(topK);
        }
        public Builder enableSearch(Boolean enableSearch) {
            return (Builder) super.enableSearch(enableSearch);
        }
        public Builder seed(Integer seed) {
            return (Builder) super.seed(seed);
        }
        public QwenStreamingChatModel build() {
            ensureOptions();
            return new QwenStreamingChatModel(apiKey, modelName, topP, topK,
enableSearch, seed);
        }
    }
}
```

langchain4j-dashscope\src\main\java\dev\langchain4j\model\dashscope\QwenStreamingLanguageModel.java

```
package dev.langchain4j.model.dashscope;
import com.alibaba.dashscope.aigc.generation.GenerationResult;
import com.alibaba.dashscope.aigc.generation.models.QwenParam;
import com.alibaba.dashscope.common.ResultCallback;
import com.alibaba.dashscope.exception.InputRequiredException;
import com.alibaba.dashscope.exception.NoApiKeyException;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.language.StreamingLanguageModel;
import dev.langchain4j.model.output.Response;
import static
com.alibaba.dashscope.aigc.generation.models.QwenParam.ResultFormat.MESSAGE;
public class QwenStreamingLanguageModel extends QwenLanguageModel implements
StreamingLanguageModel {
    public QwenStreamingLanguageModel(String apiKey,
                                      String modelName,
                                      Double topP,
                                      Integer topK,
                                      Boolean enableSearch,
                                      Integer seed) {
        super(apiKey, modelName, topP, topK, enableSearch, seed);
    }
    @Override
    public void generate(String prompt, StreamingResponseHandler<String>
handler) {
        try {
            QwenParam param = QwenParam.builder()
                    .apiKey(apiKey)
                    .model(modelName)
                    .topP(topP)
                    .topK(topK)
                    .enableSearch(enableSearch)
                    .seed(seed)
                    .prompt(prompt)
                    .resultFormat(MESSAGE)
                    .build();
            QwenStreamingResponseBuilder responseBuilder = new
QwenStreamingResponseBuilder();
            generation.streamCall(param, new
ResultCallback<GenerationResult>() {
                @Override
                public void onEvent(GenerationResult result) {
                    String delta = responseBuilder.append(result);
                    if (delta != null) {
                        handler.onNext(delta);
                    }
                }
                @Override
                public void onComplete() {
                    Response<AiMessage> response = responseBuilder.build();
                    handler.onComplete(Response.from(
                            response.content().text(),
                            response.tokenUsage(),
                            response.finishReason()
                    ));
                }
                @Override
```

```
                public void onError(Exception e) {
                    handler.onError(e);
                }
            });
        } catch (NoApiKeyException | InputRequiredException e) {
            throw new RuntimeException(e);
        }
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder extends QwenLanguageModel.Builder {
        public Builder apiKey(String apiKey) {
            return (Builder) super.apiKey(apiKey);
        }
        public Builder modelName(String modelName) {
            return (Builder) super.modelName(modelName);
        }
        public Builder topP(Double topP) {
            return (Builder) super.topP(topP);
        }
        public Builder topK(Integer topK) {
            return (Builder) super.topK(topK);
        }
        public Builder enableSearch(Boolean enableSearch) {
            return (Builder) super.enableSearch(enableSearch);
        }
        public Builder seed(Integer seed) {
            return (Builder) super.seed(seed);
        }
        public QwenStreamingLanguageModel build() {
            ensureOptions();
            return new QwenStreamingLanguageModel(apiKey, modelName, topP,
topK, enableSearch, seed);
        }
    }
}
```

langchain4j-dashscope\src\main\java\dev\langchain4j\model\dashscope\QwenStreamingResponseBuilder.java

```
package dev.langchain4j.model.dashscope;
import com.alibaba.dashscope.aigc.generation.GenerationResult;
import com.alibaba.dashscope.aigc.generation.GenerationUsage;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.output.FinishReason;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import static dev.langchain4j.model.dashscope.QwenHelper.answerFrom;
import static dev.langchain4j.model.dashscope.QwenHelper.finishReasonFrom;
public class QwenStreamingResponseBuilder {
    private String generatedContent = "";
    private Integer inputTokenCount;
    private Integer outputTokenCount;
    private FinishReason finishReason;
    public QwenStreamingResponseBuilder() {}
    public String append(GenerationResult partialResponse) {
        if (partialResponse == null) {
            return null;
        }
        GenerationUsage usage = partialResponse.getUsage();
        if (usage != null) {
            inputTokenCount = usage.getInputTokens();
            outputTokenCount = usage.getOutputTokens();
        }
        FinishReason finishReason = finishReasonFrom(partialResponse);
        if (finishReason != null) {
            this.finishReason = finishReason;
        }
        String partialContent = answerFrom(partialResponse);
        String delta = null;
        if (partialContent.length() > generatedContent.length() ) {
            delta = partialContent.substring(generatedContent.length());
            generatedContent = partialContent;
        }
        return delta;
    }
    public Response<AiMessage> build() {
        return Response.from(
                AiMessage.from(generatedContent),
                new TokenUsage(inputTokenCount, outputTokenCount),
                finishReason
        );
    }
}
```

langchain4j-dashscope\src\test\java\dev\langchain4j\model\dashscope\QwenChatModelIT.java

```
package dev.langchain4j.model.dashscope;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.internal.Utils;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import static org.assertj.core.api.Assertions.assertThat;
public class QwenChatModelIT {
    @ParameterizedTest
    @MethodSource("dev.langchain4j.model.dashscope.QwenTestHelper#chatModelNam
eProvider")
    public void should_send_messages_and_receive_response(String modelName) {
        String apiKey = QwenTestHelper.apiKey();
        if (Utils.isNullOrBlank(apiKey)) {
            return;
        }
        ChatLanguageModel model = QwenChatModel.builder()
                .apiKey(apiKey)
                .modelName(modelName)
                .build();
        Response<AiMessage> response =
model.generate(QwenTestHelper.chatMessages());
        System.out.println(response);
        assertThat(response.content().text()).containsIgnoringCase("rain");
    }
}
```

langchain4j-dashscope\src\test\java\dev\langchain4j\model\dashscope\QwenEmbeddingModelIT.java

```
package dev.langchain4j.model.dashscope;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.internal.Utils;
import dev.langchain4j.model.embedding.EmbeddingModel;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import java.util.List;
import static dev.langchain4j.data.segment.TextSegment.textSegment;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
public class QwenEmbeddingModelIT {
    private EmbeddingModel getModel(String modelName) {
        String apiKey = QwenTestHelper.apiKey();
        if (Utils.isNullOrBlank(apiKey)) {
            return null;
        }
        return QwenEmbeddingModel.builder()
                .apiKey(apiKey)
                .modelName(modelName)
                .build();
    }
    @ParameterizedTest
    @MethodSource("dev.langchain4j.model.dashscope.QwenTestHelper#embeddingModelNameProvider")
    void should_embed_one_text(String modelName) {
        EmbeddingModel model = getModel(modelName);
        if (model == null) {
            return;
        }
        Embedding embedding = model.embed("hello").content();
        assertThat(embedding.vector()).isNotEmpty();
    }
    @ParameterizedTest
    @MethodSource("dev.langchain4j.model.dashscope.QwenTestHelper#embeddingModelNameProvider")
    void should_embed_documents(String modelName) {
        EmbeddingModel model = getModel(modelName);
        if (model == null) {
            return;
        }
        List<Embedding> embeddings = model.embedAll(asList(
                textSegment("hello"),
                textSegment("how are you?")
        )).content();
        assertThat(embeddings).hasSize(2);
        assertThat(embeddings.get(0).vector()).isNotEmpty();
        assertThat(embeddings.get(1).vector()).isNotEmpty();
    }
    @ParameterizedTest
    @MethodSource("dev.langchain4j.model.dashscope.QwenTestHelper#embeddingModelNameProvider")
    void should_embed_queries(String modelName) {
        EmbeddingModel model = getModel(modelName);
        if (model == null) {
            return;
        }
```

```
        List<Embedding> embeddings = model.embedAll(asList(
                textSegment("hello",
Metadata.from(QwenEmbeddingModel.TYPE_KEY, QwenEmbeddingModel.TYPE_QUERY)),
                textSegment("how are you?",
Metadata.from(QwenEmbeddingModel.TYPE_KEY, QwenEmbeddingModel.TYPE_QUERY))
        )).content();
        assertThat(embeddings).hasSize(2);
        assertThat(embeddings.get(0).vector()).isNotEmpty();
        assertThat(embeddings.get(1).vector()).isNotEmpty();
    }
    @ParameterizedTest
    @MethodSource("dev.langchain4j.model.dashscope.QwenTestHelper#embeddingMod
elNameProvider")
    void should_embed_mix_segments(String modelName) {
        EmbeddingModel model = getModel(modelName);
        if (model == null) {
            return;
        }
        List<Embedding> embeddings = model.embedAll(asList(
                textSegment("hello",
Metadata.from(QwenEmbeddingModel.TYPE_KEY, QwenEmbeddingModel.TYPE_QUERY)),
                textSegment("how are you?")
        )).content();
        assertThat(embeddings).hasSize(2);
        assertThat(embeddings.get(0).vector()).isNotEmpty();
        assertThat(embeddings.get(1).vector()).isNotEmpty();
    }
}
```

langchain4j-dashscope\src\test\java\dev\langchain4j\model\dashscope\QwenLanguageModelIT.java

```
package dev.langchain4j.model.dashscope;
import dev.langchain4j.internal.Utils;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import static org.assertj.core.api.Assertions.assertThat;
public class QwenLanguageModelIT {
    @ParameterizedTest
    @MethodSource("dev.langchain4j.model.dashscope.QwenTestHelper#languageModelNameProvider")
    public void should_send_messages_and_receive_response(String modelName) {
        String apiKey = QwenTestHelper.apiKey();
        if (Utils.isNullOrBlank(apiKey)) {
            return;
        }
        LanguageModel model = QwenLanguageModel.builder()
                .apiKey(apiKey)
                .modelName(modelName)
                .build();
        Response<String> response = model.generate("Please say 'hello' to me");
        System.out.println(response);
        assertThat(response.content()).containsIgnoringCase("hello");
    }
}
```

langchain4j-dashscope\src\test\java\dev\langchain4j\model\dashscope\QwenStreamingChatModelIT.java

```
package dev.langchain4j.model.dashscope;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.internal.Utils;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeoutException;
import static dev.langchain4j.model.dashscope.QwenTestHelper.chatMessages;
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.assertj.core.api.Assertions.assertThat;
public class QwenStreamingChatModelIT {
    @ParameterizedTest
    @MethodSource("dev.langchain4j.model.dashscope.QwenTestHelper#chatModelNameProvider")
    public void should_send_messages_and_receive_response(String modelName) throws ExecutionException, InterruptedException, TimeoutException {
        String apiKey = QwenTestHelper.apiKey();
        if (Utils.isNullOrBlank(apiKey)) {
            return;
        }
        StreamingChatLanguageModel model = QwenStreamingChatModel.builder()
                .apiKey(apiKey)
                .modelName(modelName)
                .build();
        CompletableFuture<Response<AiMessage>> future = new CompletableFuture<>();
        model.generate(chatMessages(), new StreamingResponseHandler<AiMessage>() {
            @Override
            public void onNext(String token) {
                System.out.println("onNext: '" + token + "'");
            }
            @Override
            public void onComplete(Response<AiMessage> response) {
                future.complete(response);
                System.out.println("onComplete: '" + response.content().text() + "'");
            }
            @Override
            public void onError(Throwable error) {
                future.completeExceptionally(error);
            }
        });
        Response<AiMessage> response = future.get(30, SECONDS);
        System.out.println(response);
        assertThat(response.content().text()).containsIgnoringCase("rain");
    }
}
```

langchain4j-dashscope\src\test\java\dev\langchain4j\model\dashscope\QwenStreamingLanguageModelIT.java

```
package dev.langchain4j.model.dashscope;
import dev.langchain4j.internal.Utils;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.language.StreamingLanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeoutException;
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.assertj.core.api.Assertions.assertThat;
public class QwenStreamingLanguageModelIT {
    @ParameterizedTest
    @MethodSource("dev.langchain4j.model.dashscope.QwenTestHelper#languageModelNameProvider")
    public void should_send_messages_and_receive_response(String modelName) throws ExecutionException, InterruptedException, TimeoutException {
        String apiKey = QwenTestHelper.apiKey();
        if (Utils.isNullOrBlank(apiKey)) {
            return;
        }
        StreamingLanguageModel model = QwenStreamingLanguageModel.builder()
                .apiKey(apiKey)
                .modelName(modelName)
                .build();
        CompletableFuture<Response<String>> future = new CompletableFuture<>();
        model.generate("Please say 'hello' to me", new StreamingResponseHandler<String>() {
            @Override
            public void onNext(String token) {
                System.out.println("onNext: '" + token + "'");
            }
            @Override
            public void onComplete(Response<String> response) {
                future.complete(response);
                System.out.println("onComplete: '" + response.content() + "'");
            }
            @Override
            public void onError(Throwable error) {
                future.completeExceptionally(error);
            }
        });
        Response<String> response = future.get(30, SECONDS);
        System.out.println(response);
        assertThat(response.content()).containsIgnoringCase("hello");
    }
}
```

langchain4j-dashscope\src\test\java\dev\langchain4j\model\dashscope\QwenTestHelper.java

```
package dev.langchain4j.model.dashscope;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.data.message.UserMessage;
import org.junit.jupiter.params.provider.Arguments;
import java.util.LinkedList;
import java.util.List;
import java.util.stream.Stream;
public class QwenTestHelper {
    public static Stream<Arguments> languageModelNameProvider() {
        return Stream.of(
                Arguments.of(QwenModelName.QWEN_TURBO),
                Arguments.of(QwenModelName.QWEN_PLUS),
                Arguments.of(QwenModelName.QWEN_SPARK_V1),
                Arguments.of(QwenModelName.QWEN_SPARK_V2),
                Arguments.of(QwenModelName.QWEN_7B_CHAT),
                Arguments.of(QwenModelName.QWEN_14B_CHAT)
        );
    }
    public static Stream<Arguments> chatModelNameProvider() {
        return Stream.of(
                Arguments.of(QwenModelName.QWEN_TURBO),
                Arguments.of(QwenModelName.QWEN_PLUS),
                Arguments.of(QwenModelName.QWEN_7B_CHAT),
                Arguments.of(QwenModelName.QWEN_14B_CHAT)
        );
    }
    public static Stream<Arguments> embeddingModelNameProvider() {
        return Stream.of(
                Arguments.of(QwenModelName.TEXT_EMBEDDING_V1)
        );
    }
    public static String apiKey() {
        return System.getenv("DASHSCOPE_API_KEY");
    }
    public static List<ChatMessage> chatMessages() {
        List<ChatMessage> messages = new LinkedList<>();
        messages.add(SystemMessage.from("Your name is Jack." +
                " You like to answer other people's questions briefly." +
                " It's rainy today."));
        messages.add(UserMessage.from("Hello. What's your name?"));
        messages.add(AiMessage.from("Jack."));
        messages.add(UserMessage.from("How about the weather today?"));
        return messages;
    }
}
```

langchain4j-elasticsearch\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-elasticsearch</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Elasticsearch</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>co.elastic.clients</groupId>
            <artifactId>elasticsearch-java</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
```

```
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-elasticsearch\src\main\java\dev\langchain4j\store\embedding\elasticsearch\Document.java

```
package dev.langchain4j.store.embedding.elasticsearch;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Map;
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
class Document {
    private float[] vector;
    private String text;
    private Map<String, String> metadata;
}
```

langchain4j-elasticsearch\src\main\java\dev\langchain4j\store\embedding\elasticsearch\ElasticsearchEmbeddingStore.java

```
package dev.langchain4j.store.embedding.elasticsearch;
import co.elastic.clients.elasticsearch.ElasticsearchClient;
import co.elastic.clients.elasticsearch._types.InlineScript;
import co.elastic.clients.elasticsearch._types.mapping.DenseVectorProperty;
import co.elastic.clients.elasticsearch._types.mapping.Property;
import co.elastic.clients.elasticsearch._types.mapping.TextProperty;
import co.elastic.clients.elasticsearch._types.mapping.TypeMapping;
import co.elastic.clients.elasticsearch._types.query_dsl.Query;
import co.elastic.clients.elasticsearch._types.query_dsl.ScriptScoreQuery;
import co.elastic.clients.elasticsearch.core.BulkRequest;
import co.elastic.clients.elasticsearch.core.BulkResponse;
import co.elastic.clients.elasticsearch.core.SearchRequest;
import co.elastic.clients.elasticsearch.core.SearchResponse;
import co.elastic.clients.elasticsearch.core.bulk.BulkResponseItem;
import co.elastic.clients.json.JsonData;
import co.elastic.clients.json.jackson.JacksonJsonpMapper;
import co.elastic.clients.transport.ElasticsearchTransport;
import co.elastic.clients.transport.endpoints.BooleanResponse;
import co.elastic.clients.transport.rest_client.RestClientTransport;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import org.apache.http.Header;
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.message.BasicHeader;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import static dev.langchain4j.internal.Utils.*;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
import static dev.langchain4j.internal.ValidationUtils.ensureTrue;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.toList;
/**
 * Represents an <a href="https://www.elastic.co/">Elasticsearch</a> index as an embedding store.
 * Current implementation assumes the index uses the cosine distance metric.
 */
public class ElasticsearchEmbeddingStore implements EmbeddingStore<TextSegment> {
    private static final Logger log = LoggerFactory.getLogger(ElasticsearchEmbeddingStore.class);
    private final ElasticsearchClient client;
```

```
    private final String indexName;
    private final ObjectMapper objectMapper;
    /**
     * Creates an instance of ElasticsearchEmbeddingStore.
     *
     * @param serverUrl Elasticsearch Server URL
     * @param apiKey    Elasticsearch API key (optional)
     * @param userName  Elasticsearch userName (optional)
     * @param password  Elasticsearch password (optional)
     * @param indexName Elasticsearch index name (optional). Default value:
"default"
     */
    public ElasticsearchEmbeddingStore(String serverUrl,
                                       String apiKey,
                                       String userName,
                                       String password,
                                       String indexName) {
        RestClientBuilder restClientBuilder = RestClient
                .builder(HttpHost.create(ensureNotNull(serverUrl,
"serverUrl")));
        if (!isNullOrBlank(userName)) {
            CredentialsProvider provider = new BasicCredentialsProvider();
            provider.setCredentials(AuthScope.ANY, new
UsernamePasswordCredentials(userName, password));
            restClientBuilder.setHttpClientConfigCallback(httpClientBuilder -
> httpClientBuilder.setDefaultCredentialsProvider(provider));
        }
        if (!isNullOrBlank(apiKey)) {
            restClientBuilder.setDefaultHeaders(new Header[]{
                    new BasicHeader("Authorization", "Apikey " + apiKey)
            });
        }
        ElasticsearchTransport transport = new
RestClientTransport(restClientBuilder.build(), new JacksonJsonpMapper());
        this.client = new ElasticsearchClient(transport);
        this.indexName = ensureNotNull(indexName, "indexName");
        this.objectMapper = new ObjectMapper();
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String serverUrl;
        private String apiKey;
        private String userName;
        private String password;
        private String indexName = "default";
        /**
         * @param serverUrl Elasticsearch Server URL
         */
        public Builder serverUrl(String serverUrl) {
            this.serverUrl = serverUrl;
            return this;
        }
        /**
         * @param apiKey Elasticsearch API key (optional)
         * @return builder
         */
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
```

```
        }
        /**
         * @param userName Elasticsearch userName (optional)
         * @return builder
         */
        public Builder userName(String userName) {
            this.userName = userName;
            return this;
        }
        /**
         * @param password Elasticsearch password (optional)
         * @return builder
         */
        public Builder password(String password) {
            this.password = password;
            return this;
        }
        /**
         * @param indexName Elasticsearch index name (optional).
         *                  Default value: "default".
         */
        public Builder indexName(String indexName) {
            this.indexName = indexName;
            return this;
        }
        public ElasticsearchEmbeddingStore build() {
            return new ElasticsearchEmbeddingStore(serverUrl, apiKey,
userName, password, indexName);
        }
    }
    @Override
    public String add(Embedding embedding) {
        String id = randomUUID();
        add(id, embedding);
        return id;
    }
    @Override
    public void add(String id, Embedding embedding) {
        addInternal(id, embedding, null);
    }
    @Override
    public String add(Embedding embedding, TextSegment textSegment) {
        String id = randomUUID();
        addInternal(id, embedding, textSegment);
        return id;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings) {
        List<String> ids = embeddings.stream()
                .map(ignored -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, null);
        return ids;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
embedded) {
        List<String> ids = embeddings.stream()
                .map(ignored -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, embedded);
```

```
        return ids;
    }
    @Override
    public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
referenceEmbedding, int maxResults, double minScore) {
        try {
            // Use Script Score and cosineSimilarity to calculate
            // see https://www.elastic.co/guide/en/elasticsearch/reference/
current/query-dsl-script-score-query.html#vector-functions-cosine
            ScriptScoreQuery scriptScoreQuery =
buildDefaultScriptScoreQuery(referenceEmbedding.vector(), (float) minScore);
            SearchResponse<Document> response = client.search(
                    SearchRequest.of(s -> s.index(indexName)
                            .query(n -> n.scriptScore(scriptScoreQuery))
                            .size(maxResults)),
                    Document.class
            );
            return toEmbeddingMatch(response);
        } catch (IOException e) {
            log.error("[ElasticSearch encounter I/O Exception]", e);
            throw new ElasticsearchRequestFailedException(e.getMessage());
        }
    }
    private void addInternal(String id, Embedding embedding, TextSegment
embedded) {
        addAllInternal(singletonList(id), singletonList(embedding), embedded
== null ? null : singletonList(embedded));
    }
    private void addAllInternal(List<String> ids, List<Embedding> embeddings,
List<TextSegment> embedded) {
        if (isCollectionEmpty(ids) || isCollectionEmpty(embeddings)) {
            log.info("[do not add empty embeddings to elasticsearch]");
            return;
        }
        ensureTrue(ids.size() == embeddings.size(), "ids size is not equal to
embeddings size");
        ensureTrue(embedded == null || embeddings.size() == embedded.size(),
"embeddings size is not equal to embedded size");
        try {
            createIndexIfNotExist(embeddings.get(0).dimensions());
            bulk(ids, embeddings, embedded);
        } catch (IOException e) {
            log.error("[ElasticSearch encounter I/O Exception]", e);
            throw new ElasticsearchRequestFailedException(e.getMessage());
        }
    }
    private void createIndexIfNotExist(int dim) throws IOException {
        BooleanResponse response = client.indices().exists(c ->
c.index(indexName));
        if (!response.value()) {
            client.indices().create(c -> c.index(indexName)
                    .mappings(getDefaultMappings(dim)));
        }
    }
    private TypeMapping getDefaultMappings(int dim) {
        Map<String, Property> properties = new HashMap<>(4);
        properties.put("text", Property.of(p -> p.text(TextProperty.of(t ->
t))));
        properties.put("vector", Property.of(p ->
p.denseVector(DenseVectorProperty.of(d -> d.dims(dim)))));
        return TypeMapping.of(c -> c.properties(properties));
```

```
    }
    private void bulk(List<String> ids, List<Embedding> embeddings,
List<TextSegment> embedded) throws IOException {
        int size = ids.size();
        BulkRequest.Builder bulkBuilder = new BulkRequest.Builder();
        for (int i = 0; i < size; i++) {
            int finalI = i;
            Document document = Document.builder()
                    .vector(embeddings.get(i).vector())
                    .text(embedded == null ? null : embedded.get(i).text())
                    .metadata(embedded == null ? null :
Optional.ofNullable(embedded.get(i).metadata())
                            .map(Metadata::asMap)
                            .orElse(null))
                    .build();
            bulkBuilder.operations(op -> op.index(idx -> idx
                    .index(indexName)
                    .id(ids.get(finalI))
                    .document(document)));
        }
        BulkResponse response = client.bulk(bulkBuilder.build());
        if (response.errors()) {
            for (BulkResponseItem item : response.items()) {
                if (item.error() != null) {
                    throw new ElasticsearchRequestFailedException("type: " +
item.error().type() + ", reason: " + item.error().reason());
                }
            }
        }
    }
    private ScriptScoreQuery buildDefaultScriptScoreQuery(float[] vector,
float minScore) throws JsonProcessingException {
        JsonData queryVector = toJsonData(vector);
        return ScriptScoreQuery.of(q -> q
                .minScore(minScore)
                .query(Query.of(qu -> qu.matchAll(m -> m)))
                .script(s -> s.inline(InlineScript.of(i -> i
                        // The script adds 1.0 to the cosine similarity to
prevent the score from being negative.
                        // divided by 2 to keep score in the range [0, 1]
                        .source("(cosineSimilarity(params.query_vector,
'vector') + 1.0) / 2")
                        .params("query_vector", queryVector)))));
    }
    private <T> JsonData toJsonData(T rawData) throws JsonProcessingException
{
        return JsonData.fromJson(objectMapper.writeValueAsString(rawData));
    }
    private List<EmbeddingMatch<TextSegment>>
toEmbeddingMatch(SearchResponse<Document> response) {
        return response.hits().hits().stream()
                .map(hit -> Optional.ofNullable(hit.source())
                        .map(document -> new EmbeddingMatch<>(
                                hit.score(),
                                hit.id(),
                                new Embedding(document.getVector()),
                                document.getText() == null
                                        ? null
                                        :
TextSegment.from(document.getText(), new Metadata(document.getMetadata()))
                        )).orElse(null))
```

```
                .collect(toList());
    }
}
```

langchain4j-elasticsearch\src\main\java\dev\langchain4j\store\embedding\elasticsearch\ElasticsearchRequestFailedException.java

```
package dev.langchain4j.store.embedding.elasticsearch;
public class ElasticsearchRequestFailedException extends RuntimeException {
    public ElasticsearchRequestFailedException() {
        super();
    }
    public ElasticsearchRequestFailedException(String message) {
        super(message);
    }
    public ElasticsearchRequestFailedException(String message, Throwable
cause) {
        super(message, cause);
    }
}
```

langchain4j-elasticsearch\src\test\java\dev\langchain4j\store\embedding\elasticsearch\ElasticsearchEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.elasticsearch;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
@Disabled("needs Elasticsearch to be running locally")
class ElasticsearchEmbeddingStoreTest {
    /**
     * First start elasticsearch locally:
     * docker pull docker.elastic.co/elasticsearch/elasticsearch:8.9.0
     * docker run -d -p 9200:9200 -p 9300:9300 -e discovery.type=single-node -
e xpack.security.enabled=false docker.elastic.co/elasticsearch/
elasticsearch:8.9.0
     */
    private final EmbeddingStore<TextSegment> embeddingStore =
ElasticsearchEmbeddingStore.builder()
            .serverUrl("http://localhost:9200")
            .indexName(randomUUID())
            .build();
    private final EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    @Test
    void should_add_embedding() throws InterruptedException {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() throws InterruptedException {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
```

```
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() throws InterruptedException {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_embedding_with_segment_with_metadata() throws
InterruptedException {
        TextSegment segment = TextSegment.from(randomUUID(),
Metadata.from("test-key", "test-value"));
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() throws InterruptedException {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
```

```
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() throws
InterruptedException {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
        );
        assertThat(ids).hasSize(2);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() throws InterruptedException {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
```

```
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() throws InterruptedException {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
referenceEmbedding)),
                withPercentage(1)
        );
    }
}
```

langchain4j-hugging-face\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-hugging-face</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Hugging Face</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.squareup.retrofit2</groupId>
            <artifactId>retrofit</artifactId>
        </dependency>
        <dependency>
            <groupId>com.squareup.retrofit2</groupId>
            <artifactId>converter-gson</artifactId>
        </dependency>
        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
```

```
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\ApiKeyInsertingInterceptor.java

```
package dev.langchain4j.model.huggingface;
import okhttp3.Interceptor;
import okhttp3.Request;
import okhttp3.Response;
import java.io.IOException;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
class ApiKeyInsertingInterceptor implements Interceptor {
    private final String apiKey;
    ApiKeyInsertingInterceptor(String apiKey) {
        this.apiKey = ensureNotBlank(apiKey, "apiKey");
    }
    @Override
    public Response intercept(Chain chain) throws IOException {
        Request request = chain.request()
                .newBuilder()
                .addHeader("Authorization", "Bearer " + apiKey)
                .build();
        return chain.proceed(request);
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\EmbeddingRequest.java

```
package dev.langchain4j.model.huggingface;
import java.util.List;
class EmbeddingRequest {
    private final List<String> inputs;
    private final Options options;
    EmbeddingRequest(List<String> inputs, boolean waitForModel) {
        this.inputs = inputs;
        this.options = Options.builder()
                .waitForModel(waitForModel)
                .build();
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\HuggingFaceApi.java

```
package dev.langchain4j.model.huggingface;
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.Headers;
import retrofit2.http.POST;
import retrofit2.http.Path;
import java.util.List;
interface HuggingFaceApi {
    @POST("/models/{modelId}")
    @Headers({"Content-Type: application/json"})
    Call<List<TextGenerationResponse>> generate(@Body TextGenerationRequest
request, @Path("modelId") String modelId);
    @POST("/pipeline/feature-extraction/{modelId}")
    @Headers({"Content-Type: application/json"})
    Call<List<float[]>> embed(@Body EmbeddingRequest request,
@Path("modelId") String modelId);
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\HuggingFaceChatModel.java

```
package dev.langchain4j.model.huggingface;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.output.Response;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.Utils.isNullOrBlank;
import static dev.langchain4j.model.huggingface.HuggingFaceModelName.TII_UAE_FALCON_7B_INSTRUCT;
import static java.util.stream.Collectors.joining;
public class HuggingFaceChatModel implements ChatLanguageModel {
    private final HuggingFaceClient client;
    private final Double temperature;
    private final Integer maxNewTokens;
    private final Boolean returnFullText;
    private final Boolean waitForModel;
    public HuggingFaceChatModel(String accessToken,
                                String modelId,
                                Duration timeout,
                                Double temperature,
                                Integer maxNewTokens,
                                Boolean returnFullText,
                                Boolean waitForModel) {
        this(HuggingFaceChatModel.builder()
                .accessToken(accessToken)
                .modelId(modelId)
                .timeout(timeout)
                .temperature(temperature)
                .maxNewTokens(maxNewTokens)
                .returnFullText(returnFullText)
                .waitForModel(waitForModel));
    }
    public HuggingFaceChatModel(Builder builder) {
        this.client = new HuggingFaceClient(builder.accessToken, builder.modelId, builder.timeout);
        this.temperature = builder.temperature;
        this.maxNewTokens = builder.maxNewTokens;
        this.returnFullText = builder.returnFullText;
        this.waitForModel = builder.waitForModel;
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages) {
        TextGenerationRequest request = TextGenerationRequest.builder()
                .inputs(messages.stream()
                        .map(ChatMessage::text)
                        .collect(joining("\n")))
                .parameters(Parameters.builder()
                        .temperature(temperature)
                        .maxNewTokens(maxNewTokens)
                        .returnFullText(returnFullText)
                        .build())
                .options(Options.builder()
                        .waitForModel(waitForModel)
                        .build())
                .build();
```

```
        TextGenerationResponse textGenerationResponse = client.chat(request);
        return
Response.from(AiMessage.from(textGenerationResponse.generatedText()));
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
List<ToolSpecification> toolSpecifications) {
        throw new IllegalArgumentException("Tools are currently not supported
for HuggingFace models");
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
ToolSpecification toolSpecification) {
        throw new IllegalArgumentException("Tools are currently not supported
for HuggingFace models");
    }
    public static Builder builder() {
        return new Builder();
    }
    public static final class Builder {
        private String accessToken;
        private String modelId = TII_UAE_FALCON_7B_INSTRUCT;
        private Duration timeout = Duration.ofSeconds(15);
        private Double temperature;
        private Integer maxNewTokens;
        private Boolean returnFullText = false;
        private Boolean waitForModel = true;
        public Builder accessToken(String accessToken) {
            this.accessToken = accessToken;
            return this;
        }
        public Builder modelId(String modelId) {
            if (modelId != null) {
                this.modelId = modelId;
            }
            return this;
        }
        public Builder timeout(Duration timeout) {
            if (timeout != null) {
                this.timeout = timeout;
            }
            return this;
        }
        public Builder temperature(Double temperature) {
            this.temperature = temperature;
            return this;
        }
        public Builder maxNewTokens(Integer maxNewTokens) {
            this.maxNewTokens = maxNewTokens;
            return this;
        }
        public Builder returnFullText(Boolean returnFullText) {
            if (returnFullText != null) {
                this.returnFullText = returnFullText;
            }
            return this;
        }
        public Builder waitForModel(Boolean waitForModel) {
            if (waitForModel != null) {
                this.waitForModel = waitForModel;
            }
```

```
            return this;
        }
        public HuggingFaceChatModel build() {
            if (isNullOrBlank(accessToken)) {
                throw new IllegalArgumentException("HuggingFace access token
must be defined. It can be generated here: https://huggingface.co/settings/
tokens");
            }
            return new HuggingFaceChatModel(this);
        }
    }
    public static HuggingFaceChatModel withAccessToken(String accessToken) {
        return builder().accessToken(accessToken).build();
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\HuggingFaceClient.java

```
package dev.langchain4j.model.huggingface;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import okhttp3.OkHttpClient;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
import java.io.IOException;
import java.time.Duration;
import java.util.List;
import static com.google.gson.FieldNamingPolicy.LOWER_CASE_WITH_UNDERSCORES;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
class HuggingFaceClient {
    private final HuggingFaceApi huggingFaceApi;
    private final String modelId;
    HuggingFaceClient(String apiKey, String modelId, Duration timeout) {
        OkHttpClient okHttpClient = new OkHttpClient.Builder()
                .addInterceptor(new ApiKeyInsertingInterceptor(apiKey))
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .build();
        Gson gson = new GsonBuilder()
                .setFieldNamingPolicy(LOWER_CASE_WITH_UNDERSCORES)
                .create();
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("https://api-inference.huggingface.co")
                .client(okHttpClient)
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build();
        this.huggingFaceApi = retrofit.create(HuggingFaceApi.class);
        this.modelId = ensureNotBlank(modelId, "modelId");
    }
    TextGenerationResponse chat(TextGenerationRequest request) {
        return generate(request);
    }
    TextGenerationResponse generate(TextGenerationRequest request) {
        try {
            retrofit2.Response<List<TextGenerationResponse>> retrofitResponse
                    = huggingFaceApi.generate(request, modelId).execute();
            if (retrofitResponse.isSuccessful()) {
                return toOneResponse(retrofitResponse);
            } else {
                throw toException(retrofitResponse);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private static TextGenerationResponse
toOneResponse(Response<List<TextGenerationResponse>> retrofitResponse) {
        List<TextGenerationResponse> responses = retrofitResponse.body();
        if (responses != null && responses.size() == 1) {
            return responses.get(0);
        } else {
            throw new RuntimeException("Expected only one generated_text, but
```

```
was: " + (responses == null ? 0 : responses.size()));
        }
    }
    List<float[]> embed(EmbeddingRequest request) {
        try {
            retrofit2.Response<List<float[]>> retrofitResponse =
huggingFaceApi.embed(request, modelId).execute();
            if (retrofitResponse.isSuccessful()) {
                return retrofitResponse.body();
            } else {
                throw toException(retrofitResponse);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private static RuntimeException toException(retrofit2.Response<?>
response) throws IOException {
        int code = response.code();
        String body = response.errorBody().string();
        String errorMessage = String.format("status code: %s; body: %s",
code, body);
        return new RuntimeException(errorMessage);
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\HuggingFaceEmbeddingModel.java

```
package dev.langchain4j.model.huggingface;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.model.huggingface.HuggingFaceModelName.SENTENCE_TRANSFORMERS_ALL_MINI_LM_L6_V2;
import static java.util.stream.Collectors.toList;
public class HuggingFaceEmbeddingModel implements EmbeddingModel {
    private static final Duration DEFAULT_TIMEOUT = Duration.ofSeconds(15);
    private final HuggingFaceClient client;
    private final boolean waitForModel;
    @Builder
    public HuggingFaceEmbeddingModel(String accessToken, String modelId, Boolean waitForModel, Duration timeout) {
        if (accessToken == null || accessToken.trim().isEmpty()) {
            throw new IllegalArgumentException("HuggingFace access token must be defined. It can be generated here: https://huggingface.co/settings/tokens");
        }
        this.client = new HuggingFaceClient(
                accessToken,
                modelId == null ? SENTENCE_TRANSFORMERS_ALL_MINI_LM_L6_V2 : modelId,
                timeout == null ? DEFAULT_TIMEOUT : timeout
        );
        this.waitForModel = waitForModel == null ? true : waitForModel;
    }
    @Override
    public Response<List<Embedding>> embedAll(List<TextSegment> textSegments) {
        List<String> texts = textSegments.stream()
                .map(TextSegment::text)
                .collect(toList());
        return embedTexts(texts);
    }
    private Response<List<Embedding>> embedTexts(List<String> texts) {
        EmbeddingRequest request = new EmbeddingRequest(texts, waitForModel);
        List<float[]> response = client.embed(request);
        List<Embedding> embeddings = response.stream()
                .map(Embedding::from)
                .collect(toList());
        return Response.from(embeddings);
    }
    public static HuggingFaceEmbeddingModel withAccessToken(String accessToken) {
        return builder().accessToken(accessToken).build();
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\HuggingFaceLanguageModel.java

```
package dev.langchain4j.model.huggingface;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.output.Response;
import java.time.Duration;
import static dev.langchain4j.model.huggingface.HuggingFaceModelName.TII_UAE_FALCON_7B_INSTRUCT;
public class HuggingFaceLanguageModel implements LanguageModel {
    private final HuggingFaceClient client;
    private final Double temperature;
    private final Integer maxNewTokens;
    private final Boolean returnFullText;
    private final Boolean waitForModel;
    public HuggingFaceLanguageModel(String accessToken,
                                    String modelId,
                                    Duration timeout,
                                    Double temperature,
                                    Integer maxNewTokens,
                                    Boolean returnFullText,
                                    Boolean waitForModel) {
        this(HuggingFaceLanguageModel.builder()
                .accessToken(accessToken)
                .modelId(modelId)
                .timeout(timeout)
                .temperature(temperature)
                .maxNewTokens(maxNewTokens)
                .returnFullText(returnFullText)
                .waitForModel(waitForModel)
        );
    }
    public HuggingFaceLanguageModel(Builder builder) {
        this.client = new HuggingFaceClient(builder.accessToken, builder.modelId, builder.timeout);
        this.temperature = builder.temperature;
        this.maxNewTokens = builder.maxNewTokens;
        this.returnFullText = builder.returnFullText;
        this.waitForModel = builder.waitForModel;
    }
    @Override
    public Response<String> generate(String prompt) {
        TextGenerationRequest request = TextGenerationRequest.builder()
                .inputs(prompt)
                .parameters(Parameters.builder()
                        .temperature(temperature)
                        .maxNewTokens(maxNewTokens)
                        .returnFullText(returnFullText)
                        .build())
                .options(Options.builder()
                        .waitForModel(waitForModel)
                        .build())
                .build();
        TextGenerationResponse response = client.generate(request);
        return Response.from(response.generatedText());
    }
    public static Builder builder() {
        return new Builder();
    }
    public static final class Builder {
```

```
        private String accessToken;
        private String modelId = TII_UAE_FALCON_7B_INSTRUCT;
        private Duration timeout = Duration.ofSeconds(15);
        private Double temperature;
        private Integer maxNewTokens;
        private Boolean returnFullText = false;
        private Boolean waitForModel = true;
        public Builder accessToken(String accessToken) {
            this.accessToken = accessToken;
            return this;
        }
        public Builder modelId(String modelId) {
            if (modelId != null) {
                this.modelId = modelId;
            }
            return this;
        }
        public Builder timeout(Duration timeout) {
            if (timeout != null) {
                this.timeout = timeout;
            }
            return this;
        }
        public Builder temperature(Double temperature) {
            this.temperature = temperature;
            return this;
        }
        public Builder maxNewTokens(Integer maxNewTokens) {
            this.maxNewTokens = maxNewTokens;
            return this;
        }
        public Builder returnFullText(Boolean returnFullText) {
            if (returnFullText != null) {
                this.returnFullText = returnFullText;
            }
            return this;
        }
        public Builder waitForModel(Boolean waitForModel) {
            if (waitForModel != null) {
                this.waitForModel = waitForModel;
            }
            return this;
        }
        public HuggingFaceLanguageModel build() {
            if (accessToken == null || accessToken.trim().isEmpty()) {
                throw new IllegalArgumentException("HuggingFace access token
must be defined. It can be generated here: https://huggingface.co/settings/
tokens");
            }
            return new HuggingFaceLanguageModel(this);
        }
    }
    public static HuggingFaceLanguageModel withAccessToken(String
accessToken) {
        return builder().accessToken(accessToken).build();
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\HuggingFaceModelName.java

```
package dev.langchain4j.model.huggingface;
public class HuggingFaceModelName {
    // Use with HuggingFaceChatModel and HuggingFaceLanguageModel
    public static final String TII_UAE_FALCON_7B_INSTRUCT = "tiiuae/falcon-7b-instruct";
    // Use with HuggingFaceEmbeddingModel
    public static final String SENTENCE_TRANSFORMERS_ALL_MINI_LM_L6_V2 = "sentence-transformers/all-MiniLM-L6-v2";
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\Options.java

```
package dev.langchain4j.model.huggingface;
import java.util.Objects;
class Options {
    private final Boolean waitForModel;
    private final Boolean useCache;
    Options(Builder builder) {
        this.waitForModel = builder.waitForModel;
        this.useCache = builder.useCache;
    }
    @Override
    public boolean equals(Object another) {
        if (this == another) return true;
        return another instanceof Options
                && equalTo((Options) another);
    }
    private boolean equalTo(Options another) {
        return Objects.equals(waitForModel, another.waitForModel)
                && Objects.equals(useCache, another.useCache);
    }
    @Override
    public int hashCode() {
        int h = 5381;
        h += (h << 5) + Objects.hashCode(waitForModel);
        h += (h << 5) + Objects.hashCode(useCache);
        return h;
    }
    @Override
    public String toString() {
        return "TextGenerationRequest {"
                + " waitForModel = " + waitForModel
                + ", useCache = " + useCache
                + " }";
    }
    static Builder builder() {
        return new Builder();
    }
    static final class Builder {
        private Boolean waitForModel = true;
        private Boolean useCache;
        Builder waitForModel(Boolean waitForModel) {
            if (waitForModel != null) {
                this.waitForModel = waitForModel;
            }
            return this;
        }
        Builder useCache(Boolean useCache) {
            this.useCache = useCache;
            return this;
        }
        Options build() {
            return new Options(this);
        }
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\Parameters.java

```
package dev.langchain4j.model.huggingface;
import java.util.Objects;
class Parameters {
    private final Integer topK;
    private final Double topP;
    private final Double temperature;
    private final Double repetitionPenalty;
    private final Integer maxNewTokens;
    private final Double maxTime;
    private final Boolean returnFullText;
    private final Integer numReturnSequences;
    private final Boolean doSample;
    Parameters(Builder builder) {
        this.topK = builder.topK;
        this.topP = builder.topP;
        this.temperature = builder.temperature;
        this.repetitionPenalty = builder.repetitionPenalty;
        this.maxNewTokens = builder.maxNewTokens;
        this.maxTime = builder.maxTime;
        this.returnFullText = builder.returnFullText;
        this.numReturnSequences = builder.numReturnSequences;
        this.doSample = builder.doSample;
    }
    @Override
    public boolean equals(Object another) {
        if (this == another) return true;
        return another instanceof Parameters
                && equalTo((Parameters) another);
    }
    private boolean equalTo(Parameters another) {
        return Objects.equals(topK, another.topK)
                && Objects.equals(topP, another.topP)
                && Objects.equals(temperature, another.temperature)
                && Objects.equals(repetitionPenalty,
another.repetitionPenalty)
                && Objects.equals(maxNewTokens, another.maxNewTokens)
                && Objects.equals(maxTime, another.maxTime)
                && Objects.equals(returnFullText, another.returnFullText)
                && Objects.equals(numReturnSequences,
another.numReturnSequences)
                && Objects.equals(doSample, another.doSample);
    }
    @Override
    public int hashCode() {
        int h = 5381;
        h += (h << 5) + Objects.hashCode(topK);
        h += (h << 5) + Objects.hashCode(topP);
        h += (h << 5) + Objects.hashCode(temperature);
        h += (h << 5) + Objects.hashCode(repetitionPenalty);
        h += (h << 5) + Objects.hashCode(maxNewTokens);
        h += (h << 5) + Objects.hashCode(maxTime);
        h += (h << 5) + Objects.hashCode(returnFullText);
        h += (h << 5) + Objects.hashCode(numReturnSequences);
        h += (h << 5) + Objects.hashCode(doSample);
        return h;
    }
    @Override
```

```
public String toString() {
    return "TextGenerationRequest {"
            + " topK = " + topK
            + ", topP = " + topP
            + ", temperature = " + temperature
            + ", repetitionPenalty = " + repetitionPenalty
            + ", maxNewTokens = " + maxNewTokens
            + ", maxTime = " + maxTime
            + ", returnFullText = " + returnFullText
            + ", numReturnSequences = " + numReturnSequences
            + ", doSample = " + doSample
            + " }";
}
static Builder builder() {
    return new Builder();
}
static final class Builder {
    private Integer topK;
    private Double topP;
    private Double temperature;
    private Double repetitionPenalty;
    private Integer maxNewTokens;
    private Double maxTime;
    private Boolean returnFullText;
    private Integer numReturnSequences;
    private Boolean doSample;
    Builder topK(Integer topK) {
        this.topK = topK;
        return this;
    }
    Builder topP(Double topP) {
        this.topP = topP;
        return this;
    }
    Builder temperature(Double temperature) {
        this.temperature = temperature;
        return this;
    }
    Builder repetitionPenalty(Double repetitionPenalty) {
        this.repetitionPenalty = repetitionPenalty;
        return this;
    }
    Builder maxNewTokens(Integer maxNewTokens) {
        this.maxNewTokens = maxNewTokens;
        return this;
    }
    Builder maxTime(Double maxTime) {
        this.maxTime = maxTime;
        return this;
    }
    Builder returnFullText(Boolean returnFullText) {
        this.returnFullText = returnFullText;
        return this;
    }
    Builder numReturnSequences(Integer numReturnSequences) {
        this.numReturnSequences = numReturnSequences;
        return this;
    }
    Builder doSample(Boolean doSample) {
        this.doSample = doSample;
        return this;
```

```
        }
        Parameters build() {
            return new Parameters(this);
        }
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\TextGenerationRequest.java

```
package dev.langchain4j.model.huggingface;
import java.util.Objects;
import static dev.langchain4j.internal.Utils.quoted;
class TextGenerationRequest {
    private final String inputs;
    private final Parameters parameters;
    private final Options options;
    TextGenerationRequest(Builder builder) {
        this.inputs = builder.inputs;
        this.parameters = builder.parameters;
        this.options = builder.options;
    }
    @Override
    public boolean equals(Object another) {
        if (this == another) return true;
        return another instanceof TextGenerationRequest
                && equalTo((TextGenerationRequest) another);
    }
    private boolean equalTo(TextGenerationRequest another) {
        return Objects.equals(inputs, another.inputs)
                && Objects.equals(parameters, another.parameters)
                && Objects.equals(options, another.options);
    }
    @Override
    public int hashCode() {
        int h = 5381;
        h += (h << 5) + Objects.hashCode(inputs);
        h += (h << 5) + Objects.hashCode(parameters);
        h += (h << 5) + Objects.hashCode(options);
        return h;
    }
    @Override
    public String toString() {
        return "TextGenerationRequest {"
                + " inputs = " + quoted(inputs)
                + ", parameters = " + parameters
                + ", options = " + options
                + " }";
    }
    static Builder builder() {
        return new Builder();
    }
    static final class Builder {
        private String inputs;
        private Parameters parameters;
        private Options options;
        Builder inputs(String inputs) {
            this.inputs = inputs;
            return this;
        }
        Builder parameters(Parameters parameters) {
            this.parameters = parameters;
            return this;
        }
        Builder options(Options options) {
            this.options = options;
            return this;
```

```
        }
        TextGenerationRequest build() {
            return new TextGenerationRequest(this);
        }
    }
}
```

langchain4j-hugging-face\src\main\java\dev\langchain4j\model\huggingface\TextGenerationResponse.java

```
package dev.langchain4j.model.huggingface;
class TextGenerationResponse {
    private final String generatedText;
    TextGenerationResponse(String generatedText) {
        this.generatedText = generatedText;
    }
    public String generatedText() {
        return generatedText;
    }
}
```

langchain4j-hugging-face\src\test\java\dev\langchain4j\model\huggingface\HuggingFaceChatModelIT.java

```
package dev.langchain4j.model.huggingface;
import dev.langchain4j.data.message.AiMessage;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.NullAndEmptySource;
import static dev.langchain4j.data.message.SystemMessage.systemMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.model.huggingface.HuggingFaceModelName.TII_UAE_FALCON_7B_INSTRUCT;
import static java.time.Duration.ofSeconds;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
class HuggingFaceChatModelIT {
    @Test
    public void should_send_messages_and_receive_response() {
        HuggingFaceChatModel model = HuggingFaceChatModel.builder()
                .accessToken(System.getenv("HF_API_KEY"))
                .modelId(TII_UAE_FALCON_7B_INSTRUCT)
                .timeout(ofSeconds(15))
                .temperature(0.7)
                .maxNewTokens(20)
                .waitForModel(true)
                .build();
        AiMessage aiMessage = model.generate(
                systemMessage("You are a good friend of mine, who likes to answer with jokes"),
                userMessage("Hey Bro, what are you doing?")
        ).content();
        assertThat(aiMessage.text()).isNotBlank();
        System.out.println(aiMessage.text());
    }
    @ParameterizedTest
    @NullAndEmptySource
    public void should_fail_when_access_token_is_null_or_empty(String accessToken) {
        assertThatThrownBy(() -> HuggingFaceChatModel.withAccessToken(accessToken))
                .isExactlyInstanceOf(IllegalArgumentException.class)
                .hasMessage("HuggingFace access token must be defined. It can be generated here: https://huggingface.co/settings/tokens");
    }
}
```

langchain4j-hugging-face\src\test\java\dev\langchain4j\model\huggingface\HuggingFaceEmbeddingModelIT.java

```
package dev.langchain4j.model.huggingface;
import dev.langchain4j.data.embedding.Embedding;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.data.segment.TextSegment.textSegment;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
class HuggingFaceEmbeddingModelIT {
    HuggingFaceEmbeddingModel model = HuggingFaceEmbeddingModel.builder()
            .accessToken(System.getenv("HF_API_KEY"))
            .modelId("sentence-transformers/all-MiniLM-L6-v2")
            .build();
    @Test
    void should_embed_one_text() {
        Embedding embedding = model.embed("hello").content();
        assertThat(embedding.vector()).hasSize(384);
    }
    @Test
    void should_embed_multiple_segments() {
        List<Embedding> embeddings = model.embedAll(asList(
                textSegment("hello"),
                textSegment("how are you?")
        )).content();
        assertThat(embeddings).hasSize(2);
        assertThat(embeddings.get(0).vector()).hasSize(384);
        assertThat(embeddings.get(1).vector()).hasSize(384);
    }
}
```

langchain4j-hugging-face\src\test\java\dev\langchain4j\model\huggingface\HuggingFaceLanguageModelIT.java

```
package dev.langchain4j.model.huggingface;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.NullAndEmptySource;
import static dev.langchain4j.model.huggingface.HuggingFaceModelName.TII_UAE_FALCON_7B_INSTRUCT;
import static java.time.Duration.ofSeconds;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
class HuggingFaceLanguageModelIT {
    @Test
    public void should_send_prompt_and_receive_response() {
        HuggingFaceLanguageModel model = HuggingFaceLanguageModel.builder()
                .accessToken(System.getenv("HF_API_KEY"))
                .modelId(TII_UAE_FALCON_7B_INSTRUCT)
                .timeout(ofSeconds(15))
                .temperature(0.7)
                .maxNewTokens(20)
                .waitForModel(true)
                .build();
        String answer = model.generate("What is the capital of the USA?").content();
        assertThat(answer).containsIgnoringCase("Washington");
        System.out.println(answer);
    }
    @ParameterizedTest
    @NullAndEmptySource
    public void should_fail_when_access_token_is_null_or_empty(String accessToken) {
        assertThatThrownBy(() -> HuggingFaceLanguageModel.withAccessToken(accessToken))
                .isExactlyInstanceOf(IllegalArgumentException.class)
                .hasMessage("HuggingFace access token must be defined. It can be generated here: https://huggingface.co/settings/tokens");
    }
}
```

langchain4j-local-ai\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-local-ai</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with LocalAI</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-open-ai</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-local-ai\src\main\java\dev\langchain4j\model\localai\LocalAiChatModel.java

```
package dev.langchain4j.model.localai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.chat.ChatCompletionRequest;
import dev.ai4j.openai4j.chat.ChatCompletionResponse;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.*;
import static java.time.Duration.ofSeconds;
import static java.util.Collections.singletonList;
public class LocalAiChatModel implements ChatLanguageModel {
    private final OpenAiClient client;
    private final String modelName;
    private final Double temperature;
    private final Double topP;
    private final Integer maxTokens;
    private final Integer maxRetries;
    @Builder
    public LocalAiChatModel(String baseUrl,
                            String modelName,
                            Double temperature,
                            Double topP,
                            Integer maxTokens,
                            Duration timeout,
                            Integer maxRetries,
                            Boolean logRequests,
                            Boolean logResponses) {
        temperature = temperature == null ? 0.7 : temperature;
        timeout = timeout == null ? ofSeconds(60) : timeout;
        maxRetries = maxRetries == null ? 3 : maxRetries;
        this.client = OpenAiClient.builder()
                .openAiApiKey("ignored")
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.modelName = ensureNotBlank(modelName, "modelName");
        this.temperature = temperature;
        this.topP = topP;
        this.maxTokens = maxTokens;
        this.maxRetries = maxRetries;
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages) {
        return generate(messages, null, null);
    }
```

```
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
List<ToolSpecification> toolSpecifications) {
        return generate(messages, toolSpecifications, null);
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
ToolSpecification toolSpecification) {
        return generate(messages, singletonList(toolSpecification),
toolSpecification);
    }
    private Response<AiMessage> generate(List<ChatMessage> messages,
                                         List<ToolSpecification>
toolSpecifications,
                                         ToolSpecification
toolThatMustBeExecuted
    ) {
        ChatCompletionRequest.Builder requestBuilder =
ChatCompletionRequest.builder()
                .model(modelName)
                .messages(toOpenAiMessages(messages))
                .temperature(temperature)
                .topP(topP)
                .maxTokens(maxTokens);
        if (toolSpecifications != null && !toolSpecifications.isEmpty()) {
            requestBuilder.functions(toFunctions(toolSpecifications));
        }
        if (toolThatMustBeExecuted != null) {
            requestBuilder.functionCall(toolThatMustBeExecuted.name());
        }
        ChatCompletionRequest request = requestBuilder.build();
        ChatCompletionResponse response = withRetry(() ->
client.chatCompletion(request).execute(), maxRetries);
        return Response.from(aiMessageFrom(response));
    }
}
```

langchain4j-local-ai\src\main\java\dev\langchain4j\model\localai\LocalAiEmbeddingModel.java

```
package dev.langchain4j.model.localai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.embedding.EmbeddingRequest;
import dev.ai4j.openai4j.embedding.EmbeddingResponse;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static java.time.Duration.ofSeconds;
import static java.util.stream.Collectors.toList;
public class LocalAiEmbeddingModel implements EmbeddingModel {
    private final OpenAiClient client;
    private final String modelName;
    private final Integer maxRetries;
    @Builder
    public LocalAiEmbeddingModel(String baseUrl,
                                 String modelName,
                                 Duration timeout,
                                 Integer maxRetries,
                                 Boolean logRequests,
                                 Boolean logResponses) {
        timeout = timeout == null ? ofSeconds(60) : timeout;
        maxRetries = maxRetries == null ? 3 : maxRetries;
        this.client = OpenAiClient.builder()
                .openAiApiKey("ignored")
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.modelName = ensureNotBlank(modelName, "modelName");
        this.maxRetries = maxRetries;
    }
    @Override
    public Response<List<Embedding>> embedAll(List<TextSegment> textSegments)
{
        List<String> texts = textSegments.stream()
                .map(TextSegment::text)
                .collect(toList());
        EmbeddingRequest request = EmbeddingRequest.builder()
                .input(texts)
                .model(modelName)
                .build();
        EmbeddingResponse response = withRetry(() ->
client.embedding(request).execute(), maxRetries);
        List<Embedding> embeddings = response.data().stream()
                .map(openAiEmbedding ->
Embedding.from(openAiEmbedding.embedding()))
                .collect(toList());
```

```
        return Response.from(embeddings);
    }
}
```

langchain4j-local-ai\src\main\java\dev\langchain4j\model\localai\LocalAiLanguageModel.java

```
package dev.langchain4j.model.localai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.completion.CompletionRequest;
import dev.ai4j.openai4j.completion.CompletionResponse;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.time.Duration;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static java.time.Duration.ofSeconds;
public class LocalAiLanguageModel implements LanguageModel {
    private final OpenAiClient client;
    private final String modelName;
    private final Double temperature;
    private final Double topP;
    private final Integer maxTokens;
    private final Integer maxRetries;
    @Builder
    public LocalAiLanguageModel(String baseUrl,
                                String modelName,
                                Double temperature,
                                Double topP,
                                Integer maxTokens,
                                Duration timeout,
                                Integer maxRetries,
                                Boolean logRequests,
                                Boolean logResponses) {
        temperature = temperature == null ? 0.7 : temperature;
        timeout = timeout == null ? ofSeconds(60) : timeout;
        maxRetries = maxRetries == null ? 3 : maxRetries;
        this.client = OpenAiClient.builder()
                .openAiApiKey("ignored")
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.modelName = ensureNotBlank(modelName, "modelName");
        this.temperature = temperature;
        this.topP = topP;
        this.maxTokens = maxTokens;
        this.maxRetries = maxRetries;
    }
    @Override
    public Response<String> generate(String prompt) {
        CompletionRequest request = CompletionRequest.builder()
                .model(modelName)
                .prompt(prompt)
                .temperature(temperature)
                .topP(topP)
                .maxTokens(maxTokens)
                .build();
        CompletionResponse response = withRetry(() ->
```

```
client.completion(request).execute(), maxRetries);
        return Response.from(response.text());
    }
}
```

langchain4j-local-ai\src\main\java\dev\langchain4j\model\localai\LocalAiStreamingChatModel.java

```
package dev.langchain4j.model.localai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.chat.ChatCompletionChoice;
import dev.ai4j.openai4j.chat.ChatCompletionRequest;
import dev.ai4j.openai4j.chat.ChatCompletionResponse;
import dev.ai4j.openai4j.chat.Delta;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiStreamingResponseBuilder;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.toFunctions;
import static
dev.langchain4j.model.openai.InternalOpenAiHelper.toOpenAiMessages;
import static java.time.Duration.ofSeconds;
import static java.util.Collections.singletonList;
public class LocalAiStreamingChatModel implements StreamingChatLanguageModel {
    private final OpenAiClient client;
    private final String modelName;
    private final Double temperature;
    private final Double topP;
    private final Integer maxTokens;
    @Builder
    public LocalAiStreamingChatModel(String baseUrl,
                                     String modelName,
                                     Double temperature,
                                     Double topP,
                                     Integer maxTokens,
                                     Duration timeout,
                                     Boolean logRequests,
                                     Boolean logResponses) {
        temperature = temperature == null ? 0.7 : temperature;
        timeout = timeout == null ? ofSeconds(60) : timeout;
        this.client = OpenAiClient.builder()
                .openAiApiKey("ignored")
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .logRequests(logRequests)
                .logStreamingResponses(logResponses)
                .build();
        this.modelName = ensureNotBlank(modelName, "modelName");
        this.temperature = temperature;
        this.topP = topP;
        this.maxTokens = maxTokens;
    }
    @Override
    public void generate(List<ChatMessage> messages,
StreamingResponseHandler<AiMessage> handler) {
```

```
        generate(messages, null, null, handler);
    }
    @Override
    public void generate(List<ChatMessage> messages, List<ToolSpecification>
toolSpecifications, StreamingResponseHandler<AiMessage> handler) {
        generate(messages, toolSpecifications, null, handler);
    }
    @Override
    public void generate(List<ChatMessage> messages, ToolSpecification
toolSpecification, StreamingResponseHandler<AiMessage> handler) {
        generate(messages, singletonList(toolSpecification),
toolSpecification, handler);
    }
    private void generate(List<ChatMessage> messages,
                          List<ToolSpecification> toolSpecifications,
                          ToolSpecification toolThatMustBeExecuted,
                          StreamingResponseHandler<AiMessage> handler
    ) {
        ChatCompletionRequest.Builder requestBuilder =
ChatCompletionRequest.builder()
                .stream(true)
                .model(modelName)
                .messages(toOpenAiMessages(messages))
                .temperature(temperature)
                .topP(topP)
                .maxTokens(maxTokens);
        if (toolSpecifications != null && !toolSpecifications.isEmpty()) {
            requestBuilder.functions(toFunctions(toolSpecifications));
        }
        if (toolThatMustBeExecuted != null) {
            requestBuilder.functionCall(toolThatMustBeExecuted.name());
        }
        ChatCompletionRequest request = requestBuilder.build();
        OpenAiStreamingResponseBuilder responseBuilder = new
OpenAiStreamingResponseBuilder(0);
        client.chatCompletion(request)
                .onPartialResponse(partialResponse -> {
                    responseBuilder.append(partialResponse);
                    handle(partialResponse, handler);
                })
                .onComplete(() -> {
                    Response<AiMessage> response = responseBuilder.build();
                    handler.onComplete(response);
                })
                .onError(handler::onError)
                .execute();
    }
    private static void handle(ChatCompletionResponse partialResponse,
                               StreamingResponseHandler<AiMessage> handler) {
        List<ChatCompletionChoice> choices = partialResponse.choices();
        if (choices == null || choices.isEmpty()) {
            return;
        }
        Delta delta = choices.get(0).delta();
        String content = delta.content();
        if (content != null) {
            handler.onNext(content);
        }
    }
}
```

langchain4j-local-ai\src\main\java\dev\langchain4j\model\localai\LocalAiStreamingLanguageModel.java

```
package dev.langchain4j.model.localai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.completion.CompletionRequest;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.language.StreamingLanguageModel;
import dev.langchain4j.model.openai.OpenAiStreamingResponseBuilder;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.time.Duration;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static java.time.Duration.ofSeconds;
public class LocalAiStreamingLanguageModel implements StreamingLanguageModel {
    private final OpenAiClient client;
    private final String modelName;
    private final Double temperature;
    private final Double topP;
    private final Integer maxTokens;
    @Builder
    public LocalAiStreamingLanguageModel(String baseUrl,
                                         String modelName,
                                         Double temperature,
                                         Double topP,
                                         Integer maxTokens,
                                         Duration timeout,
                                         Boolean logRequests,
                                         Boolean logResponses) {
        temperature = temperature == null ? 0.7 : temperature;
        timeout = timeout == null ? ofSeconds(60) : timeout;
        this.client = OpenAiClient.builder()
                .openAiApiKey("ignored")
                .baseUrl(ensureNotBlank(baseUrl, "baseUrl"))
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .logRequests(logRequests)
                .logStreamingResponses(logResponses)
                .build();
        this.modelName = ensureNotBlank(modelName, "modelName");
        this.temperature = temperature;
        this.topP = topP;
        this.maxTokens = maxTokens;
    }
    @Override
    public void generate(String prompt, StreamingResponseHandler<String>
handler) {
        CompletionRequest request = CompletionRequest.builder()
                .model(modelName)
                .prompt(prompt)
                .temperature(temperature)
                .topP(topP)
                .maxTokens(maxTokens)
                .build();
        OpenAiStreamingResponseBuilder responseBuilder = new
OpenAiStreamingResponseBuilder(0);
        client.completion(request)
```

```
                .onPartialResponse(partialResponse -> {
                    responseBuilder.append(partialResponse);
                    String token = partialResponse.text();
                    if (token != null) {
                        handler.onNext(token);
                    }
                })
                .onComplete(() -> {
                    Response<AiMessage> response = responseBuilder.build();
                    handler.onComplete(Response.from(
                            response.content().text(),
                            response.tokenUsage(),
                            response.finishReason()
                    ));
                })
                .onError(handler::onError)
                .execute();
    }
}
```

langchain4j-local-ai\src\test\java\dev\langchain4j\model\localai\LocalAiChatModelIT.java

```
package dev.langchain4j.model.localai;
import dev.langchain4j.model.chat.ChatLanguageModel;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class LocalAiChatModelIT {
    @Test
    @Disabled("until we host LocalAI instance somewhere")
    void should_send_user_message_and_return_answer() {
        ChatLanguageModel model = LocalAiChatModel.builder()
                .baseUrl("http://localhost:8080")
                .modelName("ggml-gpt4all-j")
                .maxTokens(3)
                .logRequests(true)
                .logResponses(true)
                .build();
        String answer = model.generate("Say 'hello'");
        assertThat(answer).containsIgnoringCase("hello");
        System.out.println(answer);
    }
}
```

langchain4j-local-ai\src\test\java\dev\langchain4j\model\localai\LocalAiEmbeddingModelIT.java

```
package dev.langchain4j.model.localai;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.model.embedding.EmbeddingModel;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class LocalAiEmbeddingModelIT {
    @Test
    @Disabled("until we host LocalAI instance somewhere")
    void should_embed() {
        EmbeddingModel model = LocalAiEmbeddingModel.builder()
                .baseUrl("http://localhost:8080")
                .modelName("ggml-model-q4_0")
                .logRequests(true)
                .logResponses(true)
                .build();
        Embedding embedding = model.embed("hello").content();
        assertThat(embedding.vector()).hasSize(384);
    }
}
```

langchain4j-local-ai\src\test\java\dev\langchain4j\model\localai\LocalAiLanguageModelIT.java

```
package dev.langchain4j.model.localai;
import dev.langchain4j.model.language.LanguageModel;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class LocalAiLanguageModelIT {
    @Test
    @Disabled("until we host LocalAI instance somewhere")
    void should_send_prompt_and_return_answer() {
        LanguageModel model = LocalAiLanguageModel.builder()
                .baseUrl("http://localhost:8080")
                .modelName("ggml-gpt4all-j")
                .maxTokens(3)
                .logRequests(true)
                .logResponses(true)
                .build();
        String answer = model.generate("Say 'hello'").content();
        assertThat(answer).containsIgnoringCase("hello");
        System.out.println(answer);
    }
}
```

langchain4j-local-ai\src\test\java\dev\langchain4j\model\localai\LocalAiStreamingChatModelIT.java

```
package dev.langchain4j.model.localai;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import java.util.concurrent.CompletableFuture;
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.assertj.core.api.Assertions.assertThat;
class LocalAiStreamingChatModelIT {
    @Test
    @Disabled("until we host LocalAI instance somewhere")
    void should_stream_answer() throws Exception {
        StreamingChatLanguageModel model = LocalAiStreamingChatModel.builder()
                .baseUrl("http://localhost:8080")
                .modelName("ggml-gpt4all-j")
                .maxTokens(3)
                .logRequests(true)
                .logResponses(true)
                .build();
        StringBuilder answerBuilder = new StringBuilder();
        CompletableFuture<String> futureAnswer = new CompletableFuture<>();
        model.generate("Say 'hello'", new
StreamingResponseHandler<AiMessage>() {
            @Override
            public void onNext(String token) {
                answerBuilder.append(token);
            }
            @Override
            public void onComplete(Response<AiMessage> response) {
                futureAnswer.complete(answerBuilder.toString());
            }
            @Override
            public void onError(Throwable error) {
                futureAnswer.completeExceptionally(error);
            }
        });
        String answer = futureAnswer.get(30, SECONDS);
        assertThat(answer).containsIgnoringCase("hello");
    }
}
```

langchain4j-local-ai\src\test\java\dev\langchain4j\model\localai\LocalAiStreamingLanguageModelIT.java

```
package dev.langchain4j.model.localai;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.language.StreamingLanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import java.util.concurrent.CompletableFuture;
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.assertj.core.api.Assertions.assertThat;
class LocalAiStreamingLanguageModelIT {
    @Test
    @Disabled("until we host LocalAI instance somewhere")
    void should_stream_answer() throws Exception {
        StreamingLanguageModel model = LocalAiStreamingLanguageModel.builder()
                .baseUrl("http://localhost:8080")
                .modelName("ggml-gpt4all-j")
                .maxTokens(3)
                .logRequests(true)
                .logResponses(true)
                .build();
        StringBuilder answerBuilder = new StringBuilder();
        CompletableFuture<String> futureAnswer = new CompletableFuture<>();
        model.generate("Say 'hello'", new StreamingResponseHandler<String>() {
            @Override
            public void onNext(String token) {
                answerBuilder.append(token);
            }
            @Override
            public void onComplete(Response<String> response) {
                futureAnswer.complete(answerBuilder.toString());
            }
            @Override
            public void onError(Throwable error) {
                futureAnswer.completeExceptionally(error);
            }
        });
        String answer = futureAnswer.get(30, SECONDS);
        assertThat(answer).containsIgnoringCase("hello");
    }
}
```

langchain4j-milvus\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-milvus</artifactId>
    <name>LangChain4j integration with Milvus</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${parent.version}</version>
        </dependency>
        <dependency>
            <groupId>io.milvus</groupId>
            <artifactId>milvus-sdk-java</artifactId>
            <version>2.3.1</version>
            <exclusions>
                <!-- due to CVE-2022-41915 vulnerability -->
                <exclusion>
                    <groupId>io.netty</groupId>
                    <artifactId>netty-codec</artifactId>
                </exclusion>
                <!-- due to CVE-2022-42889 vulnerability -->
                <exclusion>
                    <groupId>org.apache.commons</groupId>
                    <artifactId>commons-text</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-codec</artifactId>
            <version>${netty.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-text</artifactId>
            <version>1.10.0</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
```

```
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

langchain4j-milvus\src\main\java\dev\langchain4j\store\embedding\milvus\CollectionOperationsExecutor.java

```
package dev.langchain4j.store.embedding.milvus;
import io.milvus.client.MilvusServiceClient;
import io.milvus.common.clientenum.ConsistencyLevelEnum;
import io.milvus.grpc.FlushResponse;
import io.milvus.grpc.MutationResult;
import io.milvus.grpc.QueryResults;
import io.milvus.grpc.SearchResults;
import io.milvus.param.IndexType;
import io.milvus.param.MetricType;
import io.milvus.param.R;
import io.milvus.param.RpcStatus;
import io.milvus.param.collection.*;
import io.milvus.param.dml.InsertParam;
import io.milvus.param.dml.QueryParam;
import io.milvus.param.dml.SearchParam;
import io.milvus.param.index.CreateIndexParam;
import io.milvus.response.QueryResultsWrapper;
import io.milvus.response.SearchResultsWrapper;
import java.util.List;
import static
dev.langchain4j.store.embedding.milvus.CollectionRequestBuilder.*;
import static dev.langchain4j.store.embedding.milvus.MilvusEmbeddingStore.*;
import static io.milvus.grpc.DataType.FloatVector;
import static io.milvus.grpc.DataType.VarChar;
class CollectionOperationsExecutor {
    static void flush(MilvusServiceClient milvusClient, String
collectionName) {
        FlushParam request = buildFlushRequest(collectionName);
        R<FlushResponse> response = milvusClient.flush(request);
        checkResponseNotFailed(response);
    }
    static boolean hasCollection(MilvusServiceClient milvusClient, String
collectionName) {
        HasCollectionParam request =
buildHasCollectionRequest(collectionName);
        R<Boolean> response = milvusClient.hasCollection(request);
        checkResponseNotFailed(response);
        return response.getData();
    }
    static void createCollection(MilvusServiceClient milvusClient, String
collectionName, int dimension) {
        CreateCollectionParam request = CreateCollectionParam.newBuilder()
                .withCollectionName(collectionName)
                .addFieldType(FieldType.newBuilder()
                        .withName(ID_FIELD_NAME)
                        .withDataType(VarChar)
                        .withMaxLength(36)
                        .withPrimaryKey(true)
                        .withAutoID(false)
                        .build())
                .addFieldType(FieldType.newBuilder()
                        .withName(TEXT_FIELD_NAME)
                        .withDataType(VarChar)
                        .withMaxLength(65535)
                        .build())
                .addFieldType(FieldType.newBuilder()
                        .withName(VECTOR_FIELD_NAME)
```

```
                        .withDataType(FloatVector)
                        .withDimension(dimension)
                        .build())
                .build();
        R<RpcStatus> response = milvusClient.createCollection(request);
        checkResponseNotFailed(response);
    }
    static void createIndex(MilvusServiceClient milvusClient,
                            String collectionName,
                            IndexType indexType,
                            MetricType metricType) {
        CreateIndexParam request = CreateIndexParam.newBuilder()
                .withCollectionName(collectionName)
                .withFieldName(VECTOR_FIELD_NAME)
                .withIndexType(indexType)
                .withMetricType(metricType)
                .build();
        R<RpcStatus> response = milvusClient.createIndex(request);
        checkResponseNotFailed(response);
    }
    static void insert(MilvusServiceClient milvusClient, String
collectionName, List<InsertParam.Field> fields) {
        InsertParam request = buildInsertRequest(collectionName, fields);
        R<MutationResult> response = milvusClient.insert(request);
        checkResponseNotFailed(response);
    }
    static void loadCollectionInMemory(MilvusServiceClient milvusClient,
String collectionName) {
        LoadCollectionParam request =
buildLoadCollectionInMemoryRequest(collectionName);
        R<RpcStatus> response = milvusClient.loadCollection(request);
        checkResponseNotFailed(response);
    }
    static SearchResultsWrapper search(MilvusServiceClient milvusClient,
SearchParam searchRequest) {
        R<SearchResults> response = milvusClient.search(searchRequest);
        checkResponseNotFailed(response);
        return new SearchResultsWrapper(response.getData().getResults());
    }
    static QueryResultsWrapper queryForVectors(MilvusServiceClient
milvusClient,
                                               String collectionName,
                                               List<String> rowIds,
                                               ConsistencyLevelEnum
consistencyLevel) {
        QueryParam request = buildQueryRequest(collectionName, rowIds,
consistencyLevel);
        R<QueryResults> response = milvusClient.query(request);
        checkResponseNotFailed(response);
        return new QueryResultsWrapper(response.getData());
    }
    private static <T> void checkResponseNotFailed(R<T> response) {
        if (response == null) {
            throw new RequestToMilvusFailedException("Request to Milvus DB
failed. Response is null");
        } else if (response.getStatus() != R.Status.Success.getCode()) {
            String message = String.format("Request to Milvus DB failed.
Response status:'%d'.%n", response.getStatus());
            throw new RequestToMilvusFailedException(message,
response.getException());
        }
```

```
    }
}
```

langchain4j-milvus\src\main\java\dev\langchain4j\store\embedding\milvus\CollectionRequestBuilder.java

```
package dev.langchain4j.store.embedding.milvus;
import io.milvus.common.clientenum.ConsistencyLevelEnum;
import io.milvus.param.MetricType;
import io.milvus.param.collection.FlushParam;
import io.milvus.param.collection.HasCollectionParam;
import io.milvus.param.collection.LoadCollectionParam;
import io.milvus.param.dml.InsertParam;
import io.milvus.param.dml.QueryParam;
import io.milvus.param.dml.SearchParam;
import java.util.List;
import static dev.langchain4j.store.embedding.milvus.MilvusEmbeddingStore.*;
import static java.lang.String.format;
import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.joining;
class CollectionRequestBuilder {
    static FlushParam buildFlushRequest(String collectionName) {
        return FlushParam.newBuilder()
                .withCollectionNames(singletonList(collectionName))
                .build();
    }
    static HasCollectionParam buildHasCollectionRequest(String
collectionName) {
        return HasCollectionParam.newBuilder()
                .withCollectionName(collectionName)
                .build();
    }
    static InsertParam buildInsertRequest(String collectionName,
List<InsertParam.Field> fields) {
        return InsertParam.newBuilder()
                .withCollectionName(collectionName)
                .withFields(fields)
                .build();
    }
    static LoadCollectionParam buildLoadCollectionInMemoryRequest(String
collectionName) {
        return LoadCollectionParam.newBuilder()
                .withCollectionName(collectionName)
                .build();
    }
    static SearchParam buildSearchRequest(String collectionName,
                                          List<Float> vector,
                                          int maxResults,
                                          MetricType metricType,
                                          ConsistencyLevelEnum
consistencyLevel) {
        return SearchParam.newBuilder()
                .withCollectionName(collectionName)
                .withVectors(singletonList(vector))
                .withVectorFieldName(VECTOR_FIELD_NAME)
                .withTopK(maxResults)
                .withMetricType(metricType)
                .withConsistencyLevel(consistencyLevel)
                .withOutFields(asList(ID_FIELD_NAME, TEXT_FIELD_NAME))
                .build();
    }
    static QueryParam buildQueryRequest(String collectionName,
```

```
                                        List<String> rowIds,
                                        ConsistencyLevelEnum
consistencyLevel) {
        return QueryParam.newBuilder()
                .withCollectionName(collectionName)
                .withExpr(buildQueryExpression(rowIds))
                .withConsistencyLevel(consistencyLevel)
                .withOutFields(singletonList(VECTOR_FIELD_NAME))
                .build();
    }
    private static String buildQueryExpression(List<String> rowIds) {
        return rowIds.stream()
                .map(id -> format("%s == '%s'", ID_FIELD_NAME, id))
                .collect(joining(" || "));
    }
}
```

langchain4j-milvus\src\main\java\dev\langchain4j\store\embedding\milvus\Generator.java

```
package dev.langchain4j.store.embedding.milvus;
import dev.langchain4j.internal.Utils;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
class Generator {
    static List<String> generateRandomIds(int size) {
        List<String> ids = new ArrayList<>();
        for (int i = 0; i < size; i++) {
            ids.add(Utils.randomUUID());
        }
        return ids;
    }
    static List<String> generateEmptyScalars(int size) {
        String[] arr = new String[size];
        Arrays.fill(arr, "");
        return Arrays.asList(arr);
    }
}
```

langchain4j-milvus\src\main\java\dev\langchain4j\store\embedding\milvus\Mapper.java

```
package dev.langchain4j.store.embedding.milvus;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.RelevanceScore;
import io.milvus.client.MilvusServiceClient;
import io.milvus.common.clientenum.ConsistencyLevelEnum;
import io.milvus.response.QueryResultsWrapper;
import io.milvus.response.SearchResultsWrapper;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static dev.langchain4j.internal.Utils.isNullOrBlank;
import static dev.langchain4j.store.embedding.milvus.CollectionOperationsExecu
tor.queryForVectors;
import static
dev.langchain4j.store.embedding.milvus.Generator.generateEmptyScalars;
import static dev.langchain4j.store.embedding.milvus.MilvusEmbeddingStore.*;
import static java.util.stream.Collectors.toList;
class Mapper {
    static List<List<Float>> toVectors(List<Embedding> embeddings) {
        return embeddings.stream()
                .map(Embedding::vectorAsList)
                .collect(toList());
    }
    static List<String> toScalars(List<TextSegment> textSegments, int size) {
        boolean noScalars = textSegments == null || textSegments.isEmpty();
        return noScalars ? generateEmptyScalars(size) :
textSegmentsToScalars(textSegments);
    }
    static List<String> textSegmentsToScalars(List<TextSegment> textSegments)
{
        return textSegments.stream()
                .map(TextSegment::text)
                .collect(toList());
    }
    static List<EmbeddingMatch<TextSegment>>
toEmbeddingMatches(MilvusServiceClient milvusClient,

SearchResultsWrapper resultsWrapper,
                                                                    String
collectionName,

ConsistencyLevelEnum consistencyLevel,
                                                                    boolean
queryForVectorOnSearch) {
        List<EmbeddingMatch<TextSegment>> matches = new ArrayList<>();
        List<String> rowIds = (List<String>)
resultsWrapper.getFieldWrapper(ID_FIELD_NAME).getFieldData();
        Map<String, Embedding> idToEmbedding = new HashMap<>();
        if (queryForVectorOnSearch) {
            idToEmbedding.putAll(queryEmbeddings(milvusClient,
collectionName, rowIds, consistencyLevel));
        }
        for (int i = 0; i < resultsWrapper.getRowRecords().size(); i++) {
            double score = resultsWrapper.getIDScore(0).get(i).getScore();
```

```
            String rowId = resultsWrapper.getIDScore(0).get(i).getStrID();
            Embedding embedding = idToEmbedding.get(rowId);
            String text =
String.valueOf(resultsWrapper.getFieldData(TEXT_FIELD_NAME, 0).get(i));
            TextSegment textSegment = isNullOrBlank(text) ? null :
TextSegment.from(text);
            EmbeddingMatch<TextSegment> embeddingMatch = new EmbeddingMatch<>(
                    RelevanceScore.fromCosineSimilarity(score),
                    rowId,
                    embedding,
                    textSegment
            );
            matches.add(embeddingMatch);
        }
        return matches;
    }
    private static Map<String, Embedding> queryEmbeddings(MilvusServiceClient
milvusClient,
                                                          String
collectionName,
                                                          List<String> rowIds,

ConsistencyLevelEnum consistencyLevel) {
        QueryResultsWrapper queryResultsWrapper = queryForVectors(
                milvusClient,
                collectionName,
                rowIds,
                consistencyLevel
        );
        Map<String, Embedding> idToEmbedding = new HashMap<>();
        for (QueryResultsWrapper.RowRecord row :
queryResultsWrapper.getRowRecords()) {
            String id = row.get(ID_FIELD_NAME).toString();
            List<Float> vector = (List<Float>) row.get(VECTOR_FIELD_NAME);
            idToEmbedding.put(id, Embedding.from(vector));
        }
        return idToEmbedding;
    }
}
```

langchain4j-milvus\src\main\java\dev\langchain4j\store\embedding\milvus\MilvusEmbeddingStore.java

```
package dev.langchain4j.store.embedding.milvus;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
import static
dev.langchain4j.store.embedding.milvus.CollectionOperationsExecutor.*;
import static dev.langchain4j.store.embedding.milvus.CollectionRequestBuilder.
buildSearchRequest;
import static
dev.langchain4j.store.embedding.milvus.Generator.generateRandomIds;
import static dev.langchain4j.store.embedding.milvus.Mapper.*;
import static io.milvus.common.clientenum.ConsistencyLevelEnum.EVENTUALLY;
import static io.milvus.param.IndexType.FLAT;
import static io.milvus.param.MetricType.COSINE;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.toList;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.internal.Utils;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import io.milvus.client.MilvusServiceClient;
import io.milvus.common.clientenum.ConsistencyLevelEnum;
import io.milvus.param.ConnectParam;
import io.milvus.param.IndexType;
import io.milvus.param.MetricType;
import io.milvus.param.dml.InsertParam;
import io.milvus.param.dml.SearchParam;
import io.milvus.response.SearchResultsWrapper;
import java.util.ArrayList;
import java.util.List;
/**
 * Represents an <a href="https://milvus.io/">Milvus</a> index as an
embedding store.
 * Does not support storing {@link dev.langchain4j.data.document.Metadata}
yet.
 */
public class MilvusEmbeddingStore implements EmbeddingStore<TextSegment> {
  static final String ID_FIELD_NAME = "id";
  static final String TEXT_FIELD_NAME = "text";
  static final String VECTOR_FIELD_NAME = "vector";
  private final MilvusServiceClient milvusClient;
  private final String collectionName;
  private final MetricType metricType;
  private final ConsistencyLevelEnum consistencyLevel;
  private final boolean retrieveEmbeddingsOnSearch;
  public MilvusEmbeddingStore(
    String host,
    Integer port,
    String collectionName,
    Integer dimension,
    IndexType indexType,
    MetricType metricType,
    String uri,
    String token,
    String username,
    String password,
    ConsistencyLevelEnum consistencyLevel,
```

```
    Boolean retrieveEmbeddingsOnSearch,
    String databaseName
  ) {
    ConnectParam.Builder connectBuilder = ConnectParam
      .newBuilder()
      .withHost(getOrDefault(host, "localhost"))
      .withPort(getOrDefault(port, 19530))
      .withUri(uri)
      .withToken(token)
      .withAuthorization(username, password);
    if (databaseName != null) {
      connectBuilder.withDatabaseName(databaseName);
    }
    this.milvusClient = new MilvusServiceClient(connectBuilder.build());
    this.collectionName = getOrDefault(collectionName, "default");
    this.metricType = getOrDefault(metricType, COSINE);
    this.consistencyLevel = getOrDefault(consistencyLevel, EVENTUALLY);
    this.retrieveEmbeddingsOnSearch =
getOrDefault(retrieveEmbeddingsOnSearch, false);
    if (!hasCollection(milvusClient, this.collectionName)) {
      createCollection(milvusClient, this.collectionName,
ensureNotNull(dimension, "dimension"));
      createIndex(milvusClient, this.collectionName, getOrDefault(indexType,
FLAT), this.metricType);
    }
  }
  public String add(Embedding embedding) {
    String id = Utils.randomUUID();
    add(id, embedding);
    return id;
  }
  public void add(String id, Embedding embedding) {
    addInternal(id, embedding, null);
  }
  public String add(Embedding embedding, TextSegment textSegment) {
    String id = Utils.randomUUID();
    addInternal(id, embedding, textSegment);
    return id;
  }
  public List<String> addAll(List<Embedding> embeddings) {
    List<String> ids = generateRandomIds(embeddings.size());
    addAllInternal(ids, embeddings, null);
    return ids;
  }
  public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
embedded) {
    List<String> ids = generateRandomIds(embeddings.size());
    addAllInternal(ids, embeddings, embedded);
    return ids;
  }
  public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
referenceEmbedding, int maxResults, double minScore) {
    loadCollectionInMemory(milvusClient, collectionName);
    SearchParam searchRequest = buildSearchRequest(
      collectionName,
      referenceEmbedding.vectorAsList(),
      maxResults,
      metricType,
      consistencyLevel
    );
    SearchResultsWrapper resultsWrapper = search(milvusClient, searchRequest);
```

```
    List<EmbeddingMatch<TextSegment>> matches = toEmbeddingMatches(
      milvusClient,
      resultsWrapper,
      collectionName,
      consistencyLevel,
      retrieveEmbeddingsOnSearch
    );
    return matches.stream().filter(match -> match.score() >=
minScore).collect(toList());
  }
  private void addInternal(String id, Embedding embedding, TextSegment
textSegment) {
    addAllInternal(
      singletonList(id),
      singletonList(embedding),
      textSegment == null ? null : singletonList(textSegment)
    );
  }
  private void addAllInternal(List<String> ids, List<Embedding> embeddings,
List<TextSegment> textSegments) {
    List<InsertParam.Field> fields = new ArrayList<>();
    fields.add(new InsertParam.Field(ID_FIELD_NAME, ids));
    fields.add(new InsertParam.Field(TEXT_FIELD_NAME, toScalars(textSegments,
ids.size())));
    fields.add(new InsertParam.Field(VECTOR_FIELD_NAME,
toVectors(embeddings)));
    insert(milvusClient, collectionName, fields);
    flush(milvusClient, collectionName);
  }
  public static Builder builder() {
    return new Builder();
  }
  public static class Builder {
    private String host;
    private Integer port;
    private String collectionName;
    private Integer dimension;
    private IndexType indexType;
    private MetricType metricType;
    private String uri;
    private String token;
    private String username;
    private String password;
    private ConsistencyLevelEnum consistencyLevel;
    private Boolean retrieveEmbeddingsOnSearch;
    private String databaseName;
    /**
     * @param host The host of the self-managed Milvus instance.
     *             Default value: "localhost".
     * @return builder
     */
    public Builder host(String host) {
      this.host = host;
      return this;
    }
    /**
     * @param port The port of the self-managed Milvus instance.
     *             Default value: 19530.
     * @return builder
     */
    public Builder port(Integer port) {
```

```
      this.port = port;
      return this;
    }
    /**
     * @param collectionName The name of the Milvus collection.
     *                       If there is no such collection yet, it will be
created automatically.
     *                       Default value: "default".
     * @return builder
     */
    public Builder collectionName(String collectionName) {
      this.collectionName = collectionName;
      return this;
    }
    /**
     * @param dimension The dimension of the embedding vector. (e.g. 384)
     *                  Mandatory if a new collection should be created.
     * @return builder
     */
    public Builder dimension(Integer dimension) {
      this.dimension = dimension;
      return this;
    }
    /**
     * @param indexType The type of the index.
     *                  Default value: FLAT.
     * @return builder
     */
    public Builder indexType(IndexType indexType) {
      this.indexType = indexType;
      return this;
    }
    /**
     * @param metricType The type of the metric used for similarity search.
     *                   Default value: COSINE.
     * @return builder
     */
    public Builder metricType(MetricType metricType) {
      this.metricType = metricType;
      return this;
    }
    /**
     * @param uri The URI of the managed Milvus instance. (e.g. "https://
xxx.api.gcp-us-west1.zillizcloud.com")
     * @return builder
     */
    public Builder uri(String uri) {
      this.uri = uri;
      return this;
    }
    /**
     * @param token The token (API key) of the managed Milvus instance.
     * @return builder
     */
    public Builder token(String token) {
      this.token = token;
      return this;
    }
    /**
     * @param username The username. See details <a href="https://milvus.io/
docs/authenticate.md">here</a>.
```

```
     * @return builder
     */
    public Builder username(String username) {
      this.username = username;
      return this;
    }
    /**
     * @param password The password. See details <a href="https://milvus.io/
docs/authenticate.md">here</a>.
     * @return builder
     */
    public Builder password(String password) {
      this.password = password;
      return this;
    }
    /**
     * @param consistencyLevel The consistency level used by Milvus.
     *                         Default value: EVENTUALLY.
     * @return builder
     */
    public Builder consistencyLevel(ConsistencyLevelEnum consistencyLevel) {
      this.consistencyLevel = consistencyLevel;
      return this;
    }
    /**
     * @param retrieveEmbeddingsOnSearch During a similarity search in Milvus
(when calling findRelevant()),
     *                                   the embedding itself is not
retrieved.
     *                                   To retrieve the embedding, an
additional query is required.
     *                                   Setting this parameter to "true"
will ensure that embedding is retrieved.
     *                                   Be aware that this will impact the
performance of the search.
     *                                   Default value: false.
     * @return builder
     */
    public Builder retrieveEmbeddingsOnSearch(Boolean
retrieveEmbeddingsOnSearch) {
      this.retrieveEmbeddingsOnSearch = retrieveEmbeddingsOnSearch;
      return this;
    }
    /**
     * @param databaseName Milvus name of database.
     *                     Default value: null. In this case default Milvus
database name will be used.
     * @return builder
     */
    public Builder databaseName(String databaseName) {
      this.databaseName = databaseName;
      return this;
    }
    public MilvusEmbeddingStore build() {
      return new MilvusEmbeddingStore(
        host,
        port,
        collectionName,
        dimension,
        indexType,
        metricType,
```

```
        uri,
        token,
        username,
        password,
        consistencyLevel,
        retrieveEmbeddingsOnSearch,
        databaseName
      );
    }
  }
}
```

langchain4j-milvus\src\main\java\dev\langchain4j\store\embedding\milvus\RequestToMilvusFailedException.java

```
package dev.langchain4j.store.embedding.milvus;
class RequestToMilvusFailedException extends RuntimeException {
    public RequestToMilvusFailedException() {
        super();
    }
    public RequestToMilvusFailedException(String message) {
        super(message);
    }
    public RequestToMilvusFailedException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

langchain4j-milvus\src\test\java\dev\langchain4j\store\embedding\milvus\MilvusEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.milvus;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static io.milvus.param.MetricType.IP;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
@Disabled("needs Milvus running locally")
class MilvusEmbeddingStoreTest {
    /**
     * First run Milvus locally:
     * Run "docker compose up -d" inside "langchain4j-milvus/src/test/
resources" directory.
     * If you want to create a fresh Milvus instance, don't forget to remove
"langchain4j-milvus/src/test/resources/volumes" directory.
     */
    EmbeddingStore<TextSegment> embeddingStore =
MilvusEmbeddingStore.builder()
            .host("localhost")
            .port(19530)
            .collectionName("collection_" + randomUUID().replace("-", ""))
            .dimension(384)
            .build();
    EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    @Test
    void should_add_embedding() {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isNull();
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
```

```
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isNull();
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isNull();
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isNull();
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isNull();
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
        );
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
```

```
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isNull();
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isNull();
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
```

```
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
referenceEmbedding)),
                withPercentage(1)
        );
    }
    @Test
    void should_retrieve_embeddings_when_searching() {
        EmbeddingStore<TextSegment> embeddingStore =
MilvusEmbeddingStore.builder()
                .host("localhost")
                .port(19530)
                .collectionName("collection_" + randomUUID().replace("-", ""))
                .dimension(384)
                .retrieveEmbeddingsOnSearch(true)
                .build();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_use_cloud_instance() {
        EmbeddingStore<TextSegment> embeddingStore =
MilvusEmbeddingStore.builder()
                .uri("https://in03-d11858f677102da.api.gcp-us-
west1.zillizcloud.com")
                .token(System.getenv("MILVUS_API_KEY"))
                .collectionName("test")
                .dimension(384)
                .metricType(IP) // COSINE is not supported at the moment
                .build();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
```

```
            assertThat(match.embeddingId()).isEqualTo(id);
            assertThat(match.embedding()).isNull();
            assertThat(match.embedded()).isNull();
        }
    }
```

langchain4j-open-ai\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-open-ai</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with OpenAI</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.ai4j</groupId>
            <artifactId>openai4j</artifactId>
        </dependency>
        <dependency>
            <groupId>com.knuddels</groupId>
            <artifactId>jtokkit</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
```

```
</project>
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\InternalOpenAiHelper.java

```
package dev.langchain4j.model.openai;
import dev.ai4j.openai4j.chat.*;
import dev.ai4j.openai4j.shared.Usage;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import dev.langchain4j.agent.tool.ToolParameters;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.*;
import dev.langchain4j.model.output.FinishReason;
import dev.langchain4j.model.output.TokenUsage;
import java.util.Collection;
import java.util.List;
import static dev.ai4j.openai4j.chat.Role.*;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static dev.langchain4j.model.output.FinishReason.*;
import static java.util.stream.Collectors.toList;
public class InternalOpenAiHelper {
    static final String OPENAI_URL = "https://api.openai.com/v1";
    static final String OPENAI_DEMO_API_KEY = "demo";
    static final String OPENAI_DEMO_URL = "http://langchain4j.dev/demo/openai/
v1";
    public static List<Message> toOpenAiMessages(List<ChatMessage> messages) {
        return messages.stream()
                .map(InternalOpenAiHelper::toOpenAiMessage)
                .collect(toList());
    }
    public static Message toOpenAiMessage(ChatMessage message) {
        return Message.builder()
                .role(roleFrom(message))
                .name(nameFrom(message))
                .content(message.text())
                .functionCall(functionCallFrom(message))
                .build();
    }
    private static String nameFrom(ChatMessage message) {
        if (message instanceof UserMessage) {
            return ((UserMessage) message).name();
        }
        if (message instanceof ToolExecutionResultMessage) {
            return ((ToolExecutionResultMessage) message).toolName();
        }
        return null;
    }
    private static FunctionCall functionCallFrom(ChatMessage message) {
        if (message instanceof AiMessage) {
            AiMessage aiMessage = (AiMessage) message;
            if (aiMessage.toolExecutionRequest() != null) {
                return FunctionCall.builder()
                        .name(aiMessage.toolExecutionRequest().name())
                        .arguments(aiMessage.toolExecutionRequest().arguments(
))
                        .build();
            }
        }
        return null;
    }
    public static Role roleFrom(ChatMessage message) {
        if (message instanceof AiMessage) {
```

```
            return ASSISTANT;
        } else if (message instanceof ToolExecutionResultMessage) {
            return FUNCTION;
        } else if (message instanceof SystemMessage) {
            return SYSTEM;
        } else {
            return USER;
        }
    }
    public static List<Function> toFunctions(Collection<ToolSpecification>
toolSpecifications) {
        return toolSpecifications.stream()
                .map(InternalOpenAiHelper::toFunction)
                .collect(toList());
    }
    private static Function toFunction(ToolSpecification toolSpecification) {
        return Function.builder()
                .name(toolSpecification.name())
                .description(toolSpecification.description())
                .parameters(toOpenAiParameters(toolSpecification.parameters())
)
                .build();
    }
    private static dev.ai4j.openai4j.chat.Parameters
toOpenAiParameters(ToolParameters toolParameters) {
        if (toolParameters == null) {
            return dev.ai4j.openai4j.chat.Parameters.builder().build();
        }
        return dev.ai4j.openai4j.chat.Parameters.builder()
                .properties(toolParameters.properties())
                .required(toolParameters.required())
                .build();
    }
    public static AiMessage aiMessageFrom(ChatCompletionResponse response) {
        if (response.content() != null) {
            return aiMessage(response.content());
        } else {
            FunctionCall functionCall =
response.choices().get(0).message().functionCall();
            ToolExecutionRequest toolExecutionRequest =
ToolExecutionRequest.builder()
                    .name(functionCall.name())
                    .arguments(functionCall.arguments())
                    .build();
            return aiMessage(toolExecutionRequest);
        }
    }
    public static TokenUsage tokenUsageFrom(Usage openAiUsage) {
        if (openAiUsage == null) {
            return null;
        }
        return new TokenUsage(
                openAiUsage.promptTokens(),
                openAiUsage.completionTokens(),
                openAiUsage.totalTokens()
        );
    }
    public static FinishReason finishReasonFrom(String openAiFinishReason) {
        switch (openAiFinishReason) {
            case "stop":
                return STOP;
```

```
            case "length":
                return LENGTH;
            case "function_call":
                return TOOL_EXECUTION;
            case "content_filter":
                return CONTENT_FILTER;
            default:
                return null;
        }
    }
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiChatModel.java

```
package dev.langchain4j.model.openai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.chat.ChatCompletionRequest;
import dev.ai4j.openai4j.chat.ChatCompletionResponse;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.chat.TokenCountEstimator;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.net.Proxy;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.*;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static java.time.Duration.ofSeconds;
import static java.util.Collections.singletonList;
/**
 * Represents an OpenAI language model with a chat completion interface, such
as gpt-3.5-turbo and gpt-4.
 * You can find description of parameters <a href="https://
platform.openai.com/docs/api-reference/chat/create">here</a>.
 */
public class OpenAiChatModel implements ChatLanguageModel,
TokenCountEstimator {
    private final OpenAiClient client;
    private final String modelName;
    private final Double temperature;
    private final Double topP;
    private final List<String> stop;
    private final Integer maxTokens;
    private final Double presencePenalty;
    private final Double frequencyPenalty;
    private final Integer maxRetries;
    private final Tokenizer tokenizer;
    @Builder
    public OpenAiChatModel(String baseUrl,
                           String apiKey,
                           String modelName,
                           Double temperature,
                           Double topP,
                           List<String> stop,
                           Integer maxTokens,
                           Double presencePenalty,
                           Double frequencyPenalty,
                           Duration timeout,
                           Integer maxRetries,
                           Proxy proxy,
                           Boolean logRequests,
                           Boolean logResponses,
                           Tokenizer tokenizer) {
        baseUrl = getOrDefault(baseUrl, OPENAI_URL);
        if (OPENAI_DEMO_API_KEY.equals(apiKey)) {
```

```
            baseUrl = OPENAI_DEMO_URL;
        }
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .openAiApiKey(apiKey)
                .baseUrl(baseUrl)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.modelName = getOrDefault(modelName, GPT_3_5_TURBO);
        this.temperature = getOrDefault(temperature, 0.7);
        this.topP = topP;
        this.stop = stop;
        this.maxTokens = maxTokens;
        this.presencePenalty = presencePenalty;
        this.frequencyPenalty = frequencyPenalty;
        this.maxRetries = getOrDefault(maxRetries, 3);
        this.tokenizer = getOrDefault(tokenizer, new
OpenAiTokenizer(this.modelName));
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages) {
        return generate(messages, null, null);
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
List<ToolSpecification> toolSpecifications) {
        return generate(messages, toolSpecifications, null);
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
ToolSpecification toolSpecification) {
        return generate(messages, singletonList(toolSpecification),
toolSpecification);
    }
    private Response<AiMessage> generate(List<ChatMessage> messages,
                                         List<ToolSpecification>
toolSpecifications,
                                         ToolSpecification
toolThatMustBeExecuted
    ) {
        ChatCompletionRequest.Builder requestBuilder =
ChatCompletionRequest.builder()
                .model(modelName)
                .messages(toOpenAiMessages(messages))
                .temperature(temperature)
                .topP(topP)
                .stop(stop)
                .maxTokens(maxTokens)
                .presencePenalty(presencePenalty)
                .frequencyPenalty(frequencyPenalty);
        if (toolSpecifications != null && !toolSpecifications.isEmpty()) {
            requestBuilder.functions(toFunctions(toolSpecifications));
        }
        if (toolThatMustBeExecuted != null) {
            requestBuilder.functionCall(toolThatMustBeExecuted.name());
```

```
        }
        ChatCompletionRequest request = requestBuilder.build();
        ChatCompletionResponse response = withRetry(() ->
client.chatCompletion(request).execute(), maxRetries);
        return Response.from(
                aiMessageFrom(response),
                tokenUsageFrom(response.usage()),
                finishReasonFrom(response.choices().get(0).finishReason())
        );
    }
    @Override
    public int estimateTokenCount(List<ChatMessage> messages) {
        return tokenizer.estimateTokenCountInMessages(messages);
    }
    public static OpenAiChatModel withApiKey(String apiKey) {
        return builder().apiKey(apiKey).build();
    }
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiEmbeddingModel.java

```
package dev.langchain4j.model.openai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.embedding.EmbeddingRequest;
import dev.ai4j.openai4j.embedding.EmbeddingResponse;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.embedding.TokenCountEstimator;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.net.Proxy;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.*;
import static
dev.langchain4j.model.openai.OpenAiModelName.TEXT_EMBEDDING_ADA_002;
import static java.time.Duration.ofSeconds;
import static java.util.stream.Collectors.toList;
/**
 * Represents an OpenAI embedding model, such as text-embedding-ada-002.
 */
public class OpenAiEmbeddingModel implements EmbeddingModel,
TokenCountEstimator {
    private final OpenAiClient client;
    private final String modelName;
    private final Integer maxRetries;
    private final Tokenizer tokenizer;
    @Builder
    public OpenAiEmbeddingModel(String baseUrl,
                                String apiKey,
                                String modelName,
                                Duration timeout,
                                Integer maxRetries,
                                Proxy proxy,
                                Boolean logRequests,
                                Boolean logResponses,
                                Tokenizer tokenizer) {
        baseUrl = getOrDefault(baseUrl, OPENAI_URL);
        if (OPENAI_DEMO_API_KEY.equals(apiKey)) {
            baseUrl = OPENAI_DEMO_URL;
        }
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .openAiApiKey(apiKey)
                .baseUrl(baseUrl)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.modelName = getOrDefault(modelName, TEXT_EMBEDDING_ADA_002);
```

```
        this.maxRetries = getOrDefault(maxRetries, 3);
        this.tokenizer = getOrDefault(tokenizer, new
OpenAiTokenizer(this.modelName));
    }
    @Override
    public Response<List<Embedding>> embedAll(List<TextSegment> textSegments)
{
        List<String> texts = textSegments.stream()
                .map(TextSegment::text)
                .collect(toList());
        return embedTexts(texts);
    }
    private Response<List<Embedding>> embedTexts(List<String> texts) {
        EmbeddingRequest request = EmbeddingRequest.builder()
                .input(texts)
                .model(modelName)
                .build();
        EmbeddingResponse response = withRetry(() ->
client.embedding(request).execute(), maxRetries);
        List<Embedding> embeddings = response.data().stream()
                .map(openAiEmbedding ->
Embedding.from(openAiEmbedding.embedding()))
                .collect(toList());
        return Response.from(
                embeddings,
                tokenUsageFrom(response.usage())
        );
    }
    @Override
    public int estimateTokenCount(String text) {
        return tokenizer.estimateTokenCountInText(text);
    }
    public static OpenAiEmbeddingModel withApiKey(String apiKey) {
        return builder().apiKey(apiKey).build();
    }
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiLanguageModel.java

```
package dev.langchain4j.model.openai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.completion.CompletionChoice;
import dev.ai4j.openai4j.completion.CompletionRequest;
import dev.ai4j.openai4j.completion.CompletionResponse;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.language.TokenCountEstimator;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.net.Proxy;
import java.time.Duration;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.*;
import static dev.langchain4j.model.openai.OpenAiModelName.TEXT_DAVINCI_003;
import static java.time.Duration.ofSeconds;
/**
 * Represents an OpenAI language model with a completion interface, such as text-davinci-003.
 * However, it's recommended to use {@link OpenAiChatModel} instead,
 * as it offers more advanced features like function calling, multi-turn conversations, etc.
 */
public class OpenAiLanguageModel implements LanguageModel, TokenCountEstimator {
    private final OpenAiClient client;
    private final String modelName;
    private final Double temperature;
    private final Integer maxRetries;
    private final Tokenizer tokenizer;
    @Builder
    public OpenAiLanguageModel(String baseUrl,
                               String apiKey,
                               String modelName,
                               Double temperature,
                               Duration timeout,
                               Integer maxRetries,
                               Proxy proxy,
                               Boolean logRequests,
                               Boolean logResponses,
                               Tokenizer tokenizer) {
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .baseUrl(getOrDefault(baseUrl, OPENAI_URL))
                .openAiApiKey(apiKey)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.modelName = getOrDefault(modelName, TEXT_DAVINCI_003);
        this.temperature = getOrDefault(temperature, 0.7);
        this.maxRetries = getOrDefault(maxRetries, 3);
```

```
        this.tokenizer = getOrDefault(tokenizer, new
OpenAiTokenizer(this.modelName));
    }
    @Override
    public Response<String> generate(String prompt) {
        CompletionRequest request = CompletionRequest.builder()
                .model(modelName)
                .prompt(prompt)
                .temperature(temperature)
                .build();
        CompletionResponse response = withRetry(() ->
client.completion(request).execute(), maxRetries);
        CompletionChoice completionChoice = response.choices().get(0);
        return Response.from(
                completionChoice.text(),
                tokenUsageFrom(response.usage()),
                finishReasonFrom(completionChoice.finishReason())
        );
    }
    @Override
    public int estimateTokenCount(String prompt) {
        return tokenizer.estimateTokenCountInText(prompt);
    }
    public static OpenAiLanguageModel withApiKey(String apiKey) {
        return builder().apiKey(apiKey).build();
    }
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiModelName.java

```
package dev.langchain4j.model.openai;
public class OpenAiModelName {
    // Use with OpenAiChatModel and OpenAiStreamingChatModel
    public static final String GPT_3_5_TURBO = "gpt-3.5-turbo"; // alias for
the latest model
    public static final String GPT_3_5_TURBO_0301 = "gpt-3.5-turbo-0301"; //
4k context
    public static final String GPT_3_5_TURBO_0613 = "gpt-3.5-turbo-0613"; //
4k context, functions
    public static final String GPT_3_5_TURBO_16K = "gpt-3.5-turbo-16k"; //
alias for the latest model
    public static final String GPT_3_5_TURBO_16K_0613 = "gpt-3.5-
turbo-16k-0613"; // 16k context, functions
    public static final String GPT_4 = "gpt-4"; // alias for the latest model
    public static final String GPT_4_0314 = "gpt-4-0314"; // 8k context
    public static final String GPT_4_0613 = "gpt-4-0613"; // 8k context,
functions
    public static final String GPT_4_32K = "gpt-4-32k"; // alias for the
latest model
    public static final String GPT_4_32K_0314 = "gpt-4-32k-0314"; // 32k
context
    public static final String GPT_4_32K_0613 = "gpt-4-32k-0613"; // 32k
context, functions
    // Use with OpenAiLanguageModel and OpenAiStreamingLanguageModel
    public static final String TEXT_DAVINCI_003 = "text-davinci-003";
    // Use with OpenAiEmbeddingModel
    public static final String TEXT_EMBEDDING_ADA_002 = "text-embedding-
ada-002";
    // Use with OpenAiModerationModel
    public static final String TEXT_MODERATION_STABLE = "text-moderation-
stable";
    public static final String TEXT_MODERATION_LATEST = "text-moderation-
latest";
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiModerationModel.java

```
package dev.langchain4j.model.openai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.moderation.ModerationRequest;
import dev.ai4j.openai4j.moderation.ModerationResponse;
import dev.ai4j.openai4j.moderation.ModerationResult;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.moderation.Moderation;
import dev.langchain4j.model.moderation.ModerationModel;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.net.Proxy;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.*;
import static
dev.langchain4j.model.openai.OpenAiModelName.TEXT_MODERATION_LATEST;
import static java.time.Duration.ofSeconds;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.toList;
/**
 * Represents an OpenAI moderation model, such as text-moderation-latest.
 */
public class OpenAiModerationModel implements ModerationModel {
    private final OpenAiClient client;
    private final String modelName;
    private final Integer maxRetries;
    @Builder
    public OpenAiModerationModel(String baseUrl,
                                 String apiKey,
                                 String modelName,
                                 Duration timeout,
                                 Integer maxRetries,
                                 Proxy proxy,
                                 Boolean logRequests,
                                 Boolean logResponses) {
        baseUrl = getOrDefault(baseUrl, OPENAI_URL);
        if (OPENAI_DEMO_API_KEY.equals(apiKey)) {
            baseUrl = OPENAI_DEMO_URL;
        }
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .openAiApiKey(apiKey)
                .baseUrl(baseUrl)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logResponses(logResponses)
                .build();
        this.modelName = getOrDefault(modelName, TEXT_MODERATION_LATEST);
        this.maxRetries = getOrDefault(maxRetries, 3);
```

```
    }
    @Override
    public Response<Moderation> moderate(String text) {
        return moderateInternal(singletonList(text));
    }
    private Response<Moderation> moderateInternal(List<String> inputs) {
        ModerationRequest request = ModerationRequest.builder()
                .model(modelName)
                .input(inputs)
                .build();
        ModerationResponse response = withRetry(() ->
client.moderation(request).execute(), maxRetries);
        int i = 0;
        for (ModerationResult moderationResult : response.results()) {
            if (moderationResult.isFlagged()) {
                return Response.from(Moderation.flagged(inputs.get(i)));
            }
            i++;
        }
        return Response.from(Moderation.notFlagged());
    }
    @Override
    public Response<Moderation> moderate(Prompt prompt) {
        return moderate(prompt.text());
    }
    @Override
    public Response<Moderation> moderate(ChatMessage message) {
        return moderate(message.text());
    }
    @Override
    public Response<Moderation> moderate(List<ChatMessage> messages) {
        List<String> inputs = messages.stream()
                .map(ChatMessage::text)
                .collect(toList());
        return moderateInternal(inputs);
    }
    @Override
    public Response<Moderation> moderate(TextSegment textSegment) {
        return moderate(textSegment.text());
    }
    public static OpenAiModerationModel withApiKey(String apiKey) {
        return builder().apiKey(apiKey).build();
    }
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiStreamingChatModel.java

```
package dev.langchain4j.model.openai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.chat.ChatCompletionChoice;
import dev.ai4j.openai4j.chat.ChatCompletionRequest;
import dev.ai4j.openai4j.chat.ChatCompletionResponse;
import dev.ai4j.openai4j.chat.Delta;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.chat.TokenCountEstimator;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.net.Proxy;
import java.time.Duration;
import java.util.List;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.*;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static java.time.Duration.ofSeconds;
import static java.util.Collections.singletonList;
/**
 * Represents an OpenAI language model with a chat completion interface, such
as gpt-3.5-turbo and gpt-4.
 * The model's response is streamed token by token and should be handled with
{@link StreamingResponseHandler}.
 * You can find description of parameters <a href="https://
platform.openai.com/docs/api-reference/chat/create">here</a>.
 */
public class OpenAiStreamingChatModel implements StreamingChatLanguageModel,
TokenCountEstimator {
    private final OpenAiClient client;
    private final String modelName;
    private final Double temperature;
    private final Double topP;
    private final List<String> stop;
    private final Integer maxTokens;
    private final Double presencePenalty;
    private final Double frequencyPenalty;
    private final Tokenizer tokenizer;
    @Builder
    public OpenAiStreamingChatModel(String baseUrl,
                                    String apiKey,
                                    String modelName,
                                    Double temperature,
                                    Double topP,
                                    List<String> stop,
                                    Integer maxTokens,
                                    Double presencePenalty,
                                    Double frequencyPenalty,
                                    Duration timeout,
                                    Proxy proxy,
                                    Boolean logRequests,
                                    Boolean logResponses,
                                    Tokenizer tokenizer) {
```

```
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .baseUrl(getOrDefault(baseUrl, OPENAI_URL))
                .openAiApiKey(apiKey)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logStreamingResponses(logResponses)
                .build();
        this.modelName = getOrDefault(modelName, GPT_3_5_TURBO);
        this.temperature = getOrDefault(temperature, 0.7);
        this.topP = topP;
        this.stop = stop;
        this.maxTokens = maxTokens;
        this.presencePenalty = presencePenalty;
        this.frequencyPenalty = frequencyPenalty;
        this.tokenizer = getOrDefault(tokenizer, new
OpenAiTokenizer(this.modelName));
    }
    @Override
    public void generate(List<ChatMessage> messages,
StreamingResponseHandler<AiMessage> handler) {
        generate(messages, null, null, handler);
    }
    @Override
    public void generate(List<ChatMessage> messages, List<ToolSpecification>
toolSpecifications, StreamingResponseHandler<AiMessage> handler) {
        generate(messages, toolSpecifications, null, handler);
    }
    @Override
    public void generate(List<ChatMessage> messages, ToolSpecification
toolSpecification, StreamingResponseHandler<AiMessage> handler) {
        generate(messages, singletonList(toolSpecification),
toolSpecification, handler);
    }
    private void generate(List<ChatMessage> messages,
                          List<ToolSpecification> toolSpecifications,
                          ToolSpecification toolThatMustBeExecuted,
                          StreamingResponseHandler<AiMessage> handler
    ) {
        ChatCompletionRequest.Builder requestBuilder =
ChatCompletionRequest.builder()
                .stream(true)
                .model(modelName)
                .messages(toOpenAiMessages(messages))
                .temperature(temperature)
                .topP(topP)
                .stop(stop)
                .maxTokens(maxTokens)
                .presencePenalty(presencePenalty)
                .frequencyPenalty(frequencyPenalty);
        int inputTokenCount =
tokenizer.estimateTokenCountInMessages(messages);
        if (toolSpecifications != null && !toolSpecifications.isEmpty()) {
            requestBuilder.functions(toFunctions(toolSpecifications));
            inputTokenCount +=
tokenizer.estimateTokenCountInToolSpecifications(toolSpecifications);
        }
```

```
        if (toolThatMustBeExecuted != null) {
            requestBuilder.functionCall(toolThatMustBeExecuted.name());
            inputTokenCount +=
tokenizer.estimateTokenCountInToolSpecification(toolThatMustBeExecuted);
        }
        ChatCompletionRequest request = requestBuilder.build();
        OpenAiStreamingResponseBuilder responseBuilder = new
OpenAiStreamingResponseBuilder(inputTokenCount);
        client.chatCompletion(request)
                .onPartialResponse(partialResponse -> {
                    responseBuilder.append(partialResponse);
                    handle(partialResponse, handler);
                })
                .onComplete(() -> {
                    Response<AiMessage> response = responseBuilder.build();
                    handler.onComplete(response);
                })
                .onError(handler::onError)
                .execute();
    }
    private static void handle(ChatCompletionResponse partialResponse,
                               StreamingResponseHandler<AiMessage> handler) {
        List<ChatCompletionChoice> choices = partialResponse.choices();
        if (choices == null || choices.isEmpty()) {
            return;
        }
        Delta delta = choices.get(0).delta();
        String content = delta.content();
        if (content != null) {
            handler.onNext(content);
        }
    }
    @Override
    public int estimateTokenCount(List<ChatMessage> messages) {
        return tokenizer.estimateTokenCountInMessages(messages);
    }
    public static OpenAiStreamingChatModel withApiKey(String apiKey) {
        return builder().apiKey(apiKey).build();
    }
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiStreamingLanguageModel.java

```
package dev.langchain4j.model.openai;
import dev.ai4j.openai4j.OpenAiClient;
import dev.ai4j.openai4j.completion.CompletionRequest;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.language.StreamingLanguageModel;
import dev.langchain4j.model.language.TokenCountEstimator;
import dev.langchain4j.model.output.Response;
import lombok.Builder;
import java.net.Proxy;
import java.time.Duration;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.OPENAI_URL;
import static dev.langchain4j.model.openai.OpenAiModelName.TEXT_DAVINCI_003;
import static java.time.Duration.ofSeconds;
/**
 * Represents an OpenAI language model with a completion interface, such as text-davinci-003.
 * The model's response is streamed token by token and should be handled with {@link StreamingResponseHandler}.
 * However, it's recommended to use {@link OpenAiStreamingChatModel} instead,
 * as it offers more advanced features like function calling, multi-turn conversations, etc.
 */
public class OpenAiStreamingLanguageModel implements StreamingLanguageModel, TokenCountEstimator {
    private final OpenAiClient client;
    private final String modelName;
    private final Double temperature;
    private final Tokenizer tokenizer;
    @Builder
    public OpenAiStreamingLanguageModel(String baseUrl,
                                        String apiKey,
                                        String modelName,
                                        Double temperature,
                                        Duration timeout,
                                        Proxy proxy,
                                        Boolean logRequests,
                                        Boolean logResponses,
                                        Tokenizer tokenizer) {
        timeout = getOrDefault(timeout, ofSeconds(60));
        this.client = OpenAiClient.builder()
                .baseUrl(getOrDefault(baseUrl, OPENAI_URL))
                .openAiApiKey(apiKey)
                .callTimeout(timeout)
                .connectTimeout(timeout)
                .readTimeout(timeout)
                .writeTimeout(timeout)
                .proxy(proxy)
                .logRequests(logRequests)
                .logStreamingResponses(logResponses)
                .build();
        this.modelName = getOrDefault(modelName, TEXT_DAVINCI_003);
        this.temperature = getOrDefault(temperature, 0.7);
        this.tokenizer = getOrDefault(tokenizer, new OpenAiTokenizer(this.modelName));
```

```
    }
    @Override
    public void generate(String prompt, StreamingResponseHandler<String>
handler) {
        CompletionRequest request = CompletionRequest.builder()
                .model(modelName)
                .prompt(prompt)
                .temperature(temperature)
                .build();
        int inputTokenCount = tokenizer.estimateTokenCountInText(prompt);
        OpenAiStreamingResponseBuilder responseBuilder = new
OpenAiStreamingResponseBuilder(inputTokenCount);
        client.completion(request)
                .onPartialResponse(partialResponse -> {
                    responseBuilder.append(partialResponse);
                    String token = partialResponse.text();
                    if (token != null) {
                        handler.onNext(token);
                    }
                })
                .onComplete(() -> {
                    Response<AiMessage> response = responseBuilder.build();
                    handler.onComplete(Response.from(
                            response.content().text(),
                            response.tokenUsage(),
                            response.finishReason()
                    ));
                })
                .onError(handler::onError)
                .execute();
    }
    @Override
    public int estimateTokenCount(String prompt) {
        return tokenizer.estimateTokenCountInText(prompt);
    }
    public static OpenAiStreamingLanguageModel withApiKey(String apiKey) {
        return builder().apiKey(apiKey).build();
    }
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiStreamingResponseBuilder.java

```
package dev.langchain4j.model.openai;
import dev.ai4j.openai4j.chat.ChatCompletionChoice;
import dev.ai4j.openai4j.chat.ChatCompletionResponse;
import dev.ai4j.openai4j.chat.Delta;
import dev.ai4j.openai4j.chat.FunctionCall;
import dev.ai4j.openai4j.completion.CompletionChoice;
import dev.ai4j.openai4j.completion.CompletionResponse;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import java.util.List;
import java.util.concurrent.atomic.AtomicInteger;
import static
dev.langchain4j.model.openai.InternalOpenAiHelper.finishReasonFrom;
/**
 * This class needs to be thread safe because it is called when a streaming
result comes back
 * and there is no guarantee that this thread will be the same as the one
that initiated the request,
 * in fact it almost certainly won't be.
 */
public class OpenAiStreamingResponseBuilder {
    private final StringBuffer contentBuilder = new StringBuffer();
    private final StringBuffer toolNameBuilder = new StringBuffer();
    private final StringBuffer toolArgumentsBuilder = new StringBuffer();
    private final Integer inputTokenCount;
    private final AtomicInteger outputTokenCount = new AtomicInteger();
    private volatile String finishReason;
    public OpenAiStreamingResponseBuilder(Integer inputTokenCount) {
        this.inputTokenCount = inputTokenCount;
    }
    public void append(ChatCompletionResponse partialResponse) {
        if (partialResponse == null) {
            return;
        }
        List<ChatCompletionChoice> choices = partialResponse.choices();
        if (choices == null || choices.isEmpty()) {
            return;
        }
        ChatCompletionChoice chatCompletionChoice = choices.get(0);
        if (chatCompletionChoice == null) {
            return;
        }
        String finishReason = chatCompletionChoice.finishReason();
        if (finishReason != null) {
            this.finishReason = finishReason;
        }
        Delta delta = chatCompletionChoice.delta();
        if (delta == null) {
            return;
        }
        String content = delta.content();
        if (content != null) {
            contentBuilder.append(content);
            outputTokenCount.incrementAndGet();
            return;
```

```
        }
        FunctionCall functionCall = delta.functionCall();
        if (functionCall != null) {
            if (functionCall.name() != null) {
                toolNameBuilder.append(functionCall.name());
                outputTokenCount.incrementAndGet();
            }
            if (functionCall.arguments() != null) {
                toolArgumentsBuilder.append(functionCall.arguments());
                outputTokenCount.incrementAndGet();
            }
        }
    }
    public void append(CompletionResponse partialResponse) {
        if (partialResponse == null) {
            return;
        }
        List<CompletionChoice> choices = partialResponse.choices();
        if (choices == null || choices.isEmpty()) {
            return;
        }
        CompletionChoice completionChoice = choices.get(0);
        if (completionChoice == null) {
            return;
        }
        String finishReason = completionChoice.finishReason();
        if (finishReason != null) {
            this.finishReason = finishReason;
        }
        String token = completionChoice.text();
        if (token != null) {
            contentBuilder.append(token);
            outputTokenCount.incrementAndGet();
        }
    }
    public Response<AiMessage> build() {
        String content = contentBuilder.toString();
        if (!content.isEmpty()) {
            return Response.from(
                    AiMessage.from(content),
                    new TokenUsage(inputTokenCount, outputTokenCount.get()),
                    finishReasonFrom(finishReason)
            );
        }
        String toolName = toolNameBuilder.toString();
        if (!toolName.isEmpty()) {
            return Response.from(
                    AiMessage.from(ToolExecutionRequest.builder()
                            .name(toolName)
                            .arguments(toolArgumentsBuilder.toString())
                            .build()),
                    new TokenUsage(inputTokenCount, outputTokenCount.get()),
                    finishReasonFrom(finishReason)
            );
        }
        return null;
    }
}
```

langchain4j-open-ai\src\main\java\dev\langchain4j\model\openai\OpenAiTokenizer.java

```
package dev.langchain4j.model.openai;
import com.knuddels.jtokkit.Encodings;
import com.knuddels.jtokkit.api.Encoding;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.ToolExecutionResultMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.model.Tokenizer;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.function.Supplier;
import static dev.langchain4j.internal.Exceptions.illegalArgument;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static dev.langchain4j.model.openai.InternalOpenAiHelper.roleFrom;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO_0301;
public class OpenAiTokenizer implements Tokenizer {
    private final String modelName;
    private final Optional<Encoding> encoding;
    public OpenAiTokenizer(String modelName) {
        this.modelName = ensureNotBlank(modelName, "modelName");
        // If the model is unknown, we should NOT fail fast during the
creation of OpenAiTokenizer.
        // Doing so would cause the failure of every OpenAI***Model that uses
this tokenizer.
        // This is done to account for situations when a new OpenAI model is
available,
        // but JTokkit does not yet support it.
        this.encoding =
Encodings.newLazyEncodingRegistry().getEncodingForModel(modelName);
    }
    public int estimateTokenCountInText(String text) {
        return encoding.orElseThrow(unknownModelException())
                .countTokensOrdinary(text);
    }
    @Override
    public int estimateTokenCountInMessage(ChatMessage message) {
        int tokenCount = 0;
        tokenCount += extraTokensPerMessage();
        tokenCount += estimateTokenCountInText(message.text());
        tokenCount += estimateTokenCountInText(roleFrom(message).toString());
        if (message instanceof UserMessage) {
            UserMessage userMessage = (UserMessage) message;
            if (userMessage.name() != null) {
                tokenCount += extraTokensPerName();
                tokenCount += estimateTokenCountInText(userMessage.name());
            }
        }
        if (message instanceof AiMessage) {
            AiMessage aiMessage = (AiMessage) message;
            if (aiMessage.toolExecutionRequest() != null) {
                tokenCount += 4; // found experimentally while playing with
OpenAI API
                ToolExecutionRequest toolExecutionRequest =
aiMessage.toolExecutionRequest();
```

```
                tokenCount +=
estimateTokenCountInText(toolExecutionRequest.name());
                tokenCount +=
estimateTokenCountInText(toolExecutionRequest.arguments());
            }
        }
        if (message instanceof ToolExecutionResultMessage) {
            ToolExecutionResultMessage toolExecutionResultMessage =
(ToolExecutionResultMessage) message;
            tokenCount += -1; // found experimentally while playing with
OpenAI API
            tokenCount +=
estimateTokenCountInText(toolExecutionResultMessage.toolName());
        }
        return tokenCount;
    }
    @Override
    public int estimateTokenCountInMessages(Iterable<ChatMessage> messages) {
        // see https://github.com/openai/openai-cookbook/blob/main/examples/
How_to_count_tokens_with_tiktoken.ipynb
        int tokenCount = 3; // every reply is primed with <|start|>assistant<|
message|>
        for (ChatMessage message : messages) {
            tokenCount += estimateTokenCountInMessage(message);
        }
        return tokenCount;
    }
    @Override
    public int
estimateTokenCountInToolSpecifications(Iterable<ToolSpecification>
toolSpecifications) {
        int tokenCount = 0;
        for (ToolSpecification toolSpecification : toolSpecifications) {
            tokenCount += estimateTokenCountInText(toolSpecification.name());
            tokenCount +=
estimateTokenCountInText(toolSpecification.description());
            Map<String, Map<String, Object>> properties =
toolSpecification.parameters().properties();
            for (String property : properties.keySet()) {
                for (Map.Entry<String, Object> entry :
properties.get(property).entrySet()) {
                    if ("type".equals(entry.getKey())) {
                        tokenCount += 3; // found experimentally while
playing with OpenAI API
                        tokenCount +=
estimateTokenCountInText(entry.getValue().toString());
                    } else if ("description".equals(entry.getKey())) {
                        tokenCount += 3; // found experimentally while
playing with OpenAI API
                        tokenCount +=
estimateTokenCountInText(entry.getValue().toString());
                    } else if ("enum".equals(entry.getKey())) {
                        tokenCount -= 3; // found experimentally while
playing with OpenAI API
                        for (Object enumValue : (Object[]) entry.getValue()) {
                            tokenCount += 3; // found experimentally while
playing with OpenAI API
                            tokenCount +=
estimateTokenCountInText(enumValue.toString());
                        }
                    }
```

```
                }
            }
            tokenCount += 12; // found experimentally while playing with
OpenAI API
        }
        tokenCount += 12; // found experimentally while playing with OpenAI
API
        return tokenCount;
    }
    private int extraTokensPerMessage() {
        if (modelName.equals(GPT_3_5_TURBO_0301)) {
            return 4;
        } else {
            return 3;
        }
    }
    private int extraTokensPerName() {
        if (modelName.equals(GPT_3_5_TURBO_0301)) {
            return -1; // if there's a name, the role is omitted
        } else {
            return 1;
        }
    }
    public List<Integer> encode(String text) {
        return encoding.orElseThrow(unknownModelException())
                .encodeOrdinary(text);
    }
    public List<Integer> encode(String text, int maxTokensToEncode) {
        return encoding.orElseThrow(unknownModelException())
                .encodeOrdinary(text, maxTokensToEncode).getTokens();
    }
    public String decode(List<Integer> tokens) {
        return encoding.orElseThrow(unknownModelException())
                .decode(tokens);
    }
    private Supplier<IllegalArgumentException> unknownModelException() {
        return () -> illegalArgument("Model '%s' is unknown to jtokkit",
modelName);
    }
}
```

langchain4j-open-ai\src\test\java\dev\langchain4j\model\openai\OpenAiChatModelIT.java

```
package dev.langchain4j.model.openai;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import org.junit.jupiter.api.Test;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.model.output.FinishReason.LENGTH;
import static dev.langchain4j.model.output.FinishReason.STOP;
import static org.assertj.core.api.Assertions.assertThat;
class OpenAiChatModelIT {
    @Test
    void
should_generate_answer_and_return_token_usage_and_finish_reason_stop() {
        ChatLanguageModel model =
OpenAiChatModel.withApiKey(System.getenv("OPENAI_API_KEY"));
        UserMessage userMessage = userMessage("hello, how are you?");
        Response<AiMessage> response = model.generate(userMessage);
        System.out.println(response);
        assertThat(response.content().text()).isNotBlank();
        TokenUsage tokenUsage = response.tokenUsage();
        assertThat(tokenUsage.inputTokenCount()).isEqualTo(13);
        assertThat(tokenUsage.outputTokenCount()).isGreaterThan(1);
        assertThat(tokenUsage.totalTokenCount()).isGreaterThan(14);
        assertThat(response.finishReason()).isEqualTo(STOP);
    }
    @Test
    void
should_generate_answer_and_return_token_usage_and_finish_reason_length() {
        ChatLanguageModel model = OpenAiChatModel.builder()
                .apiKey(System.getenv("OPENAI_API_KEY"))
                .maxTokens(3)
                .build();
        UserMessage userMessage = userMessage("hello, how are you?");
        Response<AiMessage> response = model.generate(userMessage);
        System.out.println(response);
        assertThat(response.content().text()).isNotBlank();
        TokenUsage tokenUsage = response.tokenUsage();
        assertThat(tokenUsage.inputTokenCount()).isEqualTo(13);
        assertThat(tokenUsage.outputTokenCount()).isEqualTo(3);
        assertThat(tokenUsage.totalTokenCount()).isEqualTo(16);
        assertThat(response.finishReason()).isEqualTo(LENGTH);
    }
}
```

langchain4j-open-ai\src\test\java\dev\langchain4j\model\openai\OpenAiEmbeddingModelIT.java

```
package dev.langchain4j.model.openai;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class OpenAiEmbeddingModelIT {
    EmbeddingModel model =
OpenAiEmbeddingModel.withApiKey(System.getenv("OPENAI_API_KEY"));
    @Test
    void should_embed_and_return_token_usage() {
        Response<Embedding> response = model.embed("hello world");
        System.out.println(response);
        assertThat(response.content().vector()).hasSize(1536);
        TokenUsage tokenUsage = response.tokenUsage();
        assertThat(tokenUsage.inputTokenCount()).isEqualTo(2);
        assertThat(tokenUsage.outputTokenCount()).isNull();
        assertThat(tokenUsage.totalTokenCount()).isEqualTo(2);
        assertThat(response.finishReason()).isNull();
    }
}
```

langchain4j-open-ai\src\test\java\dev\langchain4j\model\openai\OpenAiLanguageModelIT.java

```
package dev.langchain4j.model.openai;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import org.junit.jupiter.api.Test;
import static dev.langchain4j.model.output.FinishReason.STOP;
import static org.assertj.core.api.Assertions.assertThat;
class OpenAiLanguageModelIT {
    LanguageModel model = OpenAiLanguageModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .logRequests(true)
            .logResponses(true)
            .build();
    @Test
    void
should_generate_answer_and_return_token_usage_and_finish_reason_stop() {
        String prompt = "Hello, how are you?";
        Response<String> response = model.generate(prompt);
        System.out.println(response);
        assertThat(response.content()).isNotBlank();
        TokenUsage tokenUsage = response.tokenUsage();
        assertThat(tokenUsage.inputTokenCount()).isEqualTo(6);
        assertThat(tokenUsage.outputTokenCount()).isGreaterThan(0);
        assertThat(tokenUsage.totalTokenCount())
                .isEqualTo(tokenUsage.inputTokenCount() +
tokenUsage.outputTokenCount());
        assertThat(response.finishReason()).isEqualTo(STOP);
    }
}
```

langchain4j-open-ai\src\test\java\dev\langchain4j\model\openai\OpenAiModerationModelIT.java

```
package dev.langchain4j.model.openai;
import dev.langchain4j.model.moderation.Moderation;
import dev.langchain4j.model.moderation.ModerationModel;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class OpenAiModerationModelIT {
    ModerationModel model = OpenAiModerationModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .build();
    @Test
    void should_flag() {
        Moderation moderation = model.moderate("I want to kill
them.").content();
        assertThat(moderation.flagged()).isTrue();
    }
    @Test
    void should_not_flag() {
        Moderation moderation = model.moderate("I want to hug
them.").content();
        assertThat(moderation.flagged()).isFalse();
    }
}
```

langchain4j-open-ai\src\test\java\dev\langchain4j\model\openai\OpenAiStreamingChatModelIT.java

```
package dev.langchain4j.model.openai;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.api.Test;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeoutException;
import static dev.langchain4j.agent.tool.JsonSchemaProperty.INTEGER;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.model.output.FinishReason.STOP;
import static dev.langchain4j.model.output.FinishReason.TOOL_EXECUTION;
import static java.util.Collections.singletonList;
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.assertj.core.api.Assertions.assertThat;
class OpenAiStreamingChatModelIT {
    StreamingChatLanguageModel model = OpenAiStreamingChatModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .logRequests(true)
            .logResponses(true)
            .build();
    @Test
    void should_stream_answer() throws ExecutionException,
InterruptedException, TimeoutException {
        CompletableFuture<String> futureAnswer = new CompletableFuture<>();
        CompletableFuture<Response<AiMessage>> futureResponse = new
CompletableFuture<>();
        model.generate("What is the capital of Germany?", new
StreamingResponseHandler<AiMessage>() {
            private final StringBuilder answerBuilder = new StringBuilder();
            @Override
            public void onNext(String token) {
                System.out.println("onNext: '" + token + "'");
                answerBuilder.append(token);
            }
            @Override
            public void onComplete(Response<AiMessage> response) {
                System.out.println("onComplete: '" + response + "'");
                futureAnswer.complete(answerBuilder.toString());
                futureResponse.complete(response);
            }
            @Override
            public void onError(Throwable error) {
                futureAnswer.completeExceptionally(error);
                futureResponse.completeExceptionally(error);
            }
        });
        String answer = futureAnswer.get(30, SECONDS);
        Response<AiMessage> response = futureResponse.get(30, SECONDS);
        assertThat(answer).contains("Berlin");
        assertThat(response.content().text()).isEqualTo(answer);
        assertThat(response.tokenUsage().inputTokenCount()).isEqualTo(14);
        assertThat(response.tokenUsage().outputTokenCount()).isGreaterThan(0);
```

```
        assertThat(response.tokenUsage().totalTokenCount())
                .isEqualTo(response.tokenUsage().inputTokenCount() +
response.tokenUsage().outputTokenCount());
        assertThat(response.finishReason()).isEqualTo(STOP);
    }
    @Test
    void should_return_tool_execution_request() throws Exception {
        ToolSpecification toolSpecification = ToolSpecification.builder()
                .name("calculator")
                .description("returns a sum of two numbers")
                .addParameter("first", INTEGER)
                .addParameter("second", INTEGER)
                .build();
        UserMessage userMessage = userMessage("Two plus two?");
        CompletableFuture<Response<AiMessage>> futureResponse = new
CompletableFuture<>();
        model.generate(singletonList(userMessage),
singletonList(toolSpecification), new StreamingResponseHandler<AiMessage>() {
            @Override
            public void onNext(String token) {
                System.out.println("onNext: '" + token + "'");
                Exception e = new IllegalStateException("onNext() should
never be called when tool is executed");
                futureResponse.completeExceptionally(e);
            }
            @Override
            public void onComplete(Response<AiMessage> response) {
                System.out.println("onComplete: '" + response + "'");
                futureResponse.complete(response);
            }
            @Override
            public void onError(Throwable error) {
                futureResponse.completeExceptionally(error);
            }
        });
        Response<AiMessage> response = futureResponse.get(30, SECONDS);
        AiMessage aiMessage = response.content();
        assertThat(aiMessage.text()).isNull();
        ToolExecutionRequest toolExecutionRequest =
aiMessage.toolExecutionRequest();
        assertThat(toolExecutionRequest.name()).isEqualTo("calculator");
        assertThat(toolExecutionRequest.arguments()).isEqualToIgnoringWhitespa
ce("{\"first\": 2, \"second\": 2}");
        assertThat(response.tokenUsage().inputTokenCount()).isEqualTo(50);
        assertThat(response.tokenUsage().outputTokenCount()).isGreaterThan(0);
        assertThat(response.tokenUsage().totalTokenCount())
                .isEqualTo(response.tokenUsage().inputTokenCount() +
response.tokenUsage().outputTokenCount());
        assertThat(response.finishReason()).isEqualTo(TOOL_EXECUTION);
    }
}
```

langchain4j-open-ai\src\test\java\dev\langchain4j\model\openai\OpenAiStreamingLanguageModelIT.java

```
package dev.langchain4j.model.openai;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.language.StreamingLanguageModel;
import dev.langchain4j.model.output.Response;
import org.junit.jupiter.api.Test;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeoutException;
import static dev.langchain4j.model.output.FinishReason.STOP;
import static java.util.concurrent.TimeUnit.SECONDS;
import static org.assertj.core.api.Assertions.assertThat;
class OpenAiStreamingLanguageModelIT {
    StreamingLanguageModel model = OpenAiStreamingLanguageModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .logRequests(true)
            .logResponses(true)
            .build();
    @Test
    void should_stream_answer() throws ExecutionException,
InterruptedException, TimeoutException {
        CompletableFuture<String> futureAnswer = new CompletableFuture<>();
        CompletableFuture<Response<String>> futureResponse = new
CompletableFuture<>();
        model.generate("What is the capital of Germany?", new
StreamingResponseHandler<String>() {
            private final StringBuilder answerBuilder = new StringBuilder();
            @Override
            public void onNext(String token) {
                System.out.println("onNext: '" + token + "'");
                answerBuilder.append(token);
            }
            @Override
            public void onComplete(Response<String> response) {
                System.out.println("onComplete: '" + response + "'");
                futureAnswer.complete(answerBuilder.toString());
                futureResponse.complete(response);
            }
            @Override
            public void onError(Throwable error) {
                futureAnswer.completeExceptionally(error);
                futureResponse.completeExceptionally(error);
            }
        });
        String answer = futureAnswer.get(30, SECONDS);
        Response<String> response = futureResponse.get(30, SECONDS);
        assertThat(answer).contains("Berlin");
        assertThat(response.content()).isEqualTo(answer);
        assertThat(response.tokenUsage().inputTokenCount()).isEqualTo(7);
        assertThat(response.tokenUsage().outputTokenCount()).isGreaterThan(1);
        assertThat(response.tokenUsage().totalTokenCount()).isGreaterThan(8);
        assertThat(response.finishReason()).isEqualTo(STOP);
    }
}
```

langchain4j-open-ai\src\test\java\dev\langchain4j\model\openai\OpenAiTokenizerTest.java

```
package dev.langchain4j.model.openai;
import dev.langchain4j.agent.tool.P;
import dev.langchain4j.agent.tool.Tool;
import dev.langchain4j.agent.tool.ToolExecutionRequest;
import dev.langchain4j.data.message.ChatMessage;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Stream;
import static dev.langchain4j.data.message.AiMessage.aiMessage;
import static dev.langchain4j.data.message.ToolExecutionResultMessage.toolExec
utionResultMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
import static org.assertj.core.api.Assertions.assertThat;
class OpenAiTokenizerTest {
    OpenAiTokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
    @ParameterizedTest
    @MethodSource
    void should_count_tokens_in_messages(List<ChatMessage> messages, int
expectedTokenCount) {
        int tokenCount = tokenizer.estimateTokenCountInMessages(messages);
        assertThat(tokenCount).isEqualTo(expectedTokenCount);
    }
    static Stream<Arguments> should_count_tokens_in_messages() {
        // expected token count was taken from real OpenAI responses
(usage.prompt_tokens)
        return Stream.of(
                Arguments.of(singletonList(userMessage("hello")), 8),
                Arguments.of(singletonList(userMessage("Klaus", "hello")),
11),
                Arguments.of(asList(userMessage("hello"), aiMessage("hi
there")), 14),
                Arguments.of(asList(
                        userMessage("How much is 2 plus 2?"),
                        aiMessage(ToolExecutionRequest.builder()
                                .name("calculator")
                                .arguments("{\"a\":2, \"b\":2}")
                                .build())
                ), 35),
                Arguments.of(asList(
                        userMessage("How much is 2 plus 2?"),
                        aiMessage(ToolExecutionRequest.builder()
                                .name("calculator")
                                .arguments("{\"a\":2, \"b\":2}")
                                .build()),
                        toolExecutionResultMessage("calculator", "4")
                ), 40)
        );
    }
    static class Tools {
        @Tool
```

```
        int add(int a, int b) {
            return a + b;
        }
        @Tool("calculates the square root of the provided number")
        double squareRoot(@P("number to operate on") double number) {
            return Math.sqrt(number);
        }
        @Tool
        int temperature(String location, TemperatureUnit temperatureUnit) {
            return 0;
        }
    }
    enum TemperatureUnit {
        F, C
    }
    @Test
    void should_count_tokens_in_tools() {
        int tokenCount = tokenizer.estimateTokenCountInTools(new Tools());
        assertThat(tokenCount).isEqualTo(93); // found experimentally while
playing with OpenAI API
    }
    @Test
    void should_encode_and_decode_text() {
        String originalText = "This is a text which will be encoded and
decoded back.";
        List<Integer> tokens = tokenizer.encode(originalText);
        String decodedText = tokenizer.decode(tokens);
        assertThat(decodedText).isEqualTo(originalText);
    }
    @Test
    void should_encode_with_truncation_and_decode_text() {
        String originalText = "This is a text which will be encoded with
truncation and decoded back.";
        List<Integer> tokens = tokenizer.encode(originalText, 10);
        assertThat(tokens).hasSize(10);
        String decodedText = tokenizer.decode(tokens);
        assertThat(decodedText).isEqualTo("This is a text which will be
encoded with trunc");
    }
    @Test
    void should_count_tokens_in_short_texts() {
        assertThat(tokenizer.estimateTokenCountInText("Hello")).isEqualTo(1);
        assertThat(tokenizer.estimateTokenCountInText("Hello!")).isEqualTo(2);
        assertThat(tokenizer.estimateTokenCountInText("Hello, how are
you?")).isEqualTo(6);
    }
    @Test
    void should_count_tokens_in_average_text() {
        String text1 = "Hello, how are you doing? What do you want to talk
about?";
        assertThat(tokenizer.estimateTokenCountInText(text1)).isEqualTo(15);
        String text2 = String.join(" ", repeat("Hello, how are you doing?
What do you want to talk about?", 2));
        assertThat(tokenizer.estimateTokenCountInText(text2)).isEqualTo(2 *
15);
        String text3 = String.join(" ", repeat("Hello, how are you doing?
What do you want to talk about?", 3));
        assertThat(tokenizer.estimateTokenCountInText(text3)).isEqualTo(3 *
15);
    }
    @Test
```

```
    void should_count_tokens_in_large_text() {
        String text1 = String.join(" ", repeat("Hello, how are you doing? What do you want to talk about?", 10));
        assertThat(tokenizer.estimateTokenCountInText(text1)).isEqualTo(10 * 15);
        String text2 = String.join(" ", repeat("Hello, how are you doing? What do you want to talk about?", 50));
        assertThat(tokenizer.estimateTokenCountInText(text2)).isEqualTo(50 * 15);
        String text3 = String.join(" ", repeat("Hello, how are you doing? What do you want to talk about?", 100));
        assertThat(tokenizer.estimateTokenCountInText(text3)).isEqualTo(100 * 15);
    }
    public static List<String> repeat(String s, int n) {
        List<String> result = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            result.add(s);
        }
        return result;
    }
}
```

langchain4j-opensearch\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-opensearch</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with OpenSearch</name>
    <description>Requires Java 11</description>
    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.httpcomponents.client5</groupId>
            <artifactId>httpclient5</artifactId>
        </dependency>
        <dependency>
            <groupId>org.opensearch.client</groupId>
            <artifactId>opensearch-java</artifactId>
        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>opensearch</artifactId>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-core</artifactId>
        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>apache-client</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
```

```
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>testcontainers</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>junit-jupiter</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.opensearch</groupId>
            <artifactId>opensearch-testcontainers</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-opensearch\src\main\java\dev\langchain4j\store\embedding\opensearch\Document.java

```
package dev.langchain4j.store.embedding.opensearch;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Map;
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
class Document {
    private float[] vector;
    private String text;
    private Map<String, String> metadata;
}
```

langchain4j-opensearch\src\main\java\dev\langchain4j\store\embedding\opensearch\OpenSearchEmbeddingStore.java

```
package dev.langchain4j.store.embedding.opensearch;
import com.fasterxml.jackson.core.JsonProcessingException;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import org.apache.hc.client5.http.auth.AuthScope;
import org.apache.hc.client5.http.auth.UsernamePasswordCredentials;
import org.apache.hc.client5.http.impl.auth.BasicCredentialsProvider;
import org.apache.hc.client5.http.impl.nio.PoolingAsyncClientConnectionManager
Builder;
import org.apache.hc.core5.http.HttpHost;
import org.apache.hc.core5.http.message.BasicHeader;
import org.opensearch.client.json.JsonData;
import org.opensearch.client.json.jackson.JacksonJsonpMapper;
import org.opensearch.client.opensearch.OpenSearchClient;
import org.opensearch.client.opensearch._types.ErrorCause;
import org.opensearch.client.opensearch._types.InlineScript;
import org.opensearch.client.opensearch._types.mapping.Property;
import org.opensearch.client.opensearch._types.mapping.TextProperty;
import org.opensearch.client.opensearch._types.mapping.TypeMapping;
import org.opensearch.client.opensearch._types.query_dsl.Query;
import org.opensearch.client.opensearch._types.query_dsl.ScriptScoreQuery;
import org.opensearch.client.opensearch.core.BulkRequest;
import org.opensearch.client.opensearch.core.BulkResponse;
import org.opensearch.client.opensearch.core.SearchRequest;
import org.opensearch.client.opensearch.core.SearchResponse;
import org.opensearch.client.opensearch.core.bulk.BulkResponseItem;
import org.opensearch.client.transport.OpenSearchTransport;
import org.opensearch.client.transport.aws.AwsSdk2Transport;
import org.opensearch.client.transport.aws.AwsSdk2TransportOptions;
import org.opensearch.client.transport.endpoints.BooleanResponse;
import
org.opensearch.client.transport.httpclient5.ApacheHttpClient5TransportBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.http.SdkHttpClient;
import software.amazon.awssdk.http.apache.ApacheHttpClient;
import software.amazon.awssdk.regions.Region;
import java.io.IOException;
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import static dev.langchain4j.internal.Utils.*;
import static dev.langchain4j.internal.ValidationUtils.ensureNotNull;
import static dev.langchain4j.internal.ValidationUtils.ensureTrue;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.toList;
/**
 * Represents an <a href="https://opensearch.org/">OpenSearch</a> index as an
 * embedding store. This implementation uses K-NN and the cosinesimil space
type.
 */
public class OpenSearchEmbeddingStore implements EmbeddingStore<TextSegment> {
```

```
    private static final Logger log =
LoggerFactory.getLogger(OpenSearchEmbeddingStore.class);
    private final String indexName;
    private final OpenSearchClient client;
    /**
     * Creates an instance of OpenSearchEmbeddingStore to connect with
     * OpenSearch clusters running locally and network reachable.
     *
     * @param serverUrl OpenSearch Server URL.
     * @param apiKey    OpenSearch API key (optional)
     * @param userName  OpenSearch username (optional)
     * @param password  OpenSearch password (optional)
     * @param indexName OpenSearch index name (optional). Default value:
"default"
     */
    public OpenSearchEmbeddingStore(String serverUrl,
                                    String apiKey,
                                    String userName,
                                    String password,
                                    String indexName) {
        HttpHost openSearchHost;
        try {
            openSearchHost = HttpHost.create(serverUrl);
        } catch (URISyntaxException se) {
            log.error("[I/O OpenSearch Exception]", se);
            throw new OpenSearchRequestFailedException(se.getMessage());
        }
        OpenSearchTransport transport = ApacheHttpClient5TransportBuilder
                .builder(openSearchHost)
                .setMapper(new JacksonJsonpMapper())
                .setHttpClientConfigCallback(httpClientBuilder -> {
                    if (!isNullOrBlank(apiKey)) {
                        httpClientBuilder.setDefaultHeaders(singletonList(
                                new BasicHeader("Authorization", "ApiKey " +
apiKey)
                        ));
                    }
                    if (!isNullOrBlank(userName) && !isNullOrBlank(password))
{
                        BasicCredentialsProvider credentialsProvider = new
BasicCredentialsProvider();
                        credentialsProvider.setCredentials(new
AuthScope(openSearchHost),
                                new UsernamePasswordCredentials(userName,
password.toCharArray()));

httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
                    }
                    httpClientBuilder.setConnectionManager(PoolingAsyncClientC
onnectionManagerBuilder.create().build());
                    return httpClientBuilder;
                })
                .build();
        this.client = new OpenSearchClient(transport);
        this.indexName = ensureNotNull(indexName, "indexName");
    }
    /**
     * Creates an instance of OpenSearchEmbeddingStore to connect with
     * OpenSearch clusters running as a fully managed service at AWS.
     *
     * @param serverUrl   OpenSearch Server URL.
```

```
     * @param serviceName The AWS signing service name, one of `es` (Amazon
OpenSearch) or `aoss` (Amazon OpenSearch Serverless).
     * @param region      The AWS region for which requests will be signed.
This should typically match the region in `serverUrl`.
     * @param options     The options to establish connection with the
service. It must include which credentials should be used.
     * @param indexName   OpenSearch index name (optional). Default value:
"default"
     */
    public OpenSearchEmbeddingStore(String serverUrl,
                                    String serviceName,
                                    String region,
                                    AwsSdk2TransportOptions options,
                                    String indexName) {
        Region selectedRegion = Region.of(region);
        SdkHttpClient httpClient = ApacheHttpClient.builder().build();
        OpenSearchTransport transport = new AwsSdk2Transport(httpClient,
serverUrl, serviceName, selectedRegion, options);
        this.client = new OpenSearchClient(transport);
        this.indexName = ensureNotNull(indexName, "indexName");
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String serverUrl;
        private String apiKey;
        private String userName;
        private String password;
        private String serviceName;
        private String region;
        private AwsSdk2TransportOptions options;
        private String indexName = "default";
        public Builder serverUrl(String serverUrl) {
            this.serverUrl = serverUrl;
            return this;
        }
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        public Builder userName(String userName) {
            this.userName = userName;
            return this;
        }
        public Builder password(String password) {
            this.password = password;
            return this;
        }
        public Builder serviceName(String serviceName) {
            this.serviceName = serviceName;
            return this;
        }
        public Builder region(String region) {
            this.region = region;
            return this;
        }
        public Builder options(AwsSdk2TransportOptions options) {
            this.options = options;
            return this;
        }
```

```
        public Builder indexName(String indexName) {
            this.indexName = indexName;
            return this;
        }
        public OpenSearchEmbeddingStore build() {
            if (!isNullOrBlank(serviceName) && !isNullOrBlank(region) &&
options != null) {
                return new OpenSearchEmbeddingStore(serverUrl, serviceName,
region, options, indexName);
            }
            return new OpenSearchEmbeddingStore(serverUrl, apiKey, userName,
password, indexName);
        }
    }
    @Override
    public String add(Embedding embedding) {
        String id = randomUUID();
        add(id, embedding);
        return id;
    }
    @Override
    public void add(String id, Embedding embedding) {
        addInternal(id, embedding, null);
    }
    @Override
    public String add(Embedding embedding, TextSegment textSegment) {
        String id = randomUUID();
        addInternal(id, embedding, textSegment);
        return id;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings) {
        List<String> ids = embeddings.stream()
                .map(ignored -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, null);
        return ids;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
embedded) {
        List<String> ids = embeddings.stream()
                .map(ignored -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, embedded);
        return ids;
    }
    /**
     * This implementation uses the exact k-NN with scoring script to
calculate
     * See https://opensearch.org/docs/latest/search-plugins/knn/knn-score-
script/
     */
    @Override
    public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
referenceEmbedding, int maxResults, double minScore) {
        List<EmbeddingMatch<TextSegment>> matches;
        try {
            ScriptScoreQuery scriptScoreQuery =
buildDefaultScriptScoreQuery(referenceEmbedding.vector(), (float) minScore);
            SearchResponse<Document> response = client.search(
```

```
                    SearchRequest.of(s -> s.index(indexName)
                            .query(n -> n.scriptScore(scriptScoreQuery))
                            .size(maxResults)),
                    Document.class
            );
            matches = toEmbeddingMatch(response);
        } catch (IOException ex) {
            log.error("[I/O OpenSearch Exception]", ex);
            throw new OpenSearchRequestFailedException(ex.getMessage());
        }
        return matches;
    }
    private ScriptScoreQuery buildDefaultScriptScoreQuery(float[] vector,
float minScore) throws JsonProcessingException {
        return ScriptScoreQuery.of(q -> q.minScore(minScore)
                .query(Query.of(qu -> qu.matchAll(m -> m)))
                .script(s -> s.inline(InlineScript.of(i -> i
                        .source("knn_score")
                        .lang("knn")
                        .params("field", JsonData.of("vector"))
                        .params("query_value", JsonData.of(vector))
                        .params("space_type", JsonData.of("cosinesimil")))))
                .boost(0.5f));
        // ===> From the OpenSearch documentation:
        // "Cosine similarity returns a number between -1 and 1, and because
OpenSearch
        // relevance scores can't be below 0, the k-NN plugin adds 1 to get
the final score."
        // See https://opensearch.org/docs/latest/search-plugins/knn/knn-
score-script
        // Thus, the query applies a boost of `0.5` to keep score in the
range [0, 1]
    }
    private void addInternal(String id, Embedding embedding, TextSegment
embedded) {
        addAllInternal(singletonList(id), singletonList(embedding), embedded
== null ? null : singletonList(embedded));
    }
    private void addAllInternal(List<String> ids, List<Embedding> embeddings,
List<TextSegment> embedded) {
        if (isCollectionEmpty(ids) || isCollectionEmpty(embeddings)) {
            log.info("[do not add empty embeddings to opensearch]");
            return;
        }
        ensureTrue(ids.size() == embeddings.size(), "ids size is not equal to
embeddings size");
        ensureTrue(embedded == null || embeddings.size() == embedded.size(),
"embeddings size is not equal to embedded size");
        try {
            createIndexIfNotExist(embeddings.get(0).dimensions());
            bulk(ids, embeddings, embedded);
        } catch (IOException ex) {
            log.error("[I/O OpenSearch Exception]", ex);
            throw new OpenSearchRequestFailedException(ex.getMessage());
        }
    }
    private void createIndexIfNotExist(int dimension) throws IOException {
        BooleanResponse response = client.indices().exists(c ->
c.index(indexName));
        if (!response.value()) {
            client.indices()
```

```
                    .create(c -> c.index(indexName)
                            .settings(s -> s.knn(true))
                            .mappings(getDefaultMappings(dimension)));
        }
    }
    private TypeMapping getDefaultMappings(int dimension) {
        Map<String, Property> properties = new HashMap<>(4);
        properties.put("text", Property.of(p -> p.text(TextProperty.of(t ->
t))));
        properties.put("vector", Property.of(p -> p.knnVector(
                k -> k.dimension(dimension)
        )));
        return TypeMapping.of(c -> c.properties(properties));
    }
    private void bulk(List<String> ids, List<Embedding> embeddings,
List<TextSegment> embedded) throws IOException {
        int size = ids.size();
        BulkRequest.Builder bulkBuilder = new BulkRequest.Builder();
        for (int i = 0; i < size; i++) {
            int finalI = i;
            Document document = Document.builder()
                    .vector(embeddings.get(i).vector())
                    .text(embedded == null ? null : embedded.get(i).text())
                    .metadata(embedded == null ? null :
Optional.ofNullable(embedded.get(i).metadata())
                            .map(Metadata::asMap)
                            .orElse(null))
                    .build();
            bulkBuilder.operations(op -> op.index(
                    idx -> idx
                            .index(indexName)
                            .id(ids.get(finalI))
                            .document(document)
            ));
        }
        BulkResponse bulkResponse = client.bulk(bulkBuilder.build());
        if (bulkResponse.errors()) {
            for (BulkResponseItem item : bulkResponse.items()) {
                if (item.error() != null) {
                    ErrorCause errorCause = item.error();
                    if (errorCause != null) {
                        throw new OpenSearchRequestFailedException(
                                "type: " + errorCause.type() + "," +
                                        "reason: " + errorCause.reason());
                    }
                }
            }
        }
    }
    private List<EmbeddingMatch<TextSegment>>
toEmbeddingMatch(SearchResponse<Document> response) {
        return response.hits().hits().stream()
                .map(hit -> Optional.ofNullable(hit.source())
                        .map(document -> new EmbeddingMatch<>(
                                hit.score(),
                                hit.id(),
                                new Embedding(document.getVector()),
                                document.getText() == null
                                        ? null
                                        :
TextSegment.from(document.getText(), new Metadata(document.getMetadata()))
```

```
                        )).orElse(null))
                .collect(toList());
    }
}
```

langchain4j-opensearch\src\main\java\dev\langchain4j\store\embedding\opensearch\OpenSearchRequestFailedException.java

```
package dev.langchain4j.store.embedding.opensearch;
public class OpenSearchRequestFailedException extends RuntimeException {
    public OpenSearchRequestFailedException() {
        super();
    }
    public OpenSearchRequestFailedException(String message) {
        super(message);
    }
    public OpenSearchRequestFailedException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

langchain4j-opensearch\src\test\java\dev\langchain4j\store\embedding\opensearch\OpenSearchEmbeddingStoreAWSTest.java

```
package dev.langchain4j.store.embedding.opensearch;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.opensearch.client.transport.aws.AwsSdk2TransportOptions;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import java.util.List;
import java.util.UUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
@Disabled("Needs OpenSearch running with AWS")
public class OpenSearchEmbeddingStoreAWSTest {
    /**
     * To run the tests locally, you have to provide an Amazon OpenSearch domain. The code uses
     * the credentials stored locally in your machine. For more information about how to configure
     * your credentials locally, see https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html.
     */
    private final String domainEndpoint = "your-generated-domain-endpoint-with-no-https.us-east-1.es.amazonaws.com";
    private final ProfileCredentialsProvider credentials = ProfileCredentialsProvider.create("default");
    private final AwsSdk2TransportOptions transportOptions = AwsSdk2TransportOptions.builder()
            .setCredentials(credentials)
            .build();
    private final EmbeddingStore<TextSegment> embeddingStore = OpenSearchEmbeddingStore.builder()
            .serverUrl(domainEndpoint)
            .serviceName("es")
            .region("us-east-1")
            .options(transportOptions)
            .indexName(randomUUID())
            .build();
    private final EmbeddingModel embeddingModel = new AllMiniLmL6V2QuantizedEmbeddingModel();
    @Test
    void should_add_embedding() throws InterruptedException {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant = embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
```

```
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() throws InterruptedException {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() throws InterruptedException {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_embedding_with_segment_with_metadata() throws
InterruptedException {
        TextSegment segment = TextSegment.from(randomUUID(),
Metadata.from("test-key", "test-value"));
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() throws InterruptedException {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
```

```
secondEmbedding));
        assertThat(ids).hasSize(2);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() throws
InterruptedException {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
        );
        assertThat(ids).hasSize(2);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() throws InterruptedException {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
```

```
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() throws InterruptedException {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
referenceEmbedding)),
                withPercentage(1)
        );
    }
    public static String randomUUID() {
        return UUID.randomUUID().toString();
    }
}
```

langchain4j-opensearch\src\test\java\dev\langchain4j\store\embedding\opensearch\OpenSearchEmbeddingStoreLocalTest.java

```
package dev.langchain4j.store.embedding.opensearch;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.opensearch.testcontainers.OpensearchContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.utility.DockerImageName;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
@Disabled("Needs OpenSearch running locally")
class OpenSearchEmbeddingStoreLocalTest {
    /**
     * To run the tests locally, you don't need to have OpenSearch up-and-
running. This implementation
     * uses TestContainers (https://testcontainers.com) and the built-in
support for OpenSearch. Thus,
     * if you just execute the tests then a container will be spun up
automatically for you.
     */
    @Container
    private static final OpensearchContainer opensearch =
            new OpensearchContainer(DockerImageName.parse("opensearchproject/
opensearch:2.10.0"));
    private final EmbeddingStore<TextSegment> embeddingStore =
OpenSearchEmbeddingStore.builder()
            .serverUrl(opensearch.getHttpHostAddress())
            .indexName(randomUUID())
            .build();
    private final EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    @BeforeAll
    static void startOpenSearch() {
        opensearch.start();
    }
    @Test
    void should_add_embedding() throws InterruptedException {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
```

```
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() throws InterruptedException {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() throws InterruptedException {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_embedding_with_segment_with_metadata() throws
InterruptedException {
        TextSegment segment = TextSegment.from(randomUUID(),
Metadata.from("test-key", "test-value"));
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() throws InterruptedException {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
```

```
        assertThat(ids).hasSize(2);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() throws
InterruptedException {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
        );
        assertThat(ids).hasSize(2);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() throws InterruptedException {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
```

```
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() throws InterruptedException {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        Thread.sleep(2000);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
referenceEmbedding)),
                withPercentage(1)
        );
    }
}
```

langchain4j-parent\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-parent</artifactId>
    <version>0.23.0</version>
    <packaging>pom</packaging>
    <name>langchain4j parent POM</name>
    <description>Parent POM for langchain4j submodules</description>
    <url>https://github.com/langchain4j/langchain4j</url>
    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <openai4j.version>0.10.0</openai4j.version>
        <retrofit.version>2.9.0</retrofit.version>
        <okhttp.version>4.10.0</okhttp.version>
        <jtokkit.version>0.6.1</jtokkit.version>
        <lombok.version>1.18.30</lombok.version>
        <pdfbox.version>2.0.29</pdfbox.version>
        <jsoup.veresion>1.16.1</jsoup.veresion>
        <mustache.version>0.9.10</mustache.version>
        <slf4j-api.version>2.0.7</slf4j-api.version>
        <gson.version>2.10.1</gson.version>
        <junit.version>5.10.0</junit.version>
        <testcontainers.version>1.19.1</testcontainers.version>
        <mockito.version>4.11.0</mockito.version>
        <assertj.version>3.24.2</assertj.version>
        <tinylog.version>2.6.2</tinylog.version>
        <spring-boot.version>2.7.14</spring-boot.version>
        <snakeyaml.version>2.0</snakeyaml.version>
        <httpclient5.version>5.2.1</httpclient5.version>
        <opensearch-java.version>2.6.0</opensearch-java.version>
        <aws-java-sdk-core.version>1.12.564</aws-java-sdk-core.version>
        <aws-opensearch.version>2.20.161</aws-opensearch.version>
        <opensearch-containers.version>2.0.0</opensearch-containers.version>
        <elastic.version>8.9.0</elastic.version>
        <jackson.version>2.12.7.1</jackson.version>
        <jedis.version>5.0.0</jedis.version>
        <aws.java.sdk.version>2.20.149</aws.java.sdk.version>
        <testcontainers.version>1.19.0</testcontainers.version>
        <netty.version>4.1.100.Final</netty.version>
    </properties>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>dev.ai4j</groupId>
                <artifactId>openai4j</artifactId>
                <version>${openai4j.version}</version>
            </dependency>
            <dependency>
                <groupId>com.squareup.retrofit2</groupId>
                <artifactId>retrofit</artifactId>
                <version>${retrofit.version}</version>
            </dependency>
```

```
<dependency>
    <groupId>com.squareup.retrofit2</groupId>
    <artifactId>converter-gson</artifactId>
    <version>${retrofit.version}</version>
</dependency>
<dependency>
    <groupId>com.squareup.okhttp3</groupId>
    <artifactId>okhttp</artifactId>
    <version>${okhttp.version}</version>
</dependency>
<dependency>
    <groupId>com.knuddels</groupId>
    <artifactId>jtokkit</artifactId>
    <version>${jtokkit.version}</version>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>${lombok.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.pdfbox</groupId>
    <artifactId>pdfbox</artifactId>
    <version>${pdfbox.version}</version>
</dependency>
<dependency>
    <groupId>org.jsoup</groupId>
    <artifactId>jsoup</artifactId>
    <version>${jsoup.veresion}</version>
</dependency>
<dependency>
    <groupId>com.github.spullara.mustache.java</groupId>
    <artifactId>compiler</artifactId>
    <version>${mustache.version}</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j-api.version}</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>${gson.version}</version>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit.version}</version>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <version>${junit.version}</version>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>${junit.version}</version>
</dependency>
<dependency>
```

```
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>${testcontainers.version}</version>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>testcontainers</artifactId>
    <version>${testcontainers.version}</version>
</dependency>
<dependency>
    <groupId>org.opensearch</groupId>
    <artifactId>opensearch-testcontainers</artifactId>
    <version>${opensearch-containers.version}</version>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>${mockito.version}</version>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>${mockito.version}</version>
</dependency>
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>${assertj.version}</version>
</dependency>
<dependency>
    <groupId>org.tinylog</groupId>
    <artifactId>tinylog-impl</artifactId>
    <version>${tinylog.version}</version>
</dependency>
<dependency>
    <groupId>org.tinylog</groupId>
    <artifactId>slf4j-tinylog</artifactId>
    <version>${tinylog.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <version>${spring-boot.version}</version>
    <exclusions>
        <!-- due to vulnerabilities -->
        <exclusion>
            <groupId>org.yaml</groupId>
            <artifactId>snakeyaml</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.yaml</groupId>
    <artifactId>snakeyaml</artifactId>
    <version>${snakeyaml.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <version>${spring-boot.version}</version>
</dependency>
```

```
            <dependency>
                <groupId>org.apache.httpcomponents.client5</groupId>
                <artifactId>httpclient5</artifactId>
                <version>${httpclient5.version}</version>
            </dependency>
            <dependency>
                <groupId>org.opensearch.client</groupId>
                <artifactId>opensearch-java</artifactId>
                <version>${opensearch-java.version}</version>
            </dependency>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>opensearch</artifactId>
                <version>${aws-opensearch.version}</version>
            </dependency>
            <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-java-sdk-core</artifactId>
                <version>${aws-java-sdk-core.version}</version>
            </dependency>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>apache-client</artifactId>
                <version>${aws-opensearch.version}</version>
            </dependency>
            <dependency>
                <groupId>co.elastic.clients</groupId>
                <artifactId>elasticsearch-java</artifactId>
                <version>${elastic.version}</version>
            </dependency>
            <dependency>
                <groupId>com.fasterxml.jackson.core</groupId>
                <artifactId>jackson-databind</artifactId>
                <version>${jackson.version}</version>
            </dependency>
            <dependency>
                <groupId>org.apache.opennlp</groupId>
                <artifactId>opennlp-tools</artifactId>
                <version>1.9.4</version>
            </dependency>
            <dependency>
                <groupId>dev.langchain4j</groupId>
                <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</
artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>redis.clients</groupId>
                <artifactId>jedis</artifactId>
                <version>${jedis.version}</version>
            </dependency>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>bom</artifactId>
                <version>${aws.java.sdk.version}</version>
                <type>pom</type>
                <scope>import</scope>
                <exclusions>
                    <!-- Exclusion due to CWE-295 vulnerability -->
                    <exclusion>
                        <groupId>io.netty</groupId>
```

```
                    <artifactId>netty-handler</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>testcontainers</artifactId>
            <version>${testcontainers.version}</version>
        </dependency>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>localstack</artifactId>
            <version>${testcontainers.version}</version>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.1.2</version>
        </plugin>
        <plugin>
            <groupId>org.sonatype.plugins</groupId>
            <artifactId>nexus-staging-maven-plugin</artifactId>
            <version>1.6.13</version>
            <extensions>true</extensions>
            <configuration>
                <serverId>ossrh</serverId>
                <nexusUrl>https://s01.oss.sonatype.org/</nexusUrl>
                <autoReleaseAfterClose>false</autoReleaseAfterClose>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-source-plugin</artifactId>
            <version>3.2.1</version>
            <executions>
                <execution>
                    <id>attach-sources</id>
                    <goals>
                        <goal>jar-no-fork</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-javadoc-plugin</artifactId>
            <version>3.5.0</version>
            <executions>
                <execution>
                    <id>attach-javadocs</id>
                    <goals>
                        <goal>jar</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
```

```
                <!-- failsafe will be in charge of running the integration
tests (everything that ends in IT) -->
                <artifactId>maven-failsafe-plugin</artifactId>
                <version>3.1.2</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>integration-test</goal>
                            <goal>verify</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
    <developers>
        <developer>
            <id>deep-learning-dynamo</id>
            <email>deeplearningdynamo@gmail.com</email>
            <url>https://github.com/deep-learning-dynamo</url>
        </developer>
        <developer>
            <id>kuraleta</id>
            <email>digital.kuraleta@gmail.com</email>
            <url>https://github.com/kuraleta</url>
        </developer>
    </developers>
    <scm>
        <url>https://github.com/langchain4j/langchain4j</url>
        <connection>scm:git:git://github.com/langchain4j/langchain4j.git</
connection>
        <developerConnection>scm:git:git@github.com:langchain4j/
langchain4j.git</developerConnection>
    </scm>
    <distributionManagement>
        <snapshotRepository>
            <id>ossrh</id>
            <url>https://s01.oss.sonatype.org/content/repositories/snapshots</
url>
        </snapshotRepository>
    </distributionManagement>
    <profiles>
        <profile>
            <id>sign</id>
            <activation>
                <property>
                    <name>sign</name>
                </property>
            </activation>
            <build>
                <plugins>
                    <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
```

```
                        <artifactId>maven-gpg-plugin</artifactId>
                        <version>3.0.1</version>
                        <executions>
                            <execution>
                                <id>sign-artifacts</id>
                                <phase>verify</phase>
                                <goals>
                                    <goal>sign</goal>
                                </goals>
                            </execution>
                        </executions>
                    </plugin>
                </plugins>
            </build>
        </profile>
        <profile>
            <id>compliance</id>
            <activation>
                <property>
                    <name>compliance</name>
                </property>
            </activation>
            <build>
                <plugins>
                    <plugin>
                        <groupId>org.honton.chas</groupId>
                        <artifactId>license-maven-plugin</artifactId>
                        <version>0.0.3</version>
                        <executions>
                            <execution>
                                <goals>
                                    <goal>compliance</goal>
                                </goals>
                            </execution>
                        </executions>
                        <configuration>
                            <scopes>compile,runtime,provided,test</scopes>
                            <acceptableLicenses>
                                <license>
                                    <name>(The )?(Apache License, Version
2\.0)|(Apache-2\.0)|(The Apache Software License, Version 2\.0)</name>
                                    <url>https?://www\.apache\.org/licenses/
LICENSE-2\.0</url>
                                </license>
                                <license>
                                    <name>(The MIT License|MIT License|MIT)</
name>
                                    <url>(https?://opensource.org/licenses/
MIT|https://projectlombok.org/LICENSE)</url>
                                </license>
                                <license>
                                    <name>Eclipse Public License v2.0</name>
                                    <url>https?://www.eclipse.org/legal/epl-
v20\.html</url>
                                </license>
                            </acceptableLicenses>
                        </configuration>
                    </plugin>
                </plugins>
            </build>
        </profile>
```

```
    </profiles>
</project>
```

langchain4j-pgvector\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <properties>
        <pgvector-java.version>0.1.3</pgvector-java.version>
        <postgresql.version>42.6.0</postgresql.version>
        <spotless.version>2.40.0</spotless.version>
    </properties>
    <artifactId>langchain4j-pgvector</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with PGVector</name>
    <description>It uses the pgvector-java library</description>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.pgvector</groupId>
            <artifactId>pgvector</artifactId>
            <version>${pgvector-java.version}</version>
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>${postgresql.version}</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
```

```
        <groupId>org.testcontainers</groupId>
        <artifactId>testcontainers</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>junit-jupiter</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>postgresql</artifactId>
        <version>1.19.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.honton.chas</groupId>
            <artifactId>license-maven-plugin</artifactId>
            <configuration>
                <skipCompliance>true</skipCompliance>
            </configuration>
        </plugin>
        <plugin>
            <groupId>com.diffplug.spotless</groupId>
            <artifactId>spotless-maven-plugin</artifactId>
            <version>${spotless.version}</version>
            <configuration>
                <ratchetFrom>origin/main</ratchetFrom>
                <formats>
                    <format>
                        <includes>
                            <include>*.md</include>
                            <include>.gitignore</include>
                        </includes>
                        <trimTrailingWhitespace/>
                        <endWithNewline/>
                        <indent>
                            <tabs>true</tabs>
                            <spacesPerTab>4</spacesPerTab>
                        </indent>
                    </format>
                </formats>
                <java>
                    <googleJavaFormat>
                        <version>1.8.0</version>
                        <style>AOSP</style>
                        <reflowLongStrings>true</reflowLongStrings>
                        <formatJavadoc>false</formatJavadoc>
                    </googleJavaFormat>
                </java>
            </configuration>
        </plugin>
    </plugins>
```

```
    </build>
    <profiles>
        <profile>
            <id>compliance</id>
            <activation>
                <property>
                    <name>compliance</name>
                </property>
            </activation>
            <build>
                <plugins>
                    <plugin>
                        <groupId>org.honton.chas</groupId>
                        <artifactId>license-maven-plugin</artifactId>
                        <configuration>
                            <skipCompliance>true</skipCompliance>
                        </configuration>
                    </plugin>
                </plugins>
            </build>
        </profile>
    </profiles>
</project>
```

langchain4j-pgvector\src\main\java\dev\langchain4j\store\embedding\pgvector\PgVectorEmbeddingStore.java

```
package dev.langchain4j.store.embedding.pgvector;
import static dev.langchain4j.internal.Utils.isCollectionEmpty;
import static dev.langchain4j.internal.Utils.randomUUID;
import static dev.langchain4j.internal.ValidationUtils.ensureTrue;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.toList;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import com.pgvector.PGvector;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import java.lang.reflect.Type;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.UUID;
import lombok.NonNull;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
/**
 * PGVector EmbeddingStore Implementation
 * <p>
 * Only cosine similarity is used.
 * Only ivfflat index is used.
 */
public class PgVectorEmbeddingStore implements EmbeddingStore<TextSegment> {
    private static final Logger log =
LoggerFactory.getLogger(PgVectorEmbeddingStore.class);
    private static final Gson gson = new Gson();
    private final String dbHost;
    private final String dbUser;
    private final String dbPassword;
    private final String dbPort;
    private final String dbName;
    private final String namespace;
    /**
     * All args constructor for PgVectorEmbeddingStore Class
     *
     * @param dbHost The database host
     * @param dbUser The database user
     * @param dbPassword The database password
     * @param dbPort The database port
     * @param dbName The database name
     * @param namespace The vector namespace, as table name
```

```
     * @param dimension The vector dimension
     * @param useIndex Should use ivfflat index
     * @param indexListSize The ivfflat index size
     * @param createdTable Should create table
     * @param dropTableFirst Should drop table first, usually for testing
     */
    public PgVectorEmbeddingStore(
            @NonNull String dbHost,
            @NonNull String dbUser,
            @NonNull String dbPassword,
            @NonNull String dbPort,
            @NonNull String dbName,
            @NonNull String namespace,
            int dimension,
            boolean useIndex,
            int indexListSize,
            boolean createdTable,
            boolean dropTableFirst) {
        this.dbHost = dbHost;
        this.dbUser = dbUser;
        this.dbPassword = dbPassword;
        this.dbPort = dbPort;
        this.dbName = dbName;
        this.namespace = namespace;
        try (Connection connection = setupConnection()) {
            if (dropTableFirst) {
                Statement setupStmt = connection.createStatement();
                setupStmt.executeUpdate(String.format("DROP TABLE IF EXISTS
%s", namespace));
            }
            if (createdTable) {
                Statement createStmt = connection.createStatement();
                createStmt.executeUpdate(String.format(
                        "CREATE TABLE IF NOT EXISTS %s (vector_id UUID
PRIMARY KEY, embedding vector(%s), text TEXT NULL, metadata JSON NULL)",
                        namespace, dimension));
            }
            if (useIndex) {
                Statement indexStmt = connection.createStatement();
                indexStmt.executeUpdate(String.format(
                        "CREATE INDEX IF NOT EXISTS ON %s USING ivfflat
(embedding vector_cosine_ops) WITH (lists = %s)",
                        namespace, indexListSize));
            }
        } catch (SQLException e) {
            throw new ServiceException("Init Failure", e.getCause());
        }
    }
    Connection setupConnection() throws SQLException {
        Connection connection =
                DriverManager.getConnection(
                        String.format("jdbc:postgresql://%s:%s/%s",
this.dbHost, this.dbPort, this.dbName),
                        this.dbUser,
                        this.dbPassword);
        PGvector.addVectorType(connection);
        return connection;
    }
    /**
     * Adds a given embedding to the store.
     *
```

```
     * @param embedding The embedding to be added to the store.
     * @return The auto-generated ID associated with the added embedding.
     */
    @Override
    public String add(Embedding embedding) {
        String id = randomUUID();
        addInternal(id, embedding, null);
        return id;
    }
    /**
     * Adds a given embedding to the store.
     *
     * @param id        The unique identifier for the embedding to be added.
     * @param embedding The embedding to be added to the store.
     */
    @Override
    public void add(String id, Embedding embedding) {
        addInternal(id, embedding, null);
    }
    /**
     * Adds a given embedding and the corresponding content that has been
embedded to the store.
     *
     * @param embedding   The embedding to be added to the store.
     * @param textSegment Original content that was embedded.
     * @return The auto-generated ID associated with the added embedding.
     */
    @Override
    public String add(Embedding embedding, TextSegment textSegment) {
        String id = randomUUID();
        addInternal(id, embedding, textSegment);
        return id;
    }
    /**
     * Adds multiple embeddings to the store.
     *
     * @param embeddings A list of embeddings to be added to the store.
     * @return A list of auto-generated IDs associated with the added
embeddings.
     */
    @Override
    public List<String> addAll(List<Embedding> embeddings) {
        List<String> ids = embeddings.stream().map(ignored ->
randomUUID()).collect(toList());
        addAllInternal(ids, embeddings, null);
        return ids;
    }
    /**
     * Adds multiple embeddings and their corresponding contents that have
been embedded to the store.
     *
     * @param embeddings A list of embeddings to be added to the store.
     * @param embedded   A list of original contents that were embedded.
     * @return A list of auto-generated IDs associated with the added
embeddings.
     */
    @Override
    public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
embedded) {
        List<String> ids = embeddings.stream().map(ignored ->
randomUUID()).collect(toList());
```

```
        addAllInternal(ids, embeddings, embedded);
        return ids;
    }
    /**
     * Finds the most relevant (closest in space) embeddings to the provided
reference embedding.
     *
     * @param referenceEmbedding The embedding used as a reference. Returned
embeddings should be relevant (closest) to this one.
     * @param maxResults         The maximum number of embeddings to be
returned.
     * @param minScore           The minimum relevance score, ranging from 0
to 1 (inclusive).
     *                           Only embeddings with a score of this value
or higher will be returned.
     * @return A list of embedding matches.
     * Each embedding match includes a relevance score (derivative of cosine
distance),
     * ranging from 0 (not relevant) to 1 (highly relevant).
     */
    @Override
    public List<EmbeddingMatch<TextSegment>> findRelevant(
            Embedding referenceEmbedding, int maxResults, double minScore) {
        List<EmbeddingMatch<TextSegment>> result = new ArrayList<>();
        try (Connection conn = setupConnection()) {
            String referenceVector =
Arrays.toString(referenceEmbedding.vector());
            String query = String.format(
                    "WITH temp AS (SELECT (2 - (embedding <=> '%s')) / 2 AS
score, vector_id, embedding, text, metadata FROM %s) SELECT * FROM temp WHERE
score >= %s ORDER BY score desc LIMIT %s;",
                    referenceVector, this.namespace, minScore, maxResults);
            PreparedStatement selectStmt = conn.prepareStatement(query);
            ResultSet resultSet = selectStmt.executeQuery();
            while (resultSet.next()) {
                double score = resultSet.getDouble("score");
                String embeddingId = resultSet.getString("vector_id");
                PGvector vector = (PGvector) resultSet.getObject("embedding");
                Embedding embedding = new Embedding(vector.toArray());
                String text = resultSet.getString("text");
                String metadataJson =

Optional.ofNullable(resultSet.getString("metadata")).orElse("{}");
                Type type = new TypeToken<Map<String, String>>() {}.getType();
                Metadata metadata = new Metadata(new
HashMap<>(gson.fromJson(metadataJson, type)));
                if (text == null || text.isEmpty()) {
                    result.add(new EmbeddingMatch<>(score, embeddingId,
embedding, null));
                } else {
                    TextSegment textSegment = TextSegment.from(text,
metadata);
                    result.add(new EmbeddingMatch<>(score, embeddingId,
embedding, textSegment));
                }
            }
        } catch (SQLException e) {
            throw new ServiceException("Neighbor Query Failure",
e.getCause());
        }
        return result;
```

```
    }
    private void addInternal(String id, Embedding embedding, TextSegment
embedded) {
        addAllInternal(
                singletonList(id),
                singletonList(embedding),
                embedded == null ? null : singletonList(embedded));
    }
    private void addAllInternal(
            List<String> ids, List<Embedding> embeddings, List<TextSegment>
embedded) {
        if (isCollectionEmpty(ids) || isCollectionEmpty(embeddings)) {
            log.info("Empty embeddings - no ops");
            return;
        }
        ensureTrue(ids.size() == embeddings.size(), "ids size is not equal to
embeddings size");
        ensureTrue(
                embedded == null || embeddings.size() == embedded.size(),
                "embeddings size is not equal to embedded size");
        try (Connection conn = setupConnection()) {
            String query = String.format(
                    "INSERT INTO %s (vector_id, embedding, text, metadata)
VALUES (?, ?, ?, ?)" +
                            "ON CONFLICT (vector_id) DO UPDATE SET " +
                            "embedding = EXCLUDED.embedding," +
                            "text = EXCLUDED.text," +
                            "metadata = EXCLUDED.metadata;",
                    this.namespace);
            PreparedStatement upsertStmt = conn.prepareStatement(query);
            for (int i = 0; i < ids.size(); ++i) {
                upsertStmt.setObject(1, UUID.fromString(ids.get(i)));
                upsertStmt.setObject(2, new
PGvector(embeddings.get(i).vector()));
                if (embedded != null && embedded.get(i) != null) {
                    upsertStmt.setObject(3, embedded.get(i).text());
                    Map<String, String> metadata =
                            new HashMap<>(embedded.get(i).metadata().asMap());
                    upsertStmt.setObject(4, gson.toJson(metadata),
Types.OTHER);
                } else {
                    upsertStmt.setNull(3, Types.VARCHAR);
                    upsertStmt.setNull(4, Types.OTHER);
                }
                upsertStmt.addBatch();
            }
            upsertStmt.executeBatch();
        } catch (SQLException e) {
            throw new ServiceException("Insertion failure", e.getCause());
        }
    }
}
```

langchain4j-pgvector\src\main\java\dev\langchain4j\store\embedding\pgvector\ServiceException.java

```
package dev.langchain4j.store.embedding.pgvector;
public class ServiceException extends RuntimeException {
    public ServiceException() {
        super();
    }
    public ServiceException(String message) {
        super(message);
    }
    public ServiceException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

langchain4j-pgvector\src\test\java\dev\langchain4j\store\embedding\pgvector\PgVectorEmbeddingStoreLocalTest.java

```
package dev.langchain4j.store.embedding.pgvector;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
import static org.junit.jupiter.api.Assertions.assertTrue;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import java.util.List;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.testcontainers.containers.PostgreSQLContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.utility.DockerImageName;
@Disabled("Comment this line if you want to run locally")
public class PgVectorEmbeddingStoreLocalTest {
    /**
     * This test can be run without providing Envs to working PGVector
Postgres Instance
     */
    private static final DockerImageName pgVectorImage =
            DockerImageName.parse("ankane/
pgvector:v0.5.1").asCompatibleSubstituteFor("postgres");
    @Container
    private static final PostgreSQLContainer<?> pgVectorContainer =
            new PostgreSQLContainer<>(pgVectorImage);
    private final EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    ;
    private EmbeddingStore<TextSegment> embeddingStore;
    private PgVectorEmbeddingStore createEmbeddedStore(
            String namespace,
            int dimension,
            boolean useIndex,
            int indexListSize,
            boolean createTable,
            boolean dropTable) {
        String dbHost = pgVectorContainer.getHost();
        String dbPort = pgVectorContainer.getFirstMappedPort().toString();
        String dbUser = "test";
        String dbPassword = "test";
        String dbName = "test";
        return new PgVectorEmbeddingStore(
                dbHost,
                dbUser,
                dbPassword,
                dbPort,
                dbName,
```

```
                namespace,
                dimension,
                useIndex,
                indexListSize,
                createTable,
                dropTable);
    }
    @BeforeAll
    static void startContainer() {

pgVectorContainer.withExposedPorts(5432).withInitScript("init.sql").start();
    }
    @BeforeEach
    void initPgVectorEmbeddingStore() {
        embeddingStore = createEmbeddedStore("test_namespace", 384, false,
100, true, true);
    }
    @Test
    void should_aa_container_should_be_running() {
        assertTrue(pgVectorContainer.isRunning());
    }
    @Test
    void should_add_embedding() {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
```

```
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_embedding_with_segment_with_metadata() {
        TextSegment segment =
                TextSegment.from(randomUUID(), Metadata.from("test-key",
"test-value"));
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
                embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids =
                embeddingStore.addAll(
                        asList(firstEmbedding, secondEmbedding),
                        asList(firstSegment, secondSegment));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
                embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
```

```
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        List<EmbeddingMatch<TextSegment>> relevant =
                embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
                embeddingStore.findRelevant(firstEmbedding, 10,
secondMatch.score() - 0.01);
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
                embeddingStore.findRelevant(firstEmbedding, 10,
secondMatch.score());
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
                embeddingStore.findRelevant(firstEmbedding, 10,
secondMatch.score() + 0.01);
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        List<EmbeddingMatch<TextSegment>> relevant =
                embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score())
                .isCloseTo(
                        RelevanceScore.fromCosineSimilarity(
                                CosineSimilarity.between(embedding,
referenceEmbedding)),
```

```
                        withPercentage(1));
    }
}
```

langchain4j-pgvector\src\test\java\dev\langchain4j\store\embedding\pgvector\PgVectorEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.pgvector;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import java.util.List;
import java.util.Optional;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.EnabledIfEnvironmentVariable;
import org.junit.jupiter.api.condition.EnabledIfEnvironmentVariables;
@EnabledIfEnvironmentVariables({
    @EnabledIfEnvironmentVariable(named = "DB_HOST", matches = ".+"),
    @EnabledIfEnvironmentVariable(named = "DB_USER", matches = ".+"),
    @EnabledIfEnvironmentVariable(named = "DB_PASSWORD", matches = ".+"),
    @EnabledIfEnvironmentVariable(named = "DB_NAME", matches = ".+")
})
public class PgVectorEmbeddingStoreTest {
    private final EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    private EmbeddingStore<TextSegment> embeddingStore;
    private PgVectorEmbeddingStore createEmbeddedStore(
            String namespace,
            int dimension,
            boolean useIndex,
            int indexListSize,
            boolean createTable,
            boolean dropTable) {
        String dbHost = System.getenv("DB_HOST");
        String dbUser = System.getenv("DB_USER");
        String dbPassword = System.getenv("DB_PASSWORD");
        String dbPort =
Optional.ofNullable(System.getenv("DB_PORT")).orElse("5432");
        String dbName = System.getenv("DB_NAME");
        return new PgVectorEmbeddingStore(
                dbHost,
                dbUser,
                dbPassword,
                dbPort,
                dbName,
                namespace,
                dimension,
                useIndex,
                indexListSize,
                createTable,
                dropTable);
    }
    @BeforeEach
```

```
    void initPgVectorEmbeddingStore() {
        embeddingStore = createEmbeddedStore("test_namespace", 384, false,
100, true, true);
    }
    @Test
    void should_add_embedding() {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_embedding_with_segment_with_metadata() {
        TextSegment segment =
                TextSegment.from(randomUUID(), Metadata.from("test-key",
"test-value"));
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
```

```
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
                embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids =
                embeddingStore.addAll(
                        asList(firstEmbedding, secondEmbedding),
                        asList(firstSegment, secondSegment));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
                embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
```

```
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        List<EmbeddingMatch<TextSegment>> relevant =
                embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
                embeddingStore.findRelevant(firstEmbedding, 10,
secondMatch.score() - 0.01);
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
                embeddingStore.findRelevant(firstEmbedding, 10,
secondMatch.score());
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
                embeddingStore.findRelevant(firstEmbedding, 10,
secondMatch.score() + 0.01);
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        List<EmbeddingMatch<TextSegment>> relevant =
                embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score())
                .isCloseTo(
                        RelevanceScore.fromCosineSimilarity(
                                CosineSimilarity.between(embedding,
referenceEmbedding)),
                        withPercentage(1));
    }
}
```

langchain4j-pgvector\src\test\resources\init.sql

```
CREATE EXTENSION IF NOT EXISTS vector;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-pinecone</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Pinecone</name>
    <description>It uses the io.pinecone.pinecone-client library, which has a
proprietary license.</description>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>io.pinecone</groupId>
            <artifactId>pinecone-client</artifactId>
            <version>0.2.3</version>
            <exclusions>
                <!-- due to CVE-2022-41915 vulnerability -->
                <exclusion>
                    <groupId>io.netty</groupId>
                    <artifactId>netty-codec</artifactId>
                </exclusion>
                <!-- due to CVE-2023-44487 vulnerability -->
                <exclusion>
                    <groupId>io.netty</groupId>
                    <artifactId>netty-codec-http2</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-codec</artifactId>
            <version>${netty.version}</version>
        </dependency>
        <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-codec-http2</artifactId>
            <version>${netty.version}</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
```

```
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.honton.chas</groupId>
                <artifactId>license-maven-plugin</artifactId>
                <configuration>
                    <!-- the pinecone module has non-permissive licenses -->
                    <skipCompliance>true</skipCompliance>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <profiles>
        <profile>
            <id>compliance</id>
            <activation>
                <property>
                    <name>compliance</name>
                </property>
            </activation>
            <build>
                <plugins>
                    <plugin>
                        <groupId>org.honton.chas</groupId>
                        <artifactId>license-maven-plugin</artifactId>
                        <configuration>
                            <!-- the pinecone module has non-permissive
licenses -->
                            <skipCompliance>true</skipCompliance>
                        </configuration>
                    </plugin>
                </plugins>
            </build>
        </profile>
    </profiles>
</project>
```

langchain4j-pinecone\src\main\java\dev\langchain4j\store\embedding\pinecone\PineconeEmbeddingStore.java

```
package dev.langchain4j.store.embedding.pinecone;
import com.google.protobuf.Struct;
import com.google.protobuf.Value;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import io.pinecone.PineconeClient;
import io.pinecone.PineconeClientConfig;
import io.pinecone.PineconeConnection;
import io.pinecone.PineconeConnectionConfig;
import io.pinecone.proto.*;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Collections.emptyList;
import static java.util.Collections.singletonList;
import static java.util.Comparator.comparingDouble;
import static java.util.stream.Collectors.toList;
/**
 * Represents a <a href="https://www.pinecone.io/">Pinecone</a> index as an embedding store.
 * Current implementation assumes the index uses the cosine distance metric.
 * Does not support storing {@link dev.langchain4j.data.document.Metadata} yet.
 */
public class PineconeEmbeddingStore implements EmbeddingStore<TextSegment> {
    private static final String DEFAULT_NAMESPACE = "default"; // do not change, will break backward compatibility!
    private static final String METADATA_TEXT_SEGMENT = "text_segment"; // do not change, will break backward compatibility!
    private final PineconeConnection connection;
    private final String nameSpace;
    /**
     * Creates an instance of PineconeEmbeddingStore.
     *
     * @param apiKey      The Pinecone API key.
     * @param environment The environment (e.g., "northamerica-northeast1-gcp").
     * @param projectId   The ID of the project (e.g., "19a129b"). This is <b>not</b> a project name.
     *                    The ID can be found in the Pinecone URL: https://app.pinecone.io/organizations/.../projects/...:{projectId}/indexes.
     * @param index       The name of the index (e.g., "test").
     * @param nameSpace   (Optional) Namespace. If not provided, "default" will be used.
     */
    public PineconeEmbeddingStore(String apiKey,
                                  String environment,
                                  String projectId,
                                  String index,
                                  String nameSpace) {
        PineconeClientConfig configuration = new PineconeClientConfig()
                .withApiKey(apiKey)
```

```
                .withEnvironment(environment)
                .withProjectName(projectId);
        PineconeClient pineconeClient = new PineconeClient(configuration);
        PineconeConnectionConfig connectionConfig = new
PineconeConnectionConfig()
                .withIndexName(index);
        this.connection = pineconeClient.connect(connectionConfig);
        this.nameSpace = nameSpace == null ? DEFAULT_NAMESPACE : nameSpace;
    }
    @Override
    public String add(Embedding embedding) {
        String id = randomUUID();
        add(id, embedding);
        return id;
    }
    @Override
    public void add(String id, Embedding embedding) {
        addInternal(id, embedding, null);
    }
    @Override
    public String add(Embedding embedding, TextSegment textSegment) {
        String id = randomUUID();
        addInternal(id, embedding, textSegment);
        return id;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings) {
        List<String> ids = embeddings.stream()
                .map(ignored -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, null);
        return ids;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
textSegments) {
        List<String> ids = embeddings.stream()
                .map(ignored -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, textSegments);
        return ids;
    }
    private void addInternal(String id, Embedding embedding, TextSegment
textSegment) {
        addAllInternal(singletonList(id), singletonList(embedding),
textSegment == null ? null : singletonList(textSegment));
    }
    private void addAllInternal(List<String> ids, List<Embedding> embeddings,
List<TextSegment> textSegments) {
        UpsertRequest.Builder upsertRequestBuilder =
UpsertRequest.newBuilder()
                .setNamespace(nameSpace);
        for (int i = 0; i < embeddings.size(); i++) {
            String id = ids.get(i);
            Embedding embedding = embeddings.get(i);
            Vector.Builder vectorBuilder = Vector.newBuilder()
                    .setId(id)
                    .addAllValues(embedding.vectorAsList());
            if (textSegments != null) {
                vectorBuilder.setMetadata(Struct.newBuilder()
                        .putFields(METADATA_TEXT_SEGMENT, Value.newBuilder()
```

```
                                .setStringValue(textSegments.get(i).text())
                                .build()));
            }
            upsertRequestBuilder.addVectors(vectorBuilder.build());
        }
        connection.getBlockingStub().upsert(upsertRequestBuilder.build());
    }
    @Override
    public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
referenceEmbedding, int maxResults, double minScore) {
        QueryVector queryVector = QueryVector
                .newBuilder()
                .addAllValues(referenceEmbedding.vectorAsList())
                .setTopK(maxResults)
                .setNamespace(nameSpace)
                .build();
        QueryRequest queryRequest = QueryRequest
                .newBuilder()
                .addQueries(queryVector)
                .setTopK(maxResults)
                .build();
        List<String> matchedVectorIds = connection.getBlockingStub()
                .query(queryRequest)
                .getResultsList()
                .get(0)
                .getMatchesList()
                .stream()
                .map(ScoredVector::getId)
                .collect(toList());
        if (matchedVectorIds.isEmpty()) {
            return emptyList();
        }
        Collection<Vector> matchedVectors =
connection.getBlockingStub().fetch(FetchRequest.newBuilder()
                        .addAllIds(matchedVectorIds)
                        .setNamespace(nameSpace)
                        .build())
                .getVectorsMap()
                .values();
        List<EmbeddingMatch<TextSegment>> matches = matchedVectors.stream()
                .map(vector -> toEmbeddingMatch(vector, referenceEmbedding))
                .filter(match -> match.score() >= minScore)
                .sorted(comparingDouble(EmbeddingMatch::score))
                .collect(toList());
        Collections.reverse(matches);
        return matches;
    }
    private static EmbeddingMatch<TextSegment> toEmbeddingMatch(Vector
vector, Embedding referenceEmbedding) {
        Value textSegmentValue = vector.getMetadata()
                .getFieldsMap()
                .get(METADATA_TEXT_SEGMENT);
        Embedding embedding = Embedding.from(vector.getValuesList());
        double cosineSimilarity = CosineSimilarity.between(embedding,
referenceEmbedding);
        return new EmbeddingMatch<>(
                RelevanceScore.fromCosineSimilarity(cosineSimilarity),
                vector.getId(),
                embedding,
                textSegmentValue == null ? null :
TextSegment.from(textSegmentValue.getStringValue())
```

```
        );
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String apiKey;
        private String environment;
        private String projectId;
        private String index;
        private String nameSpace;
        /**
         * @param apiKey The Pinecone API key.
         */
        public Builder apiKey(String apiKey) {
            this.apiKey = apiKey;
            return this;
        }
        /**
         * @param environment The environment (e.g., "northamerica-northeast1-
gcp").
         */
        public Builder environment(String environment) {
            this.environment = environment;
            return this;
        }
        /**
         * @param projectId The ID of the project (e.g., "19a129b"). This is
<b>not</b> a project name.
         *                  The ID can be found in the Pinecone URL: https://
app.pinecone.io/organizations/.../projects/...:{projectId}/indexes.
         */
        public Builder projectId(String projectId) {
            this.projectId = projectId;
            return this;
        }
        /**
         * @param index The name of the index (e.g., "test").
         */
        public Builder index(String index) {
            this.index = index;
            return this;
        }
        /**
         * @param nameSpace (Optional) Namespace. If not provided, "default"
will be used.
         */
        public Builder nameSpace(String nameSpace) {
            this.nameSpace = nameSpace;
            return this;
        }
        public PineconeEmbeddingStore build() {
            return new PineconeEmbeddingStore(apiKey, environment, projectId,
index, nameSpace);
        }
    }
}
```

langchain4j-pinecone\src\test\java\dev\langchain4j\store\embedding\pinecone\PineconeEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.pinecone;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.EnabledIfEnvironmentVariable;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
@EnabledIfEnvironmentVariable(named = "PINECONE_API_KEY", matches = ".+")
class PineconeEmbeddingStoreTest {
    private final PineconeEmbeddingStore embeddingStore =
PineconeEmbeddingStore.builder()
            .apiKey(System.getenv("PINECONE_API_KEY"))
            .environment("asia-southeast1-gcp-free")
            .projectId("75dc67a")
            .index("test")
            .nameSpace(randomUUID())
            .build();
    private final EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    @Test
    void should_add_embedding() {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() {
```

```
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
        );
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
```

```
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
```

```
referenceEmbedding)),
                withPercentage(1)
        );
    }
}
```

langchain4j-redis\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://maven.apache.org/POM/4.0.0"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-redis</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Redis</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.tinylog</groupId>
            <artifactId>tinylog-impl</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.tinylog</groupId>
            <artifactId>slf4j-tinylog</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
            <scope>test</scope>
```

```
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-redis\src\main\java\dev\langchain4j\store\embedding\redis\MetricType.java

```
package dev.langchain4j.store.embedding.redis;
/**
 * Similarity metric used by Redis
 */
enum MetricType {
    /**
     * cosine similarity
     */
    COSINE,
    /**
     * inner product
     */
    IP,
    /**
     * euclidean distance
     */
    L2
}
```

langchain4j-redis\src\main\java\dev\langchain4j\store\embedding\redis\RedisEmbeddingStore.java

```
package dev.langchain4j.store.embedding.redis;
import com.google.gson.Gson;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import redis.clients.jedis.JedisPooled;
import redis.clients.jedis.Pipeline;
import redis.clients.jedis.json.Path2;
import redis.clients.jedis.search.*;
import java.util.*;
import java.util.stream.Collectors;
import static dev.langchain4j.internal.Utils.isCollectionEmpty;
import static dev.langchain4j.internal.Utils.randomUUID;
import static dev.langchain4j.internal.ValidationUtils.*;
import static
dev.langchain4j.store.embedding.redis.RedisSchema.SCORE_FIELD_NAME;
import static java.lang.String.format;
import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.toList;
import static redis.clients.jedis.search.IndexDefinition.Type.JSON;
import static redis.clients.jedis.search.RediSearchUtil.ToByteArray;
/**
 * Represents a <a href="https://redis.io/">Redis</a> index as an embedding
store.
 * Current implementation assumes the index uses the cosine distance metric.
 */
public class RedisEmbeddingStore implements EmbeddingStore<TextSegment> {
    private static final Logger log =
LoggerFactory.getLogger(RedisEmbeddingStore.class);
    private static final Gson GSON = new Gson();
    private final JedisPooled client;
    private final RedisSchema schema;
    /**
     * Creates an instance of RedisEmbeddingStore
     *
     * @param host               Redis Stack Server host
     * @param port               Redis Stack Server port
     * @param user               Redis Stack username (optional)
     * @param password           Redis Stack password (optional)
     * @param dimension          embedding vector dimension
     * @param metadataFieldsName metadata fields name (optional)
     */
    public RedisEmbeddingStore(String host,
                               Integer port,
                               String user,
                               String password,
                               Integer dimension,
                               List<String> metadataFieldsName) {
        ensureNotBlank(host, "host");
        ensureNotNull(port, "port");
        ensureNotNull(dimension, "dimension");
        this.client = user == null ? new JedisPooled(host, port) : new
```

```
JedisPooled(host, port, user, password);
        this.schema = RedisSchema.builder()
                .dimension(dimension)
                .metadataFieldsName(metadataFieldsName)
                .build();
        if (!isIndexExist(schema.getIndexName())) {
            createIndex(schema.getIndexName());
        }
    }
    @Override
    public String add(Embedding embedding) {
        String id = randomUUID();
        add(id, embedding);
        return id;
    }
    @Override
    public void add(String id, Embedding embedding) {
        addInternal(id, embedding, null);
    }
    @Override
    public String add(Embedding embedding, TextSegment textSegment) {
        String id = randomUUID();
        addInternal(id, embedding, textSegment);
        return id;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings) {
        List<String> ids = embeddings.stream()
                .map(ignored -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, null);
        return ids;
    }
    @Override
    public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
embedded) {
        List<String> ids = embeddings.stream()
                .map(ignored -> randomUUID())
                .collect(toList());
        addAllInternal(ids, embeddings, embedded);
        return ids;
    }
    @Override
    public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
referenceEmbedding, int maxResults, double minScore) {
        // Using KNN query on @vector field
        String queryTemplate = "*=>[ KNN %d @%s $BLOB AS %s ]";
        List<String> returnFields = new
ArrayList<>(schema.getMetadataFieldsName());
        returnFields.addAll(asList(schema.getVectorFieldName(),
schema.getScalarFieldName(), SCORE_FIELD_NAME));
        Query query = new Query(format(queryTemplate, maxResults,
schema.getVectorFieldName(), SCORE_FIELD_NAME))
                .addParam("BLOB", toByteArray(referenceEmbedding.vector()))
                .returnFields(returnFields.toArray(new String[0]))
                .setSortBy(SCORE_FIELD_NAME, true)
                .dialect(2);
        SearchResult result = client.ftSearch(schema.getIndexName(), query);
        List<Document> documents = result.getDocuments();
        return toEmbeddingMatch(documents, minScore);
    }
```

```
    private void createIndex(String indexName) {
        IndexDefinition indexDefinition = new IndexDefinition(JSON);
        indexDefinition.setPrefixes(schema.getPrefix());
        String res = client.ftCreate(indexName, FTCreateParams.createParams()
                .on(IndexDataType.JSON)
                .addPrefix(schema.getPrefix()), schema.toSchemaFields());
        if (!"OK".equals(res)) {
            if (log.isErrorEnabled()) {
                log.error("create index error, msg={}", res);
            }
            throw new RedisRequestFailedException("create index error, msg="
+ res);
        }
    }
    private boolean isIndexExist(String indexName) {
        Set<String> indexSets = client.ftList();
        return indexSets.contains(indexName);
    }
    private void addInternal(String id, Embedding embedding, TextSegment
embedded) {
        addAllInternal(singletonList(id), singletonList(embedding), embedded
== null ? null : singletonList(embedded));
    }
    private void addAllInternal(List<String> ids, List<Embedding> embeddings,
List<TextSegment> embedded) {
        if (isCollectionEmpty(ids) || isCollectionEmpty(embeddings)) {
            log.info("do not add empty embeddings to redis");
            return;
        }
        ensureTrue(ids.size() == embeddings.size(), "ids size is not equal to
embeddings size");
        ensureTrue(embedded == null || embeddings.size() == embedded.size(),
"embeddings size is not equal to embedded size");
        Pipeline pipeline = client.pipelined();
        int size = ids.size();
        for (int i = 0; i < size; i++) {
            String id = ids.get(i);
            Embedding embedding = embeddings.get(i);
            TextSegment textSegment = embedded == null ? null :
embedded.get(i);
            Map<String, Object> fields = new HashMap<>();
            fields.put(schema.getVectorFieldName(), embedding.vector());
            if (textSegment != null) {
                // do not check metadata key is included in
RedisSchema#metadataFieldsName
                fields.put(schema.getScalarFieldName(), textSegment.text());
                fields.putAll(textSegment.metadata().asMap());
            }
            String key = schema.getPrefix() + id;
            pipeline.jsonSetWithEscape(key, Path2.of("$"), fields);
        }
        List<Object> responses = pipeline.syncAndReturnAll();
        Optional<Object> errResponse = responses.stream().filter(response -
> !"OK".equals(response)).findAny();
        if (errResponse.isPresent()) {
            if (log.isErrorEnabled()) {
                log.error("add embedding failed, msg={}", errResponse.get());
            }
            throw new RedisRequestFailedException("add embedding failed,
msg=" + errResponse.get());
        }
```

```
    }
    private List<EmbeddingMatch<TextSegment>> toEmbeddingMatch(List<Document>
documents, double minScore) {
        if (documents == null || documents.isEmpty()) {
            return new ArrayList<>();
        }
        return documents.stream()
                .map(document -> {
                    double score = (2 -
Double.parseDouble(document.getString(SCORE_FIELD_NAME))) / 2;
                    String id =
document.getId().substring(schema.getPrefix().length());
                    String text =
document.hasProperty(schema.getScalarFieldName()) ?
document.getString(schema.getScalarFieldName()) : null;
                    TextSegment embedded = null;
                    if (text != null) {
                        List<String> metadataFieldsName =
schema.getMetadataFieldsName();
                        Map<String, String> metadata =
metadataFieldsName.stream()
                                .filter(document::hasProperty)
                                .collect(Collectors.toMap(metadataFieldName -
> metadataFieldName, document::getString));
                        embedded = new TextSegment(text, new
Metadata(metadata));
                    }
                    Embedding embedding = new
Embedding(GSON.fromJson(document.getString(schema.getVectorFieldName()),
float[].class));
                    return new EmbeddingMatch<>(score, id, embedding,
embedded);
                })
                .filter(embeddingMatch -> embeddingMatch.score() >= minScore)
                .collect(toList());
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String host;
        private Integer port;
        private String user;
        private String password;
        private Integer dimension;
        private List<String> metadataFieldsName = new ArrayList<>();
        /**
         * @param host Redis Stack host
         */
        public Builder host(String host) {
            this.host = host;
            return this;
        }
        /**
         * @param port Redis Stack port
         */
        public Builder port(Integer port) {
            this.port = port;
            return this;
        }
        /**
```

```
         * @param user Redis Stack username (optional)
         */
        public Builder user(String user) {
            this.user = user;
            return this;
        }
        /**
         * @param password Redis Stack password (optional)
         */
        public Builder password(String password) {
            this.password = password;
            return this;
        }
        /**
         * @param dimension embedding vector dimension
         * @return builder
         */
        public Builder dimension(Integer dimension) {
            this.dimension = dimension;
            return this;
        }
        /**
         * @param metadataFieldsName metadata fields name (optional)
         */
        public Builder metadataFieldsName(List<String> metadataFieldsName) {
            this.metadataFieldsName = metadataFieldsName;
            return this;
        }
        public RedisEmbeddingStore build() {
            return new RedisEmbeddingStore(host, port, user, password,
dimension, metadataFieldsName);
        }
    }
}
```

langchain4j-redis\src\main\java\dev\langchain4j\store\embedding\redis\RedisRequestFailedException.java

```
package dev.langchain4j.store.embedding.redis;
public class RedisRequestFailedException extends RuntimeException {
    public RedisRequestFailedException() {
        super();
    }
    public RedisRequestFailedException(String message) {
        super(message);
    }
    public RedisRequestFailedException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

langchain4j-redis\src\main\java\dev\langchain4j\store\embedding\redis\RedisSchema.java

```
package dev.langchain4j.store.embedding.redis;
import lombok.AllArgsConstructor;
import lombok.Builder;
import redis.clients.jedis.search.schemafields.SchemaField;
import redis.clients.jedis.search.schemafields.TextField;
import redis.clients.jedis.search.schemafields.VectorField;
import redis.clients.jedis.search.schemafields.VectorField.VectorAlgorithm;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static dev.langchain4j.store.embedding.redis.MetricType.COSINE;
import static
redis.clients.jedis.search.schemafields.VectorField.VectorAlgorithm.HNSW;
/**
 * Redis Schema Description
 */
@Builder
@AllArgsConstructor
class RedisSchema {
    public static final String SCORE_FIELD_NAME = "vector_score";
    private static final String JSON_PATH_PREFIX = "$.";
    private static final VectorAlgorithm DEFAULT_VECTOR_ALGORITHM = HNSW;
    private static final MetricType DEFAULT_METRIC_TYPE = COSINE;
    /* Redis schema field settings */
    @Builder.Default
    private String indexName = "embedding-index";
    @Builder.Default
    private String prefix = "embedding:";
    @Builder.Default
    private String vectorFieldName = "vector";
    @Builder.Default
    private String scalarFieldName = "text";
    @Builder.Default
    private List<String> metadataFieldsName = new ArrayList<>();
    /* Vector field settings */
    @Builder.Default
    private VectorAlgorithm vectorAlgorithm = DEFAULT_VECTOR_ALGORITHM;
    private int dimension;
    @Builder.Default
    private MetricType metricType = DEFAULT_METRIC_TYPE;
    public RedisSchema(int dimension) {
        this.dimension = dimension;
    }
    public SchemaField[] toSchemaFields() {
        Map<String, Object> vectorAttrs = new HashMap<>();
        vectorAttrs.put("DIM", dimension);
        vectorAttrs.put("DISTANCE_METRIC", metricType.name());
        vectorAttrs.put("TYPE", "FLOAT32");
        vectorAttrs.put("INITIAL_CAP", 5);
        List<SchemaField> fields = new ArrayList<>();
        fields.add(TextField.of(JSON_PATH_PREFIX +
scalarFieldName).as(scalarFieldName).weight(1.0));
        fields.add(VectorField.builder()
                .fieldName(JSON_PATH_PREFIX + vectorFieldName)
                .algorithm(vectorAlgorithm)
                .attributes(vectorAttrs)
```

```
                .as(vectorFieldName)
                .build());
        if (metadataFieldsName != null && !metadataFieldsName.isEmpty()) {
            for (String metadataFieldName : metadataFieldsName) {
                fields.add(TextField.of(JSON_PATH_PREFIX +
metadataFieldName).as(metadataFieldName).weight(1.0));
            }
        }
        return fields.toArray(new SchemaField[0]);
    }
    public String getIndexName() {
        return indexName;
    }
    public String getPrefix() {
        return prefix;
    }
    public String getVectorFieldName() {
        return vectorFieldName;
    }
    public String getScalarFieldName() {
        return scalarFieldName;
    }
    public List<String> getMetadataFieldsName() {
        return metadataFieldsName;
    }
}
```

langchain4j-redis\src\test\java\dev\langchain4j\store\embedding\redis\RedisEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.redis;
import dev.langchain4j.data.document.Metadata;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import redis.clients.jedis.JedisPooled;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
@Disabled("needs Redis running locally")
class RedisEmbeddingStoreTest {
    /**
     * First start Redis locally:
     * docker pull redis/redis-stack:latest
     * docker run -d -p 6379:6379 -p 8001:8001 redis/redis-stack:latest
     */
    private static final String HOST = "localhost";
    private static final int PORT = 6379;
    private static final String METADATA_KEY = "test-key";
    private EmbeddingStore<TextSegment> embeddingStore;
    private final EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    @BeforeEach
    void initEmptyRedisEmbeddingStore() {
        flushDB();
        embeddingStore = RedisEmbeddingStore.builder()
                .host(HOST)
                .port(PORT)
                .dimension(384)
                .build();
    }
    private static void flushDB() {
        try (JedisPooled jedis = new JedisPooled(HOST, PORT)) {
            jedis.flushDB();
        }
    }
    @Test
    void should_add_embedding() {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
```

```
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() {
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_embedding_with_segment_with_metadata() {
        flushDB();
        embeddingStore = RedisEmbeddingStore.builder()
                .host(HOST)
                .port(PORT)
                .dimension(384)
                .metadataFieldsName(singletonList(METADATA_KEY))
                .build();
        TextSegment segment = TextSegment.from(randomUUID(),
Metadata.from(METADATA_KEY, "test-value"));
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
```

```
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
        );
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
```

```
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
referenceEmbedding)),
                withPercentage(1)
        );
    }
}
```

langchain4j-spring-boot-starter\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-spring-boot-starter</artifactId>
    <packaging>jar</packaging>
    <name>langchain4j-spring-boot-starter</name>
    <description>Spring Boot Starter for LangChain4j</description>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-open-ai</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-local-ai</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-hugging-face</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\ChatModel.java

```
package dev.langchain4j;
class ChatModel {
    private ModelProvider provider;
    private OpenAi openAi;
    private HuggingFace huggingFace;
    private LocalAi localAi;
    public ModelProvider getProvider() {
        return provider;
    }
    public void setProvider(ModelProvider provider) {
        this.provider = provider;
    }
    public OpenAi getOpenAi() {
        return openAi;
    }
    public void setOpenAi(OpenAi openAi) {
        this.openAi = openAi;
    }
    public HuggingFace getHuggingFace() {
        return huggingFace;
    }
    public void setHuggingFace(HuggingFace huggingFace) {
        this.huggingFace = huggingFace;
    }
    public LocalAi getLocalAi() {
        return localAi;
    }
    public void setLocalAi(LocalAi localAi) {
        this.localAi = localAi;
    }
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\EmbeddingModel.java

```
package dev.langchain4j;
class EmbeddingModel {
    private ModelProvider provider;
    private OpenAi openAi;
    private HuggingFace huggingFace;
    private LocalAi localAi;
    public ModelProvider getProvider() {
        return provider;
    }
    public void setProvider(ModelProvider provider) {
        this.provider = provider;
    }
    public OpenAi getOpenAi() {
        return openAi;
    }
    public void setOpenAi(OpenAi openAi) {
        this.openAi = openAi;
    }
    public HuggingFace getHuggingFace() {
        return huggingFace;
    }
    public void setHuggingFace(HuggingFace huggingFace) {
        this.huggingFace = huggingFace;
    }
    public LocalAi getLocalAi() {
        return localAi;
    }
    public void setLocalAi(LocalAi localAi) {
        this.localAi = localAi;
    }
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\HuggingFace.java

```
package dev.langchain4j;
import java.time.Duration;
class HuggingFace {
    private String accessToken;
    private String modelId;
    private Duration timeout;
    private Double temperature;
    private Integer maxNewTokens;
    private Boolean returnFullText;
    private Boolean waitForModel;
    public String getAccessToken() {
        return accessToken;
    }
    public void setAccessToken(String accessToken) {
        this.accessToken = accessToken;
    }
    public String getModelId() {
        return modelId;
    }
    public void setModelId(String modelId) {
        this.modelId = modelId;
    }
    public Duration getTimeout() {
        return timeout;
    }
    public void setTimeout(Duration timeout) {
        this.timeout = timeout;
    }
    public Double getTemperature() {
        return temperature;
    }
    public void setTemperature(Double temperature) {
        this.temperature = temperature;
    }
    public Integer getMaxNewTokens() {
        return maxNewTokens;
    }
    public void setMaxNewTokens(Integer maxNewTokens) {
        this.maxNewTokens = maxNewTokens;
    }
    public Boolean getReturnFullText() {
        return returnFullText;
    }
    public void setReturnFullText(Boolean returnFullText) {
        this.returnFullText = returnFullText;
    }
    public Boolean getWaitForModel() {
        return waitForModel;
    }
    public void setWaitForModel(Boolean waitForModel) {
        this.waitForModel = waitForModel;
    }
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\LangChain4jAutoConfiguration.java

```
package dev.langchain4j;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.huggingface.HuggingFaceChatModel;
import dev.langchain4j.model.huggingface.HuggingFaceEmbeddingModel;
import dev.langchain4j.model.huggingface.HuggingFaceLanguageModel;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.localai.LocalAiChatModel;
import dev.langchain4j.model.localai.LocalAiEmbeddingModel;
import dev.langchain4j.model.localai.LocalAiLanguageModel;
import dev.langchain4j.model.moderation.ModerationModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.openai.OpenAiEmbeddingModel;
import dev.langchain4j.model.openai.OpenAiLanguageModel;
import dev.langchain4j.model.openai.OpenAiModerationModel;
import org.springframework.boot.autoconfigure.AutoConfiguration;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Lazy;
import static dev.langchain4j.ModelProvider.OPEN_AI;
import static
dev.langchain4j.exception.IllegalConfigurationException.illegalConfiguration;
import static dev.langchain4j.internal.Utils.isNullOrBlank;
@AutoConfiguration
@EnableConfigurationProperties(LangChain4jProperties.class)
public class LangChain4jAutoConfiguration {
    @Bean
    @Lazy
    @ConditionalOnMissingBean
    ChatLanguageModel chatLanguageModel(LangChain4jProperties properties) {
        if (properties.getChatModel() == null) {
            throw illegalConfiguration("\n\nPlease define 'langchain4j.chat-
model' properties, for example:\n"
                    + "langchain4j.chat-model.provider = openai\n"
                    + "langchain4j.chat-model.openai.api-key = sk-...\n");
        }
        switch (properties.getChatModel().getProvider()) {
            case OPEN_AI:
                OpenAi openAi = properties.getChatModel().getOpenAi();
                if (openAi == null || isNullOrBlank(openAi.getApiKey())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.chat-model.openai.api-key' property");
                }
                return OpenAiChatModel.builder()
                        .baseUrl(openAi.getBaseUrl())
                        .apiKey(openAi.getApiKey())
                        .modelName(openAi.getModelName())
                        .temperature(openAi.getTemperature())
                        .topP(openAi.getTopP())
                        .maxTokens(openAi.getMaxTokens())
                        .presencePenalty(openAi.getPresencePenalty())
                        .frequencyPenalty(openAi.getFrequencyPenalty())
                        .timeout(openAi.getTimeout())
                        .maxRetries(openAi.getMaxRetries())
```

```
                        .logRequests(openAi.getLogRequests())
                        .logResponses(openAi.getLogResponses())
                        .build();
            case HUGGING_FACE:
                HuggingFace huggingFace =
properties.getChatModel().getHuggingFace();
                if (huggingFace == null ||
isNullOrBlank(huggingFace.getAccessToken())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.chat-model.huggingface.access-token' property");
                }
                return HuggingFaceChatModel.builder()
                        .accessToken(huggingFace.getAccessToken())
                        .modelId(huggingFace.getModelId())
                        .timeout(huggingFace.getTimeout())
                        .temperature(huggingFace.getTemperature())
                        .maxNewTokens(huggingFace.getMaxNewTokens())
                        .returnFullText(huggingFace.getReturnFullText())
                        .waitForModel(huggingFace.getWaitForModel())
                        .build();
            case LOCAL_AI:
                LocalAi localAi = properties.getChatModel().getLocalAi();
                if (localAi == null || isNullOrBlank(localAi.getBaseUrl())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.chat-model.localai.base-url' property");
                }
                if (isNullOrBlank(localAi.getModelName())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.chat-model.localai.model-name' property");
                }
                return LocalAiChatModel.builder()
                        .baseUrl(localAi.getBaseUrl())
                        .modelName(localAi.getModelName())
                        .temperature(localAi.getTemperature())
                        .topP(localAi.getTopP())
                        .maxTokens(localAi.getMaxTokens())
                        .timeout(localAi.getTimeout())
                        .maxRetries(localAi.getMaxRetries())
                        .logRequests(localAi.getLogRequests())
                        .logResponses(localAi.getLogResponses())
                        .build();
            default:
                throw illegalConfiguration("Unsupported chat model provider:
%s", properties.getChatModel().getProvider());
        }
    }
    @Bean
    @Lazy
    @ConditionalOnMissingBean
    LanguageModel languageModel(LangChain4jProperties properties) {
        if (properties.getLanguageModel() == null) {
            throw illegalConfiguration("\n\nPlease define
'langchain4j.language-model' properties, for example:\n"
                    + "langchain4j.language-model.provider = openai\n"
                    + "langchain4j.language-model.openai.api-key = sk-...\n");
        }
        switch (properties.getLanguageModel().getProvider()) {
            case OPEN_AI:
                OpenAi openAi = properties.getLanguageModel().getOpenAi();
                if (openAi == null || isNullOrBlank(openAi.getApiKey())) {
                    throw illegalConfiguration("\n\nPlease define
```

```
'langchain4j.language-model.openai.api-key' property");
                }
                return OpenAiLanguageModel.builder()
                        .apiKey(openAi.getApiKey())
                        .modelName(openAi.getModelName())
                        .temperature(openAi.getTemperature())
                        .timeout(openAi.getTimeout())
                        .maxRetries(openAi.getMaxRetries())
                        .logRequests(openAi.getLogRequests())
                        .logResponses(openAi.getLogResponses())
                        .build();
            case HUGGING_FACE:
                HuggingFace huggingFace =
properties.getLanguageModel().getHuggingFace();
                if (huggingFace == null ||
isNullOrBlank(huggingFace.getAccessToken())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.language-model.huggingface.access-token' property");
                }
                return HuggingFaceLanguageModel.builder()
                        .accessToken(huggingFace.getAccessToken())
                        .modelId(huggingFace.getModelId())
                        .timeout(huggingFace.getTimeout())
                        .temperature(huggingFace.getTemperature())
                        .maxNewTokens(huggingFace.getMaxNewTokens())
                        .returnFullText(huggingFace.getReturnFullText())
                        .waitForModel(huggingFace.getWaitForModel())
                        .build();
            case LOCAL_AI:
                LocalAi localAi = properties.getLanguageModel().getLocalAi();
                if (localAi == null || isNullOrBlank(localAi.getBaseUrl())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.language-model.localai.base-url' property");
                }
                if (isNullOrBlank(localAi.getModelName())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.language-model.localai.model-name' property");
                }
                return LocalAiLanguageModel.builder()
                        .baseUrl(localAi.getBaseUrl())
                        .modelName(localAi.getModelName())
                        .temperature(localAi.getTemperature())
                        .topP(localAi.getTopP())
                        .maxTokens(localAi.getMaxTokens())
                        .timeout(localAi.getTimeout())
                        .maxRetries(localAi.getMaxRetries())
                        .logRequests(localAi.getLogRequests())
                        .logResponses(localAi.getLogResponses())
                        .build();
            default:
                throw illegalConfiguration("Unsupported language model
provider: %s", properties.getLanguageModel().getProvider());
        }
    }
    @Bean
    @Lazy
    @ConditionalOnMissingBean
    EmbeddingModel embeddingModel(LangChain4jProperties properties) {
        if (properties.getEmbeddingModel() == null ||
properties.getEmbeddingModel().getProvider() == null) {
            throw illegalConfiguration("\n\nPlease define
```

```
'langchain4j.embedding-model' properties, for example:\n"
                    + "langchain4j.embedding-model.provider = openai\n"
                    + "langchain4j.embedding-model.openai.api-key = sk-...
\n");
        }
        switch (properties.getEmbeddingModel().getProvider()) {
            case OPEN_AI:
                OpenAi openAi = properties.getEmbeddingModel().getOpenAi();
                if (openAi == null || isNullOrBlank(openAi.getApiKey())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.embedding-model.openai.api-key' property"); // TODO exception
type
                }
                return OpenAiEmbeddingModel.builder()
                        .apiKey(openAi.getApiKey())
                        .modelName(openAi.getModelName())
                        .timeout(openAi.getTimeout())
                        .maxRetries(openAi.getMaxRetries())
                        .logRequests(openAi.getLogRequests())
                        .logResponses(openAi.getLogResponses())
                        .build();
            case HUGGING_FACE:
                HuggingFace huggingFace =
properties.getEmbeddingModel().getHuggingFace();
                if (huggingFace == null ||
isNullOrBlank(huggingFace.getAccessToken())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.embedding-model.huggingface.access-token' property");
                }
                return HuggingFaceEmbeddingModel.builder()
                        .accessToken(huggingFace.getAccessToken())
                        .modelId(huggingFace.getModelId())
                        .waitForModel(huggingFace.getWaitForModel())
                        .timeout(huggingFace.getTimeout())
                        .build();
            case LOCAL_AI:
                LocalAi localAi = properties.getEmbeddingModel().getLocalAi();
                if (localAi == null || isNullOrBlank(localAi.getBaseUrl())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.embedding-model.localai.base-url' property");
                }
                if (isNullOrBlank(localAi.getModelName())) {
                    throw illegalConfiguration("\n\nPlease define
'langchain4j.embedding-model.localai.model-name' property");
                }
                return LocalAiEmbeddingModel.builder()
                        .baseUrl(localAi.getBaseUrl())
                        .modelName(localAi.getModelName())
                        .timeout(localAi.getTimeout())
                        .maxRetries(localAi.getMaxRetries())
                        .logRequests(localAi.getLogRequests())
                        .logResponses(localAi.getLogResponses())
                        .build();
            default:
                throw illegalConfiguration("Unsupported embedding model
provider: %s", properties.getEmbeddingModel().getProvider());
        }
    }
    @Bean
    @Lazy
    @ConditionalOnMissingBean
```

```
    ModerationModel moderationModel(LangChain4jProperties properties) {
        if (properties.getModerationModel() == null) {
            throw illegalConfiguration("\n\nPlease define
'langchain4j.moderation-model' properties, for example:\n"
                    + "langchain4j.moderation-model.provider = openai\n"
                    + "langchain4j.moderation-model.openai.api-key = sk-...
\n");
        }
        if (properties.getModerationModel().getProvider() != OPEN_AI) {
            throw illegalConfiguration("Unsupported moderation model
provider: %s", properties.getModerationModel().getProvider());
        }
        OpenAi openAi = properties.getModerationModel().getOpenAi();
        if (openAi == null || isNullOrBlank(openAi.getApiKey())) {
            throw illegalConfiguration("\n\nPlease define
'langchain4j.moderation-model.openai.api-key' property");
        }
        return OpenAiModerationModel.builder()
                .apiKey(openAi.getApiKey())
                .modelName(openAi.getModelName())
                .timeout(openAi.getTimeout())
                .maxRetries(openAi.getMaxRetries())
                .logRequests(openAi.getLogRequests())
                .logResponses(openAi.getLogResponses())
                .build();
    }
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\LangChain4jProperties.java

```
package dev.langchain4j;
import org.springframework.boot.context.properties.ConfigurationProperties;
@ConfigurationProperties(prefix = "langchain4j")
public class LangChain4jProperties {
    private ChatModel chatModel;
    private LanguageModel languageModel;
    private EmbeddingModel embeddingModel;
    private ModerationModel moderationModel;
    public ChatModel getChatModel() {
        return chatModel;
    }
    public void setChatModel(ChatModel chatModel) {
        this.chatModel = chatModel;
    }
    public LanguageModel getLanguageModel() {
        return languageModel;
    }
    public void setLanguageModel(LanguageModel languageModel) {
        this.languageModel = languageModel;
    }
    public EmbeddingModel getEmbeddingModel() {
        return embeddingModel;
    }
    public void setEmbeddingModel(EmbeddingModel embeddingModel) {
        this.embeddingModel = embeddingModel;
    }
    public ModerationModel getModerationModel() {
        return moderationModel;
    }
    public void setModerationModel(ModerationModel moderationModel) {
        this.moderationModel = moderationModel;
    }
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\LanguageModel.java

```
package dev.langchain4j;
class LanguageModel {
    private ModelProvider provider;
    private OpenAi openAi;
    private HuggingFace huggingFace;
    private LocalAi localAi;
    public ModelProvider getProvider() {
        return provider;
    }
    public void setProvider(ModelProvider provider) {
        this.provider = provider;
    }
    public OpenAi getOpenAi() {
        return openAi;
    }
    public void setOpenAi(OpenAi openAi) {
        this.openAi = openAi;
    }
    public HuggingFace getHuggingFace() {
        return huggingFace;
    }
    public void setHuggingFace(HuggingFace huggingFace) {
        this.huggingFace = huggingFace;
    }
    public LocalAi getLocalAi() {
        return localAi;
    }
    public void setLocalAi(LocalAi localAi) {
        this.localAi = localAi;
    }
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\LocalAi.java

```
package dev.langchain4j;
import java.time.Duration;
class LocalAi {
    private String baseUrl;
    private String modelName;
    private Double temperature;
    private Double topP;
    private Integer maxTokens;
    private Duration timeout;
    private Integer maxRetries;
    private Boolean logRequests;
    private Boolean logResponses;
    public String getBaseUrl() {
        return baseUrl;
    }
    public void setBaseUrl(String baseUrl) {
        this.baseUrl = baseUrl;
    }
    public String getModelName() {
        return modelName;
    }
    public void setModelName(String modelName) {
        this.modelName = modelName;
    }
    public Double getTemperature() {
        return temperature;
    }
    public void setTemperature(Double temperature) {
        this.temperature = temperature;
    }
    public Double getTopP() {
        return topP;
    }
    public void setTopP(Double topP) {
        this.topP = topP;
    }
    public Integer getMaxTokens() {
        return maxTokens;
    }
    public void setMaxTokens(Integer maxTokens) {
        this.maxTokens = maxTokens;
    }
    public Duration getTimeout() {
        return timeout;
    }
    public void setTimeout(Duration timeout) {
        this.timeout = timeout;
    }
    public Integer getMaxRetries() {
        return maxRetries;
    }
    public void setMaxRetries(Integer maxRetries) {
        this.maxRetries = maxRetries;
    }
    public Boolean getLogRequests() {
        return logRequests;
    }
    public void setLogRequests(Boolean logRequests) {
```

```
        this.logRequests = logRequests;
    }
    public Boolean getLogResponses() {
        return logResponses;
    }
    public void setLogResponses(Boolean logResponses) {
        this.logResponses = logResponses;
    }
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\ModelProvider.java

```
package dev.langchain4j;
enum ModelProvider {
    OPEN_AI,
    HUGGING_FACE,
    LOCAL_AI
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\ModerationModel.java

```
package dev.langchain4j;
class ModerationModel {
    private ModelProvider provider;
    private OpenAi openAi;
    public ModelProvider getProvider() {
        return provider;
    }
    public void setProvider(ModelProvider provider) {
        this.provider = provider;
    }
    public OpenAi getOpenAi() {
        return openAi;
    }
    public void setOpenAi(OpenAi openAi) {
        this.openAi = openAi;
    }
}
```

langchain4j-spring-boot-starter\src\main\java\dev\langchain4j\OpenAi.java

```
package dev.langchain4j;
import java.time.Duration;
class OpenAi {
    private String baseUrl;
    private String apiKey;
    private String modelName;
    private Double temperature;
    private Double topP;
    private Integer maxTokens;
    private Double presencePenalty;
    private Double frequencyPenalty;
    private Duration timeout;
    private Integer maxRetries;
    private Boolean logRequests;
    private Boolean logResponses;
    public String getBaseUrl() {
        return baseUrl;
    }
    public void setBaseUrl(String baseUrl) {
        this.baseUrl = baseUrl;
    }
    public String getApiKey() {
        return apiKey;
    }
    public void setApiKey(String apiKey) {
        this.apiKey = apiKey;
    }
    public String getModelName() {
        return modelName;
    }
    public void setModelName(String modelName) {
        this.modelName = modelName;
    }
    public Double getTemperature() {
        return temperature;
    }
    public void setTemperature(Double temperature) {
        this.temperature = temperature;
    }
    public Double getTopP() {
        return topP;
    }
    public void setTopP(Double topP) {
        this.topP = topP;
    }
    public Integer getMaxTokens() {
        return maxTokens;
    }
    public void setMaxTokens(Integer maxTokens) {
        this.maxTokens = maxTokens;
    }
    public Double getPresencePenalty() {
        return presencePenalty;
    }
    public void setPresencePenalty(Double presencePenalty) {
        this.presencePenalty = presencePenalty;
    }
    public Double getFrequencyPenalty() {
```

```
        return frequencyPenalty;
    }
    public void setFrequencyPenalty(Double frequencyPenalty) {
        this.frequencyPenalty = frequencyPenalty;
    }
    public Duration getTimeout() {
        return timeout;
    }
    public void setTimeout(Duration timeout) {
        this.timeout = timeout;
    }
    public Integer getMaxRetries() {
        return maxRetries;
    }
    public void setMaxRetries(Integer maxRetries) {
        this.maxRetries = maxRetries;
    }
    public Boolean getLogRequests() {
        return logRequests;
    }
    public void setLogRequests(Boolean logRequests) {
        this.logRequests = logRequests;
    }
    public Boolean getLogResponses() {
        return logResponses;
    }
    public void setLogResponses(Boolean logResponses) {
        this.logResponses = logResponses;
    }
}
```

```
 langchain4j-spring-boot-starter\src\main\resources\META-
INF\spring\org.springframework.boot.autoconfigure.AutoConfiguration.imports

dev.langchain4j.LangChain4jAutoConfiguration
```

langchain4j-spring-boot-starter\src\main\resources\META-INF\spring.factories

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=dev.langchain4j.LangChain4jAutoConfiguration
```

langchain4j-vertex-ai\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-vertex-ai</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Vertex AI</name>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.google.cloud</groupId>
            <artifactId>google-cloud-aiplatform</artifactId>
            <version>3.24.0</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-vertex-ai\src\main\java\dev\langchain4j\model\vertexai\VertexAiChatInstance.java

```
package dev.langchain4j.model.vertexai;
import java.util.List;
class VertexAiChatInstance {
    private final String context;
    private final List<Message> messages;
    VertexAiChatInstance(String context, List<Message> messages) {
        this.context = context;
        this.messages = messages;
    }
    static class Message {
        private final String author;
        private final String content;
        Message(String author, String content) {
            this.author = author;
            this.content = content;
        }
    }
}
```

langchain4j-vertex-ai\src\main\java\dev\langchain4j\model\vertexai\VertexAiChatModel.java

```
package dev.langchain4j.model.vertexai;
import com.google.cloud.aiplatform.v1.EndpointName;
import com.google.cloud.aiplatform.v1.PredictResponse;
import com.google.cloud.aiplatform.v1.PredictionServiceClient;
import com.google.cloud.aiplatform.v1.PredictionServiceSettings;
import com.google.protobuf.Value;
import com.google.protobuf.util.JsonFormat;
import dev.langchain4j.agent.tool.ToolSpecification;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import java.io.IOException;
import java.util.List;
import static com.google.protobuf.Value.newBuilder;
import static dev.langchain4j.data.message.ChatMessageType.AI;
import static dev.langchain4j.data.message.ChatMessageType.SYSTEM;
import static dev.langchain4j.data.message.ChatMessageType.USER;
import static dev.langchain4j.internal.Json.toJson;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;
/**
 * Represents a Google Vertex AI language model with a chat completion
interface, such as chat-bison.
 * See details <a href="https://cloud.google.com/vertex-ai/docs/generative-ai/
chat/chat-prompts">here</a>.
 */
public class VertexAiChatModel implements ChatLanguageModel {
    private final PredictionServiceSettings settings;
    private final EndpointName endpointName;
    private final VertexAiParameters vertexAiParameters;
    private final Integer maxRetries;
    public VertexAiChatModel(String endpoint,
                             String project,
                             String location,
                             String publisher,
                             String modelName,
                             Double temperature,
                             Integer maxOutputTokens,
                             Integer topK,
                             Double topP,
                             Integer maxRetries) {
        try {
            this.settings = PredictionServiceSettings.newBuilder()
                    .setEndpoint(ensureNotBlank(endpoint, "endpoint"))
                    .build();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        this.endpointName = EndpointName.ofProjectLocationPublisherModelName(
                ensureNotBlank(project, "project"),
                ensureNotBlank(location, "location"),
                ensureNotBlank(publisher, "publisher"),
```

```
                ensureNotBlank(modelName, "modelName")
        );
        this.vertexAiParameters = new VertexAiParameters(temperature,
maxOutputTokens, topK, topP);
        this.maxRetries = maxRetries == null ? 3 : maxRetries;
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages) {
        try (PredictionServiceClient client =
PredictionServiceClient.create(settings)) {
            VertexAiChatInstance vertexAiChatInstance = new
VertexAiChatInstance(
                    toContext(messages),
                    toVertexMessages(messages)
            );
            Value.Builder instanceBuilder = newBuilder();
            JsonFormat.parser().merge(toJson(vertexAiChatInstance),
instanceBuilder);
            List<Value> instances = singletonList(instanceBuilder.build());
            Value.Builder parametersBuilder = newBuilder();
            JsonFormat.parser().merge(toJson(vertexAiParameters),
parametersBuilder);
            Value parameters = parametersBuilder.build();
            PredictResponse response = withRetry(() ->
client.predict(endpointName, instances, parameters), maxRetries);
            return Response.from(
                    AiMessage.from(extractContent(response)),
                    new TokenUsage(
                            extractTokenCount(response, "inputTokenCount"),
                            extractTokenCount(response, "outputTokenCount")
                    )
            );
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private static String extractContent(PredictResponse predictResponse) {
        return predictResponse.getPredictions(0)
                .getStructValue()
                .getFieldsMap()
                .get("candidates")
                .getListValue()
                .getValues(0)
                .getStructValue()
                .getFieldsMap()
                .get("content")
                .getStringValue();
    }
    static int extractTokenCount(PredictResponse predictResponse, String
fieldName) {
        return (int) predictResponse.getMetadata()
                .getStructValue()
                .getFieldsMap()
                .get("tokenMetadata")
                .getStructValue()
                .getFieldsMap()
                .get(fieldName)
                .getStructValue()
                .getFieldsMap()
                .get("totalTokens")
                .getNumberValue();
```

```
    }
    private static List<VertexAiChatInstance.Message>
toVertexMessages(List<ChatMessage> messages) {
        return messages.stream()
                .filter(chatMessage -> chatMessage.type() == USER ||
chatMessage.type() == AI)
                .map(chatMessage -> new
VertexAiChatInstance.Message(chatMessage.type().name(), chatMessage.text()))
                .collect(toList());
    }
    private static String toContext(List<ChatMessage> messages) {
        return messages.stream()
                .filter(chatMessage -> chatMessage.type() == SYSTEM)
                .map(ChatMessage::text)
                .collect(joining("\n"));
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
List<ToolSpecification> toolSpecifications) {
        throw new IllegalArgumentException("Tools are currently not supported
for Vertex AI models");
    }
    @Override
    public Response<AiMessage> generate(List<ChatMessage> messages,
ToolSpecification toolSpecification) {
        throw new IllegalArgumentException("Tools are currently not supported
for Vertex AI models");
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String endpoint;
        private String project;
        private String location;
        private String publisher;
        private String modelName;
        private Double temperature;
        private Integer maxOutputTokens = 200;
        private Integer topK;
        private Double topP;
        private Integer maxRetries;
        public Builder endpoint(String endpoint) {
            this.endpoint = endpoint;
            return this;
        }
        public Builder project(String project) {
            this.project = project;
            return this;
        }
        public Builder location(String location) {
            this.location = location;
            return this;
        }
        public Builder publisher(String publisher) {
            this.publisher = publisher;
            return this;
        }
        public Builder modelName(String modelName) {
            this.modelName = modelName;
            return this;
```

```
        }
        public Builder temperature(Double temperature) {
            this.temperature = temperature;
            return this;
        }
        public Builder maxOutputTokens(Integer maxOutputTokens) {
            this.maxOutputTokens = maxOutputTokens;
            return this;
        }
        public Builder topK(Integer topK) {
            this.topK = topK;
            return this;
        }
        public Builder topP(Double topP) {
            this.topP = topP;
            return this;
        }
        public Builder maxRetries(Integer maxRetries) {
            this.maxRetries = maxRetries;
            return this;
        }
        public VertexAiChatModel build() {
            return new VertexAiChatModel(
                    endpoint,
                    project,
                    location,
                    publisher,
                    modelName,
                    temperature,
                    maxOutputTokens,
                    topK,
                    topP,
                    maxRetries);
        }
    }
}
```

langchain4j-vertex-ai\src\main\java\dev\langchain4j\model\vertexai\VertexAiEmbeddingInstance.java

```
package dev.langchain4j.model.vertexai;
class VertexAiEmbeddingInstance {
    private final String content;
    VertexAiEmbeddingInstance(String content) {
        this.content = content;
    }
}
```

langchain4j-vertex-ai\src\main\java\dev\langchain4j\model\vertexai\VertexAiEmbeddingModel.java

```
package dev.langchain4j.model.vertexai;
import com.google.cloud.aiplatform.v1.EndpointName;
import com.google.cloud.aiplatform.v1.PredictResponse;
import com.google.cloud.aiplatform.v1.PredictionServiceClient;
import com.google.cloud.aiplatform.v1.PredictionServiceSettings;
import com.google.protobuf.Value;
import com.google.protobuf.util.JsonFormat;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import static com.google.cloud.aiplatform.util.ValueConverter.EMPTY_VALUE;
import static dev.langchain4j.internal.Json.toJson;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.Utils.getOrDefault;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static java.util.stream.Collectors.toList;
/**
 * Represents a Google Vertex AI embedding model, such as textembedding-gecko.
 * See details <a href="https://cloud.google.com/vertex-ai/docs/generative-ai/
embeddings/get-text-embeddings">here</a>.
 */
public class VertexAiEmbeddingModel implements EmbeddingModel {
    private static final int BATCH_SIZE = 5; // Vertex AI has a limit of up
to 5 input texts per request
    private final PredictionServiceSettings settings;
    private final EndpointName endpointName;
    private final Integer maxRetries;
    public VertexAiEmbeddingModel(String endpoint,
                                  String project,
                                  String location,
                                  String publisher,
                                  String modelName,
                                  Integer maxRetries) {
        try {
            this.settings = PredictionServiceSettings.newBuilder()
                    .setEndpoint(ensureNotBlank(endpoint, "endpoint"))
                    .build();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        this.endpointName = EndpointName.ofProjectLocationPublisherModelName(
                ensureNotBlank(project, "project"),
                ensureNotBlank(location, "location"),
                ensureNotBlank(publisher, "publisher"),
                ensureNotBlank(modelName, "modelName")
        );
        this.maxRetries = getOrDefault(maxRetries, 3);
    }
    @Override
    public Response<List<Embedding>> embedAll(List<TextSegment> segments) {
        try (PredictionServiceClient client =
PredictionServiceClient.create(settings)) {
```

```
            List<Embedding> embeddings = new ArrayList<>();
            int inputTokenCount = 0;
            for (int i = 0; i < segments.size(); i += BATCH_SIZE) {
                List<TextSegment> batch = segments.subList(i, Math.min(i +
BATCH_SIZE, segments.size()));
                List<Value> instances = new ArrayList<>();
                for (TextSegment segment : batch) {
                    Value.Builder instanceBuilder = Value.newBuilder();
                    JsonFormat.parser().merge(toJson(new
VertexAiEmbeddingInstance(segment.text())), instanceBuilder);
                    instances.add(instanceBuilder.build());
                }
                PredictResponse response = withRetry(() ->
client.predict(endpointName, instances, EMPTY_VALUE), maxRetries);
                embeddings.addAll(response.getPredictionsList().stream()
                        .map(VertexAiEmbeddingModel::toEmbedding)
                        .collect(toList()));
                for (Value prediction : response.getPredictionsList()) {
                    inputTokenCount += extractTokenCount(prediction);
                }
            }
            return Response.from(
                    embeddings,
                    new TokenUsage(inputTokenCount)
            );
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
    private static Embedding toEmbedding(Value prediction) {
        List<Float> vector = prediction.getStructValue()
                .getFieldsMap()
                .get("embeddings")
                .getStructValue()
                .getFieldsOrThrow("values")
                .getListValue()
                .getValuesList()
                .stream()
                .map(v -> (float) v.getNumberValue())
                .collect(toList());
        return Embedding.from(vector);
    }
    private static int extractTokenCount(Value prediction) {
        return (int) prediction.getStructValue()
                .getFieldsMap()
                .get("embeddings")
                .getStructValue()
                .getFieldsMap()
                .get("statistics")
                .getStructValue()
                .getFieldsMap()
                .get("token_count")
                .getNumberValue();
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String endpoint;
        private String project;
        private String location;
```

```
        private String publisher;
        private String modelName;
        private Integer maxRetries;
        public Builder endpoint(String endpoint) {
            this.endpoint = endpoint;
            return this;
        }
        public Builder project(String project) {
            this.project = project;
            return this;
        }
        public Builder location(String location) {
            this.location = location;
            return this;
        }
        public Builder publisher(String publisher) {
            this.publisher = publisher;
            return this;
        }
        public Builder modelName(String modelName) {
            this.modelName = modelName;
            return this;
        }
        public Builder maxRetries(Integer maxRetries) {
            this.maxRetries = maxRetries;
            return this;
        }
        public VertexAiEmbeddingModel build() {
            return new VertexAiEmbeddingModel(
                    endpoint,
                    project,
                    location,
                    publisher,
                    modelName,
                    maxRetries);
        }
    }
}
```

langchain4j-vertex-ai\src\main\java\dev\langchain4j\model\vertexai\VertexAiParameters.java

```
package dev.langchain4j.model.vertexai;
class VertexAiParameters {
    private final Double temperature;
    private final Integer maxOutputTokens;
    private final Integer topK;
    private final Double topP;
    VertexAiParameters(Double temperature,
                       Integer maxOutputTokens,
                       Integer topK,
                       Double topP) {
        this.temperature = temperature;
        this.maxOutputTokens = maxOutputTokens;
        this.topK = topK;
        this.topP = topP;
    }
}
```

langchain4j-vertex-ai\src\main\java\dev\langchain4j\model\vertexai\VertexAiTextInstance.java

```
package dev.langchain4j.model.vertexai;
class VertexAiTextInstance {
    private final String prompt;
    VertexAiTextInstance(String prompt) {
        this.prompt = prompt;
    }
}
```

langchain4j-vertex-ai\src\main\java\dev\langchain4j\model\vertexai\VextexAiLanguageModel.java

```
package dev.langchain4j.model.vertexai;
import com.google.cloud.aiplatform.v1.EndpointName;
import com.google.cloud.aiplatform.v1.PredictResponse;
import com.google.cloud.aiplatform.v1.PredictionServiceClient;
import com.google.cloud.aiplatform.v1.PredictionServiceSettings;
import com.google.protobuf.Value;
import com.google.protobuf.util.JsonFormat;
import dev.langchain4j.model.language.LanguageModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import java.io.IOException;
import java.util.List;
import static com.google.protobuf.Value.newBuilder;
import static dev.langchain4j.internal.Json.toJson;
import static dev.langchain4j.internal.RetryUtils.withRetry;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static
dev.langchain4j.model.vertexai.VertexAiChatModel.extractTokenCount;
import static java.util.Collections.singletonList;
/**
 * Represents a Google Vertex AI language model with a text interface, such
as text-bison.
 * See details <a href="https://cloud.google.com/vertex-ai/docs/generative-ai/
text/text-overview">here</a>.
 */
public class VextexAiLanguageModel implements LanguageModel {
    private final PredictionServiceSettings settings;
    private final EndpointName endpointName;
    private final VertexAiParameters vertexAiParameters;
    private final Integer maxRetries;
    public VextexAiLanguageModel(String endpoint,
                                 String project,
                                 String location,
                                 String publisher,
                                 String modelName,
                                 Double temperature,
                                 Integer maxOutputTokens,
                                 Integer topK,
                                 Double topP,
                                 Integer maxRetries) {
        try {
            this.settings = PredictionServiceSettings.newBuilder()
                    .setEndpoint(ensureNotBlank(endpoint, "endpoint"))
                    .build();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        this.endpointName = EndpointName.ofProjectLocationPublisherModelName(
                ensureNotBlank(project, "project"),
                ensureNotBlank(location, "location"),
                ensureNotBlank(publisher, "publisher"),
                ensureNotBlank(modelName, "modelName")
        );
        this.vertexAiParameters = new VertexAiParameters(temperature,
maxOutputTokens, topK, topP);
        this.maxRetries = maxRetries == null ? 3 : maxRetries;
    }
```

```
    @Override
    public Response<String> generate(String prompt) {
        try (PredictionServiceClient client =
PredictionServiceClient.create(settings)) {
            Value.Builder instanceBuilder = newBuilder();
            JsonFormat.parser().merge(toJson(new
VertexAiTextInstance(prompt)), instanceBuilder);
            List<Value> instances = singletonList(instanceBuilder.build());
            Value.Builder parametersBuilder = Value.newBuilder();
            JsonFormat.parser().merge(toJson(vertexAiParameters),
parametersBuilder);
            Value parameters = parametersBuilder.build();
            PredictResponse response = withRetry(() ->
client.predict(endpointName, instances, parameters), maxRetries);
            return Response.from(
                    extractContent(response),
                    new TokenUsage(
                            extractTokenCount(response, "inputTokenCount"),
                            extractTokenCount(response, "outputTokenCount")
                    )
            );
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private static String extractContent(PredictResponse predictResponse) {
        return predictResponse.getPredictions(0)
                .getStructValue()
                .getFieldsMap()
                .get("content")
                .getStringValue();
    }
    public static Builder builder() {
        return new Builder();
    }
    public static class Builder {
        private String endpoint;
        private String project;
        private String location;
        private String publisher;
        private String modelName;
        private Double temperature;
        private Integer maxOutputTokens = 200;
        private Integer topK;
        private Double topP;
        private Integer maxRetries;
        public Builder endpoint(String endpoint) {
            this.endpoint = endpoint;
            return this;
        }
        public Builder project(String project) {
            this.project = project;
            return this;
        }
        public Builder location(String location) {
            this.location = location;
            return this;
        }
        public Builder publisher(String publisher) {
            this.publisher = publisher;
            return this;
```

```
        }
        public Builder modelName(String modelName) {
            this.modelName = modelName;
            return this;
        }
        public Builder temperature(Double temperature) {
            this.temperature = temperature;
            return this;
        }
        public Builder maxOutputTokens(Integer maxOutputTokens) {
            this.maxOutputTokens = maxOutputTokens;
            return this;
        }
        public Builder topK(Integer topK) {
            this.topK = topK;
            return this;
        }
        public Builder topP(Double topP) {
            this.topP = topP;
            return this;
        }
        public Builder maxRetries(Integer maxRetries) {
            this.maxRetries = maxRetries;
            return this;
        }
        public VextexAiLanguageModel build() {
            return new VextexAiLanguageModel(
                    endpoint,
                    project,
                    location,
                    publisher,
                    modelName,
                    temperature,
                    maxOutputTokens,
                    topK,
                    topP,
                    maxRetries);
        }
    }
}
```

langchain4j-vertex-ai\src\test\java\dev\langchain4j\model\vertexai\VertexAiChatModelIT.java

```
package dev.langchain4j.model.vertexai;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class VertexAiChatModelIT {
    @Test
    @Disabled("To run this test, you must have provide your own endpoint, project and location")
    void testChatModel() {
        VertexAiChatModel vertexAiChatModel = VertexAiChatModel.builder()
                .endpoint("us-central1-aiplatform.googleapis.com:443")
                .project("langchain4j")
                .location("us-central1")
                .publisher("google")
                .modelName("chat-bison@001")
                .temperature(1.0)
                .maxOutputTokens(50)
                .topK(0)
                .topP(0.0)
                .maxRetries(3)
                .build();
        Response<AiMessage> response = vertexAiChatModel.generate(UserMessage.from("hi, how are you doing?"));
        System.out.println(response);
        assertThat(response.content().text()).isNotBlank();
        TokenUsage tokenUsage = response.tokenUsage();
        assertThat(tokenUsage.inputTokenCount()).isEqualTo(7);
        assertThat(tokenUsage.outputTokenCount()).isGreaterThan(1);
        assertThat(tokenUsage.totalTokenCount()).isGreaterThan(8);
        assertThat(response.finishReason()).isNull();
    }
}
```

langchain4j-vertex-ai\src\test\java\dev\langchain4j\model\vertexai\VertexAiEmbeddingModelIT.java

```
package dev.langchain4j.model.vertexai;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import java.util.Arrays;
import java.util.List;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
class VertexAiEmbeddingModelIT {
    @Test
    @Disabled("To run this test, you must have provide your own endpoint,
project and location")
    void testEmbeddingModel() {
        EmbeddingModel embeddingModel = VertexAiEmbeddingModel.builder()
                .endpoint("us-central1-aiplatform.googleapis.com:443")
                .project("langchain4j")
                .location("us-central1")
                .publisher("google")
                .modelName("textembedding-gecko@001")
                .maxRetries(3)
                .build();
        List<TextSegment> segments = asList(
                TextSegment.from("one"),
                TextSegment.from("two"),
                TextSegment.from("three"),
                TextSegment.from("four"),
                TextSegment.from("five"),
                TextSegment.from("six")
        );
        Response<List<Embedding>> response =
embeddingModel.embedAll(segments);
        List<Embedding> embeddings = response.content();
        assertThat(embeddings).hasSize(6);
        Embedding embedding = embeddings.get(0);
        assertThat(embedding.vector()).hasSize(768);
        System.out.println(Arrays.toString(embedding.vector()));
        TokenUsage tokenUsage = response.tokenUsage();
        assertThat(tokenUsage.inputTokenCount()).isEqualTo(6);
        assertThat(tokenUsage.outputTokenCount()).isNull();
        assertThat(tokenUsage.totalTokenCount()).isEqualTo(6);
        assertThat(response.finishReason()).isNull();
    }
}
```

langchain4j-vertex-ai\src\test\java\dev\langchain4j\model\vertexai\VextexAiLanguageModelIT.java

```
package dev.langchain4j.model.vertexai;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.output.TokenUsage;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;
class VextexAiLanguageModelIT {
    @Test
    @Disabled("To run this test, you must have provide your own endpoint, project and location")
    void testLanguageModel() {
        VextexAiLanguageModel vextexAiLanguageModel = VextexAiLanguageModel.builder()
                .endpoint("us-central1-aiplatform.googleapis.com:443")
                .project("langchain4j")
                .location("us-central1")
                .publisher("google")
                .modelName("text-bison@001")
                .temperature(0.2)
                .maxOutputTokens(50)
                .topK(40)
                .topP(0.95)
                .maxRetries(3)
                .build();
        Response<String> response = vextexAiLanguageModel.generate("hi, what is java?");
        assertThat(response.content()).containsIgnoringCase("java");
        System.out.println(response);
        TokenUsage tokenUsage = response.tokenUsage();
        assertThat(tokenUsage.inputTokenCount()).isEqualTo(6);
        assertThat(tokenUsage.outputTokenCount()).isGreaterThan(1);
        assertThat(tokenUsage.totalTokenCount()).isGreaterThan(7);
        assertThat(response.finishReason()).isNull();
    }
}
```

langchain4j-vespa\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-vespa</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Vespa</name>
    <description>
        Vespa is a fully featured search engine and vector database.
        Integration with LangChain4j uses:
        - Vespa's client &amp;amp; vespa-feed-client libraries which have Apache
License 2.0:
        https://github.com/vespa-engine/vespa/blob/master/LICENSE
        - org.apache.httpcomponents.client5.httpclient5-fluent library which
has Apache License 2.0:
        https://github.com/apache/httpcomponents-client/blob/master/
LICENSE.txt
    </description>
    <properties>
        <vespa.version>8.190.2</vespa.version> <!-- the latest Java 8 version
-->
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.yahoo.vespa</groupId>
            <artifactId>client</artifactId>
            <version>${vespa.version}</version>
        </dependency>
        <dependency>
            <groupId>com.yahoo.vespa</groupId>
            <artifactId>vespa-feed-client</artifactId>
            <version>${vespa.version}</version>
        </dependency>
        <dependency>
            <groupId>com.squareup.retrofit2</groupId>
            <artifactId>retrofit</artifactId>
        </dependency>
        <dependency>
            <groupId>com.squareup.retrofit2</groupId>
            <artifactId>converter-gson</artifactId>
        </dependency>
        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <licenses>
        <license>
            <name>Apache-2.0</name>
            <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
            <distribution>repo</distribution>
            <comments>A business-friendly OSS license</comments>
        </license>
    </licenses>
</project>
```

langchain4j-vespa\src\main\java\dev\langchain4j\store\embedding\vespa\QueryResponse.java

```
package dev.langchain4j.store.embedding.vespa;
import java.util.List;
class QueryResponse {
  private RootNode root;
  public RootNode getRoot() {
    return root;
  }
  public void setRoot(RootNode root) {
    this.root = root;
  }
  public static class RootNode {
    private List<Record> children;
    public List<Record> getChildren() {
      return children;
    }
    public void setChildren(List<Record> children) {
      this.children = children;
    }
  }
}
```

langchain4j-vespa\src\main\java\dev\langchain4j\store\embedding\vespa\Record.java

```
package dev.langchain4j.store.embedding.vespa;
import com.google.gson.annotations.SerializedName;
import java.util.List;
class Record {
  private String id;
  private Double relevance;
  private Fields fields;
  public Record(String id, String textSegment, List<Float> vector) {
    this.id = id;
    this.fields = new Fields(textSegment, vector);
  }
  public String getId() {
    return id;
  }
  public void setId(String id) {
    this.id = id;
  }
  public double getRelevance() {
    return relevance;
  }
  public void setRelevance(double relevance) {
    this.relevance = relevance;
  }
  public Fields getFields() {
    return fields;
  }
  public void setFields(Fields fields) {
    this.fields = fields;
  }
  public static class Fields {
    @SerializedName("documentid")
    private String documentId;
    @SerializedName("text_segment")
    private String textSegment;
    private Vector vector;
    public Fields(String textSegment, List<Float> vector) {
      this.textSegment = textSegment;
      this.vector = new Vector(vector);
    }
    public String getDocumentId() {
      return documentId;
    }
    public void setDocumentId(String documentId) {
      this.documentId = documentId;
    }
    public String getTextSegment() {
      return textSegment;
    }
    public void setTextSegment(String textSegment) {
      this.textSegment = textSegment;
    }
    public Vector getVector() {
      return vector;
    }
    public void setVector(Vector vector) {
      this.vector = vector;
    }
```

```
    public static class Vector {
      private List<Float> values;
      public Vector(List<Float> values) {
        this.values = values;
      }
      public List<Float> getValues() {
        return values;
      }
      public void setValues(List<Float> values) {
        this.values = values;
      }
    }
  }
}
```

langchain4j-vespa\src\main\java\dev\langchain4j\store\embedding\vespa\VespaEmbeddingStore.java

```
package dev.langchain4j.store.embedding.vespa;
import static dev.langchain4j.internal.Utils.generateUUIDFrom;
import static dev.langchain4j.internal.Utils.randomUUID;
import static
dev.langchain4j.store.embedding.vespa.VespaQueryClient.createInstance;
import ai.vespa.client.dsl.A;
import ai.vespa.client.dsl.Annotation;
import ai.vespa.client.dsl.NearestNeighbor;
import ai.vespa.client.dsl.Q;
import ai.vespa.feed.client.*;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.internal.Json;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.net.URI;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.atomic.AtomicReference;
import java.util.stream.Collectors;
import lombok.Builder;
import lombok.SneakyThrows;
import retrofit2.Response;
/**
 * Represents the <a href="https://vespa.ai/">Vespa</a> - search engine and
vector database.
 * Does not support storing {@link dev.langchain4j.data.document.Metadata}
yet.
 * Example server configuration contains cosine similarity search rank
profile, of course other Vespa neighbor search
 * methods are supported too. Read more <a href="https://docs.vespa.ai/en/
nearest-neighbor-search.html">here</a>.
 */
public class VespaEmbeddingStore implements EmbeddingStore<TextSegment> {
  private static final Duration DEFAULT_TIMEOUT = Duration.ofSeconds(5);
  private static final String DEFAULT_NAMESPACE = "namespace";
  private static final String DEFAULT_DOCUMENT_TYPE = "langchain4j";
  private static final boolean DEFAULT_AVOID_DUPS = true;
  private static final String FIELD_NAME_TEXT_SEGMENT = "text_segment";
  private static final String FIELD_NAME_VECTOR = "vector";
  private static final String FIELD_NAME_DOCUMENT_ID = "documentid";
  private static final String DEFAULT_RANK_PROFILE = "cosine_similarity";
  private static final int DEFAULT_TARGET_HITS = 10;
  private final String url;
  private final Path keyPath;
  private final Path certPath;
  private final Duration timeout;
  private final String namespace;
  private final String documentType;
  private final String rankProfile;
  private final int targetHits;
```

```
  private final boolean avoidDups;
  private VespaQueryApi queryApi;
  /**
   * Creates a new VespaEmbeddingStore instance.
   *
   * @param url          server url, local or cloud one. The latter you can
find under Endpoint of your Vespa
   *                     application, e.g. https://alexey-
heezer.langchain4j.mytenant346.aws-us-east-1c.dev.z.vespa-app.cloud/
   * @param keyPath      local path to the SSL private key file in PEM
format. Read
   *                     <a href="https://cloud.vespa.ai/en/getting-started-
java">docs</a> for details.
   * @param certPath     local path to the SSL certificate file in PEM
format. Read
   *                     <a href="https://cloud.vespa.ai/en/getting-started-
java">docs</a> for details.
   * @param timeout      for Vespa Java client in <code>java.time.Duration</
code> format.
   * @param namespace    required for document ID generation, find more
details
   *                     <a href="https://docs.vespa.ai/en/
documents.html#namespace">here</a>.
   * @param documentType document type, used for document ID generation, find
more details
   *                     <a href="https://docs.vespa.ai/en/
documents.html#namespace">here</a> and data querying
   * @param rankProfile  rank profile from your .sd schema. Provided example
schema configures cosine similarity match
   * @param targetHits   sets the number of hits (10 is default) exposed to
the real Vespa's first-phase ranking
   *                     function per content node, find more details
   *                     <a href="https://docs.vespa.ai/en/nearest-neighbor-
search.html#querying-using-nearestneighbor-query-operator">here</a>.
   * @param avoidDups    if true (default), then <code>VespaEmbeddingStore</
code> will generate a hashed ID based on
   *                     provided text segment, which avoids duplicated
entries in DB.
   *                     If false, then random ID will be generated.
   */
  @Builder
  public VespaEmbeddingStore(
    String url,
    String keyPath,
    String certPath,
    Duration timeout,
    String namespace,
    String documentType,
    String rankProfile,
    Integer targetHits,
    Boolean avoidDups
  ) {
    this.url = url;
    this.keyPath = Paths.get(keyPath);
    this.certPath = Paths.get(certPath);
    this.timeout = timeout != null ? timeout : DEFAULT_TIMEOUT;
    this.namespace = namespace != null ? namespace : DEFAULT_NAMESPACE;
    this.documentType = documentType != null ? documentType :
DEFAULT_DOCUMENT_TYPE;
    this.rankProfile = rankProfile != null ? rankProfile :
DEFAULT_RANK_PROFILE;
```

```
    this.targetHits = targetHits != null ? targetHits : DEFAULT_TARGET_HITS;
    this.avoidDups = avoidDups != null ? avoidDups : DEFAULT_AVOID_DUPS;
  }
  @Override
  public String add(Embedding embedding) {
    return add(null, embedding, null);
  }
  /**
   * Adds a new embedding with provided ID to the store.
   *
   * @param id        "user-specified" part of document ID, find more details
   *                  <a href="https://docs.vespa.ai/en/
documents.html#namespace">here</a>
   * @param embedding the embedding to add
   */
  @Override
  public void add(String id, Embedding embedding) {
    add(id, embedding, null);
  }
  @Override
  public String add(Embedding embedding, TextSegment textSegment) {
    return add(null, embedding, textSegment);
  }
  @Override
  public List<String> addAll(List<Embedding> embeddings) {
    return addAll(embeddings, null);
  }
  @Override
  public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
embedded) {
    if (embedded != null && embeddings.size() != embedded.size()) {
      throw new IllegalArgumentException("The list of embeddings and embedded
must have the same size");
    }
    List<String> ids = new ArrayList<>();
    try (JsonFeeder jsonFeeder = buildJsonFeeder()) {
      List<Record> records = new ArrayList<>();
      for (int i = 0; i < embeddings.size(); i++) {
        records.add(buildRecord(embeddings.get(i), embedded != null ?
embedded.get(i) : null));
      }
      jsonFeeder.feedMany(
        Json.toInputStream(records, List.class),
        new JsonFeeder.ResultCallback() {
          @Override
          public void onNextResult(Result result, FeedException error) {
            if (error != null) {
              throw new RuntimeException(error.getMessage());
            } else if (Result.Type.success.equals(result.type())) {
              ids.add(result.documentId().toString());
            }
          }
          @Override
          public void onError(FeedException error) {
            throw new RuntimeException(error.getMessage());
          }
        }
      );
    } catch (IOException e) {
      throw new RuntimeException(e);
    }
```

```
    return ids;
  }
  /**
   * {@inheritDoc}
   * The score inside {@link EmbeddingMatch} is Vespa relevance according to
provided rank profile.
   */
  @Override
  @SneakyThrows
  public List<EmbeddingMatch<TextSegment>> findRelevant(Embedding
referenceEmbedding, int maxResults, double minScore) {
    try {
      String searchQuery = Q
        .select(FIELD_NAME_DOCUMENT_ID, FIELD_NAME_TEXT_SEGMENT,
FIELD_NAME_VECTOR)
        .from(documentType)
        .where(buildNearestNeighbor())
        .fix()
        .hits(maxResults)
        .ranking(rankProfile)
        .param("input.query(q)",
Json.toJson(referenceEmbedding.vectorAsList()))
        .param("input.query(threshold)", String.valueOf(minScore))
        .build();
      Response<QueryResponse> response =
getQueryApi().search(searchQuery).execute();
      if (response.isSuccessful()) {
        QueryResponse parsedResponse = response.body();
        return parsedResponse
          .getRoot()
          .getChildren()
          .stream()
          .map(VespaEmbeddingStore::toEmbeddingMatch)
          .collect(Collectors.toList());
      } else {
        throw new RuntimeException("Request failed");
      }
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }
  private String add(String id, Embedding embedding, TextSegment textSegment)
{
    AtomicReference<String> resId = new AtomicReference<>();
    try (JsonFeeder jsonFeeder = buildJsonFeeder()) {
      jsonFeeder
        .feedSingle(Json.toJson(buildRecord(id, embedding, textSegment)))
        .whenComplete(
          (
            (result, throwable) -> {
              if (throwable != null) {
                throw new RuntimeException(throwable);
              } else if (Result.Type.success.equals(result.type())) {
                resId.set(result.documentId().toString());
              }
            }
          )
        );
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
```

```
    return resId.get();
  }
  private JsonFeeder buildJsonFeeder() {
    return JsonFeeder
      .builder(FeedClientBuilder.create(URI.create(url)).setCertificate(certPa
th, keyPath).build())
      .withTimeout(timeout)
      .build();
  }
  private VespaQueryApi getQueryApi() {
    if (queryApi == null) {
      queryApi = createInstance(url, certPath, keyPath);
    }
    return queryApi;
  }
  private static EmbeddingMatch<TextSegment> toEmbeddingMatch(Record in) {
    return new EmbeddingMatch<>(
      in.getRelevance(),
      in.getFields().getDocumentId(),
      Embedding.from(in.getFields().getVector().getValues()),
      TextSegment.from(in.getFields().getTextSegment())
    );
  }
  private Record buildRecord(String id, Embedding embedding, TextSegment
textSegment) {
    String recordId = id != null
      ? id
      : avoidDups && textSegment != null ?
generateUUIDFrom(textSegment.text()) : randomUUID();
    DocumentId documentId = DocumentId.of(namespace, documentType, recordId);
    String text = textSegment != null ? textSegment.text() : null;
    return new Record(documentId.toString(), text, embedding.vectorAsList());
  }
  private Record buildRecord(Embedding embedding, TextSegment textSegment) {
    return buildRecord(null, embedding, textSegment);
  }
  private NearestNeighbor buildNearestNeighbor()
    throws NoSuchMethodException, IllegalAccessException,
InvocationTargetException {
    NearestNeighbor nb = Q.nearestNeighbor(FIELD_NAME_VECTOR, "q");
    // workaround to invoke ai.vespa.client.dsl.NearestNeighbor#annotate,
    // see https://github.com/vespa-engine/vespa/issues/28029
    // The bug is fixed in the meantime, but the baseline has been upgraded
to Java 11, hence this workaround remains here
    Method method = NearestNeighbor.class.getDeclaredMethod("annotate", new
Class<?>[] { Annotation.class });
    method.setAccessible(true);
    method.invoke(nb, A.a("targetHits", targetHits));
    return nb;
  }
}
```

langchain4j-vespa\src\main\java\dev\langchain4j\store\embedding\vespa\VespaQueryApi.java

```
package dev.langchain4j.store.embedding.vespa;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;
interface VespaQueryApi {
  @GET("search/{query}")
  Call<QueryResponse> search(@Path(value = "query", encoded = true) String query);
}
```

langchain4j-vespa\src\main\java\dev\langchain4j\store\embedding\vespa\VespaQueryClient.java

```
// Copyright Yahoo. Licensed under the terms of the Apache 2.0 license. See LICENSE in the project root.
package dev.langchain4j.store.embedding.vespa;
import com.google.gson.GsonBuilder;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.security.GeneralSecurityException;
import java.security.KeyFactory;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.Certificate;
import java.security.cert.X509Certificate;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.ArrayList;
import java.util.List;
import javax.net.ssl.*;
import okhttp3.HttpUrl;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import org.bouncycastle.asn1.ASN1ObjectIdentifier;
import org.bouncycastle.asn1.pkcs.PKCSObjectIdentifiers;
import org.bouncycastle.asn1.pkcs.PrivateKeyInfo;
import org.bouncycastle.asn1.x9.X9ObjectIdentifiers;
import org.bouncycastle.cert.X509CertificateHolder;
import org.bouncycastle.cert.jcajce.JcaX509CertificateConverter;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.openssl.PEMKeyPair;
import org.bouncycastle.openssl.PEMParser;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
/**
 * This Workaround is needed because of <a href="https://github.com/vespa-engine/vespa/issues/28026">this request</a>.
 * It will be redundant as soon as vespa-client is implemented. This class is copied from <code>vespa-feed-client</code>.
 * BouncyCastle integration for creating a {@link SSLContext} instance from PEM encoded material
 */
class VespaQueryClient {
  static final BouncyCastleProvider bcProvider = new BouncyCastleProvider();
  public static VespaQueryApi createInstance(String baseUrl, Path certificate, Path privateKey) {
    try {
      KeyStore keystore = KeyStore.getInstance("PKCS12");
      keystore.load(null);
      keystore.setKeyEntry("cert", privateKey(privateKey), new char[0], certificates(certificate));
      // Protocol version must be equal to TlsContext.SSL_CONTEXT_VERSION or higher
      SSLContext sslContext = SSLContext.getInstance("TLSv1.3");
      sslContext.init(createKeyManagers(keystore), null, /*Default secure random algorithm*/null);
      TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(
        TrustManagerFactory.getDefaultAlgorithm()
      );
```

```
      trustManagerFactory.init(keystore);
      OkHttpClient client = new OkHttpClient.Builder()
        .sslSocketFactory(sslContext.getSocketFactory(), (X509TrustManager)
trustManagerFactory.getTrustManagers()[0])
        .addInterceptor(chain -> {
          // trick to format the query URL exactly how Vespa expects it
(search/?query),
          // see https://docs.vespa.ai/en/reference/query-language-
reference.html
          Request request = chain.request();
          HttpUrl url = request
            .url()
            .newBuilder()
            .removePathSegment(1)
            .addPathSegment("")
            .encodedQuery(request.url().encodedPathSegments().get(1))
            .build();
          request = request.newBuilder().url(url).build();
          return chain.proceed(request);
        })
        .build();
      Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(baseUrl)
        .client(client)
        .addConverterFactory(GsonConverterFactory.create(new
GsonBuilder().create()))
        .build();
      return retrofit.create(VespaQueryApi.class);
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }
  private static KeyManager[] createKeyManagers(KeyStore keystore) throws
GeneralSecurityException {
    KeyManagerFactory kmf =
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
    kmf.init(keystore, new char[0]);
    return kmf.getKeyManagers();
  }
  private static Certificate[] certificates(Path file) throws IOException,
GeneralSecurityException {
    try (PEMParser parser = new PEMParser(Files.newBufferedReader(file))) {
      List<X509Certificate> result = new ArrayList<>();
      Object pemObject;
      while ((pemObject = parser.readObject()) != null) {
        result.add(toX509Certificate(pemObject));
      }
      if (result.isEmpty()) throw new IOException("File contains no PEM
encoded certificates: " + file);
      return result.toArray(new Certificate[0]);
    }
  }
  private static PrivateKey privateKey(Path file) throws IOException,
GeneralSecurityException {
    try (PEMParser parser = new PEMParser(Files.newBufferedReader(file))) {
      Object pemObject;
      while ((pemObject = parser.readObject()) != null) {
        if (pemObject instanceof PrivateKeyInfo) {
          PrivateKeyInfo keyInfo = (PrivateKeyInfo) pemObject;
          PKCS8EncodedKeySpec keySpec = new
PKCS8EncodedKeySpec(keyInfo.getEncoded());
```

```
          return createKeyFactory(keyInfo).generatePrivate(keySpec);
        } else if (pemObject instanceof PEMKeyPair) {
          PEMKeyPair pemKeypair = (PEMKeyPair) pemObject;
          PrivateKeyInfo keyInfo = pemKeypair.getPrivateKeyInfo();
          return createKeyFactory(keyInfo).generatePrivate(new
PKCS8EncodedKeySpec(keyInfo.getEncoded()));
        }
      }
      throw new IOException("Could not find private key in PEM file");
    }
  }
  private static X509Certificate toX509Certificate(Object pemObject) throws
IOException, GeneralSecurityException {
    if (pemObject instanceof X509Certificate) return (X509Certificate)
pemObject;
    if (pemObject instanceof X509CertificateHolder) {
      return new JcaX509CertificateConverter()
        .setProvider(bcProvider)
        .getCertificate((X509CertificateHolder) pemObject);
    }
    throw new IOException("Invalid type of PEM object: " + pemObject);
  }
  private static KeyFactory createKeyFactory(PrivateKeyInfo info) throws
IOException, GeneralSecurityException {
    ASN1ObjectIdentifier algorithm =
info.getPrivateKeyAlgorithm().getAlgorithm();
    if (X9ObjectIdentifiers.id_ecPublicKey.equals(algorithm)) {
      return KeyFactory.getInstance("EC", bcProvider);
    } else if (PKCSObjectIdentifiers.rsaEncryption.equals(algorithm)) {
      return KeyFactory.getInstance("RSA", bcProvider);
    } else {
      throw new IOException("Unknown key algorithm: " + algorithm);
    }
  }
}
```

langchain4j-weaviate\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-parent</artifactId>
        <version>0.23.0</version>
        <relativePath>../langchain4j-parent/pom.xml</relativePath>
    </parent>
    <artifactId>langchain4j-weaviate</artifactId>
    <packaging>jar</packaging>
    <name>LangChain4j integration with Weaviate</name>
    <description>Uses io.weaviate.client library which has a BSD 3-Clause
license:
        https://github.com/weaviate/java-client/blob/main/LICENSE
    </description>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>io.weaviate</groupId>
            <artifactId>client</artifactId>
            <version>4.3.0</version>
            <exclusions>
                <!-- due to Cxeb68d52e-5509 vulnerability -->
                <exclusion>
                    <groupId>commons-codec</groupId>
                    <artifactId>commons-codec</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
            <version>1.16.0</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2-q</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.honton.chas</groupId>
                <artifactId>license-maven-plugin</artifactId>
                <configuration>
                    <!-- The weaviate client has a BSD-3 license, see https://
github.com/weaviate/java-client/blob/main/LICENSE -->
                    <skipCompliance>true</skipCompliance>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

langchain4j-weaviate\src\main\java\dev\langchain4j\store\embedding\weaviate\WeaviateEmbeddingStore.java

```
package dev.langchain4j.store.embedding.weaviate;
import static dev.langchain4j.internal.Utils.*;
import static dev.langchain4j.internal.ValidationUtils.ensureNotBlank;
import static
io.weaviate.client.v1.data.replication.model.ConsistencyLevel.QUORUM;
import static java.util.Arrays.stream;
import static java.util.Collections.emptyList;
import static java.util.Collections.singletonList;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import io.weaviate.client.Config;
import io.weaviate.client.WeaviateAuthClient;
import io.weaviate.client.WeaviateClient;
import io.weaviate.client.base.Result;
import io.weaviate.client.base.WeaviateErrorMessage;
import io.weaviate.client.v1.auth.exception.AuthException;
import io.weaviate.client.v1.data.model.WeaviateObject;
import io.weaviate.client.v1.graphql.model.GraphQLError;
import io.weaviate.client.v1.graphql.model.GraphQLResponse;
import io.weaviate.client.v1.graphql.query.argument.NearVectorArgument;
import io.weaviate.client.v1.graphql.query.fields.Field;
import java.util.*;
import lombok.Builder;
/**
 * Represents the <a href="https://weaviate.io/">Weaviate</a> vector database.
 * Current implementation assumes the cosine distance metric is used.
 * Does not support storing {@link dev.langchain4j.data.document.Metadata}
yet.
 */
public class WeaviateEmbeddingStore implements EmbeddingStore<TextSegment> {
  private static final String METADATA_TEXT_SEGMENT = "text";
  private static final String ADDITIONALS = "_additional";
  private final WeaviateClient client;
  private final String objectClass;
  private final boolean avoidDups;
  private final String consistencyLevel;
  /**
   * Creates a new WeaviateEmbeddingStore instance.
   *
   * @param apiKey           Your Weaviate API key
   * @param scheme           The scheme, e.g. "https" of cluster URL. Find in
under Details of your Weaviate cluster.
   * @param host             The host, e.g.
"langchain4j-4jw7ufd9.weaviate.network" of cluster URL.
   *                         Find in under Details of your Weaviate cluster.
   * @param objectClass      The object class you want to store, e.g.
"MyGreatClass". Must start from an uppercase letter.
   * @param avoidDups        If true (default), then
<code>WeaviateEmbeddingStore</code> will generate a hashed ID based on
   *                         provided text segment, which avoids duplicated
entries in DB.
   *                         If false, then random ID will be generated.
   * @param consistencyLevel Consistency level: ONE, QUORUM (default) or ALL.
```

```
Find more details <a href="https://weaviate.io/developers/weaviate/concepts/
replication-architecture/consistency#tunable-write-consistency">here</a>.
   */
  @Builder
  public WeaviateEmbeddingStore(
    String apiKey,
    String scheme,
    String host,
    String objectClass,
    Boolean avoidDups,
    String consistencyLevel
  ) {
    try {
      Config config = new Config(ensureNotBlank(scheme, "scheme"),
ensureNotBlank(host, "host"));
      this.client = WeaviateAuthClient.apiKey(config, ensureNotBlank(apiKey,
"apiKey"));
    } catch (AuthException e) {
      throw new IllegalArgumentException(e);
    }
    this.objectClass = getOrDefault(objectClass, "Default");
    this.avoidDups = getOrDefault(avoidDups, true);
    this.consistencyLevel = getOrDefault(consistencyLevel, QUORUM);
  }
  @Override
  public String add(Embedding embedding) {
    String id = randomUUID();
    add(id, embedding);
    return id;
  }
  /**
   * Adds a new embedding with provided ID to the store.
   *
   * @param id        the ID of the embedding to add in UUID format, since
it's Weaviate requirement.
   *                  See <a href="https://weaviate.io/developers/weaviate/
manage-data/create#id">Weaviate docs</a> and
   *                  <a href="https://en.wikipedia.org/wiki/
Universally_unique_identifier">UUID on Wikipedia</a>
   * @param embedding the embedding to add
   */
  @Override
  public void add(String id, Embedding embedding) {
    addAll(singletonList(id), singletonList(embedding), null);
  }
  @Override
  public String add(Embedding embedding, TextSegment textSegment) {
    return addAll(singletonList(embedding),
singletonList(textSegment)).stream().findFirst().orElse(null);
  }
  @Override
  public List<String> addAll(List<Embedding> embeddings) {
    return addAll(embeddings, null);
  }
  @Override
  public List<String> addAll(List<Embedding> embeddings, List<TextSegment>
embedded) {
    return addAll(null, embeddings, embedded);
  }
  /**
   * {@inheritDoc}
```

```
   * The score inside {@link EmbeddingMatch} is Weaviate's certainty.
   */
  @Override
  public List<EmbeddingMatch<TextSegment>> findRelevant(
    Embedding referenceEmbedding,
    int maxResults,
    double minCertainty
  ) {
    Result<GraphQLResponse> result = client
      .graphQL()
      .get()
      .withClassName(objectClass)
      .withFields(
        Field.builder().name(METADATA_TEXT_SEGMENT).build(),
        Field
          .builder()
          .name(ADDITIONALS)
          .fields(
            Field.builder().name("id").build(),
            Field.builder().name("certainty").build(),
            Field.builder().name("vector").build()
          )
          .build()
      )
      .withNearVector(
        NearVectorArgument
          .builder()
          .vector(referenceEmbedding.vectorAsList().toArray(new Float[0]))
          .certainty((float) minCertainty)
          .build()
      )
      .withLimit(maxResults)
      .run();
    if (result.hasErrors()) {
      throw new IllegalArgumentException(
        result.getError().getMessages().stream().map(WeaviateErrorMessage::get
Message).collect(joining("\n"))
      );
    }
    GraphQLError[] errors = result.getResult().getErrors();
    if (errors != null && errors.length > 0) {
      throw new IllegalArgumentException(stream(errors).map(GraphQLError::getM
essage).collect(joining("\n")));
    }
    Optional<Map.Entry<String, Map>> resGetPart =
      ((Map<String, Map>)
result.getResult().getData()).entrySet().stream().findFirst();
    if (!resGetPart.isPresent()) {
      return emptyList();
    }
    Optional resItemsPart =
resGetPart.get().getValue().entrySet().stream().findFirst();
    if (!resItemsPart.isPresent()) {
      return emptyList();
    }
    List<Map<String, ?>> resItems = ((Map.Entry<String, List<Map<String, ?
>>>) resItemsPart.get()).getValue();
    return resItems.stream().map(WeaviateEmbeddingStore::toEmbeddingMatch).col
lect(toList());
  }
  private List<String> addAll(List<String> ids, List<Embedding> embeddings,
```

```
List<TextSegment> embedded) {
    if (embedded != null && embeddings.size() != embedded.size()) {
      throw new IllegalArgumentException("The list of embeddings and embedded
must have the same size");
    }
    List<String> resIds = new ArrayList<>();
    List<WeaviateObject> objects = new ArrayList<>();
    for (int i = 0; i < embeddings.size(); i++) {
      String id = ids != null
        ? ids.get(i)
        : avoidDups && embedded != null ?
generateUUIDFrom(embedded.get(i).text()) : randomUUID();
      resIds.add(id);
      objects.add(buildObject(id, embeddings.get(i), embedded != null ?
embedded.get(i).text() : ""));
    }
    client
      .batch()
      .objectsBatcher()
      .withObjects(objects.toArray(new WeaviateObject[0]))
      .withConsistencyLevel(consistencyLevel)
      .run();
    return resIds;
  }
  private WeaviateObject buildObject(String id, Embedding embedding, String
text) {
    Map<String, Object> props = new HashMap<>();
    props.put(METADATA_TEXT_SEGMENT, text);
    return WeaviateObject
      .builder()
      .className(objectClass)
      .id(id)
      .vector(embedding.vectorAsList().toArray(new Float[0]))
      .properties(props)
      .build();
  }
  private static EmbeddingMatch<TextSegment> toEmbeddingMatch(Map<String, ?>
item) {
    Map<String, ?> additional = (Map<String, ?>) item.get(ADDITIONALS);
    String text = (String) item.get(METADATA_TEXT_SEGMENT);
    return new EmbeddingMatch<>(
      (Double) additional.get("certainty"),
      (String) additional.get("id"),
      Embedding.from(
        ((List<Double>)
additional.get("vector")).stream().map(Double::floatValue).collect(toList())
      ),
      isNullOrBlank(text) ? null : TextSegment.from(text)
    );
  }
}
```

langchain4j-weaviate\src\test\java\dev\langchain4j\store\embedding\weaviate\WeaviateEmbeddingStoreTest.java

```
package dev.langchain4j.store.embedding.weaviate;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2QuantizedEmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.CosineSimilarity;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.RelevanceScore;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.EnabledIfEnvironmentVariable;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
import static java.util.Arrays.asList;
import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.data.Percentage.withPercentage;
@EnabledIfEnvironmentVariable(named = "WEAVIATE_API_KEY", matches = ".+")
class WeaviateEmbeddingStoreTest {
    EmbeddingStore<TextSegment> embeddingStore =
WeaviateEmbeddingStore.builder()
            .apiKey(System.getenv("WEAVIATE_API_KEY"))
            .scheme("https")
            .host("test3-bwsieg9y.weaviate.network")
            .objectClass("Test" + randomUUID().replace("-", ""))
            .build();
    EmbeddingModel embeddingModel = new
AllMiniLmL6V2QuantizedEmbeddingModel();
    @Test
    void should_add_embedding() {
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_id() {
        String id = randomUUID();
        Embedding embedding = embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(id, embedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isNull();
    }
    @Test
    void should_add_embedding_with_segment() {
```

```
        TextSegment segment = TextSegment.from(randomUUID());
        Embedding embedding = embeddingModel.embed(segment.text()).content();
        String id = embeddingStore.add(embedding, segment);
        assertThat(id).isNotNull();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(embedding, 10);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(1, withPercentage(1));
        assertThat(match.embeddingId()).isEqualTo(id);
        assertThat(match.embedding()).isEqualTo(embedding);
        assertThat(match.embedded()).isEqualTo(segment);
    }
    @Test
    void should_add_multiple_embeddings() {
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        List<String> ids = embeddingStore.addAll(asList(firstEmbedding,
secondEmbedding));
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isNull();
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
        assertThat(secondMatch.embedded()).isNull();
    }
    @Test
    void should_add_multiple_embeddings_with_segments() {
        TextSegment firstSegment = TextSegment.from(randomUUID());
        Embedding firstEmbedding =
embeddingModel.embed(firstSegment.text()).content();
        TextSegment secondSegment = TextSegment.from(randomUUID());
        Embedding secondEmbedding =
embeddingModel.embed(secondSegment.text()).content();
        List<String> ids = embeddingStore.addAll(
                asList(firstEmbedding, secondEmbedding),
                asList(firstSegment, secondSegment)
        );
        assertThat(ids).hasSize(2);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(ids.get(0));
        assertThat(firstMatch.embedding()).isEqualTo(firstEmbedding);
        assertThat(firstMatch.embedded()).isEqualTo(firstSegment);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(ids.get(1));
        assertThat(secondMatch.embedding()).isEqualTo(secondEmbedding);
```

```
        assertThat(secondMatch.embedded()).isEqualTo(secondSegment);
    }
    @Test
    void should_find_with_min_score() {
        String firstId = randomUUID();
        Embedding firstEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(firstId, firstEmbedding);
        String secondId = randomUUID();
        Embedding secondEmbedding =
embeddingModel.embed(randomUUID()).content();
        embeddingStore.add(secondId, secondEmbedding);
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(firstEmbedding, 10);
        assertThat(relevant).hasSize(2);
        EmbeddingMatch<TextSegment> firstMatch = relevant.get(0);
        assertThat(firstMatch.score()).isCloseTo(1, withPercentage(1));
        assertThat(firstMatch.embeddingId()).isEqualTo(firstId);
        EmbeddingMatch<TextSegment> secondMatch = relevant.get(1);
        assertThat(secondMatch.score()).isBetween(0d, 1d);
        assertThat(secondMatch.embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant2 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() - 0.01
        );
        assertThat(relevant2).hasSize(2);
        assertThat(relevant2.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant2.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant3 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score()
        );
        assertThat(relevant3).hasSize(2);
        assertThat(relevant3.get(0).embeddingId()).isEqualTo(firstId);
        assertThat(relevant3.get(1).embeddingId()).isEqualTo(secondId);
        List<EmbeddingMatch<TextSegment>> relevant4 =
embeddingStore.findRelevant(
                firstEmbedding,
                10,
                secondMatch.score() + 0.01
        );
        assertThat(relevant4).hasSize(1);
        assertThat(relevant4.get(0).embeddingId()).isEqualTo(firstId);
    }
    @Test
    void should_return_correct_score() {
        Embedding embedding = embeddingModel.embed("hello").content();
        String id = embeddingStore.add(embedding);
        assertThat(id).isNotNull();
        Embedding referenceEmbedding = embeddingModel.embed("hi").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(referenceEmbedding, 1);
        assertThat(relevant).hasSize(1);
        EmbeddingMatch<TextSegment> match = relevant.get(0);
        assertThat(match.score()).isCloseTo(

RelevanceScore.fromCosineSimilarity(CosineSimilarity.between(embedding,
```

```
referenceEmbedding)),
                withPercentage(1)
        );
    }
}
```

LICENSE

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.
5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!) The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

mvnw

```
#!/bin/sh
# ----------------------------------------------------------------------------
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
# ----------------------------------------------------------------------------
# ----------------------------------------------------------------------------
# Apache Maven Wrapper startup batch script, version 3.2.0
#
# Required ENV vars:
# ------------------
#   JAVA_HOME - location of a JDK home dir
#
# Optional ENV vars
# -----------------
#   MAVEN_OPTS - parameters passed to the Java VM when running Maven
#     e.g. to debug Maven itself, use
#       set MAVEN_OPTS=-Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000
#   MAVEN_SKIP_RC - flag to disable loading of mavenrc files
# ----------------------------------------------------------------------------
if [ -z "$MAVEN_SKIP_RC" ] ; then
  if [ -f /usr/local/etc/mavenrc ] ; then
    . /usr/local/etc/mavenrc
  fi
  if [ -f /etc/mavenrc ] ; then
    . /etc/mavenrc
  fi
  if [ -f "$HOME/.mavenrc" ] ; then
    . "$HOME/.mavenrc"
  fi
fi
# OS specific support.  $var _must_ be set to either true or false.
cygwin=false;
darwin=false;
mingw=false
case "$(uname)" in
  CYGWIN*) cygwin=true ;;
  MINGW*) mingw=true;;
  Darwin*) darwin=true
    # Use /usr/libexec/java_home if available, otherwise fall back to /
Library/Java/Home
    # See https://developer.apple.com/library/mac/qa/qa1170/_index.html
    if [ -z "$JAVA_HOME" ]; then
      if [ -x "/usr/libexec/java_home" ]; then
```

```
        JAVA_HOME="$(/usr/libexec/java_home)"; export JAVA_HOME
      else
        JAVA_HOME="/Library/Java/Home"; export JAVA_HOME
      fi
    fi
    ;;
esac
if [ -z "$JAVA_HOME" ] ; then
  if [ -r /etc/gentoo-release ] ; then
    JAVA_HOME=$(java-config --jre-home)
  fi
fi
# For Cygwin, ensure paths are in UNIX format before anything is touched
if $cygwin ; then
  [ -n "$JAVA_HOME" ] &&
    JAVA_HOME=$(cygpath --unix "$JAVA_HOME")
  [ -n "$CLASSPATH" ] &&
    CLASSPATH=$(cygpath --path --unix "$CLASSPATH")
fi
# For Mingw, ensure paths are in UNIX format before anything is touched
if $mingw ; then
  [ -n "$JAVA_HOME" ] && [ -d "$JAVA_HOME" ] &&
    JAVA_HOME="$(cd "$JAVA_HOME" || (echo "cannot cd into $JAVA_HOME."; exit
1); pwd)"
fi
if [ -z "$JAVA_HOME" ]; then
  javaExecutable="$(which javac)"
  if [ -n "$javaExecutable" ] && ! [ "$(expr "\"$javaExecutable\"" :
'\([^ ]*\)')" = "no" ]; then
    # readlink(1) is not available as standard on Solaris 10.
    readLink=$(which readlink)
    if [ ! "$(expr "$readLink" : '\([^ ]*\)')" = "no" ]; then
      if $darwin ; then
        javaHome="$(dirname "\"$javaExecutable\"")"
        javaExecutable="$(cd "\"$javaHome\"" && pwd -P)/javac"
      else
        javaExecutable="$(readlink -f "\"$javaExecutable\"")"
      fi
      javaHome="$(dirname "\"$javaExecutable\"")"
      javaHome=$(expr "$javaHome" : '\(.*\)/bin')
      JAVA_HOME="$javaHome"
      export JAVA_HOME
    fi
  fi
fi
if [ -z "$JAVACMD" ] ; then
  if [ -n "$JAVA_HOME"  ] ; then
    if [ -x "$JAVA_HOME/jre/sh/java" ] ; then
      # IBM's JDK on AIX uses strange locations for the executables
      JAVACMD="$JAVA_HOME/jre/sh/java"
    else
      JAVACMD="$JAVA_HOME/bin/java"
    fi
  else
    JAVACMD="$(\unset -f command 2>/dev/null; \command -v java)"
  fi
fi
if [ ! -x "$JAVACMD" ] ; then
  echo "Error: JAVA_HOME is not defined correctly." >&2
  echo "  We cannot execute $JAVACMD" >&2
  exit 1
```

```
fi
if [ -z "$JAVA_HOME" ] ; then
  echo "Warning: JAVA_HOME environment variable is not set."
fi
# traverses directory structure from process work directory to filesystem root
# first directory with .mvn subdirectory is considered project base directory
find_maven_basedir() {
  if [ -z "$1" ]
  then
    echo "Path not specified to find_maven_basedir"
    return 1
  fi
  basedir="$1"
  wdir="$1"
  while [ "$wdir" != '/' ] ; do
    if [ -d "$wdir"/.mvn ] ; then
      basedir=$wdir
      break
    fi
    # workaround for JBEAP-8937 (on Solaris 10/Sparc)
    if [ -d "${wdir}" ]; then
      wdir=$(cd "$wdir/.." || exit 1; pwd)
    fi
    # end of workaround
  done
  printf '%s' "$(cd "$basedir" || exit 1; pwd)"
}
# concatenates all lines of a file
concat_lines() {
  if [ -f "$1" ]; then
    # Remove \r in case we run on Windows within Git Bash
    # and check out the repository with auto CRLF management
    # enabled. Otherwise, we may read lines that are delimited with
    # \r\n and produce $'-Xarg\r' rather than -Xarg due to word
    # splitting rules.
    tr -s '\r\n' ' ' < "$1"
  fi
}
log() {
  if [ "$MVNW_VERBOSE" = true ]; then
    printf '%s\n' "$1"
  fi
}
BASE_DIR=$(find_maven_basedir "$(dirname "$0")")
if [ -z "$BASE_DIR" ]; then
  exit 1;
fi
MAVEN_PROJECTBASEDIR=${MAVEN_BASEDIR:-"$BASE_DIR"}; export
MAVEN_PROJECTBASEDIR
log "$MAVEN_PROJECTBASEDIR"
##########################################################################################
############
# Extension to allow automatically downloading the maven-wrapper.jar from
Maven-central
# This allows using the maven wrapper in projects that prohibit checking in
binary data.
##########################################################################################
############
wrapperJarPath="$MAVEN_PROJECTBASEDIR/.mvn/wrapper/maven-wrapper.jar"
if [ -r "$wrapperJarPath" ]; then
    log "Found $wrapperJarPath"
```

```
else
    log "Couldn't find $wrapperJarPath, downloading it ..."
    if [ -n "$MVNW_REPOURL" ]; then
      wrapperUrl="$MVNW_REPOURL/org/apache/maven/wrapper/maven-wrapper/3.2.0/
maven-wrapper-3.2.0.jar"
    else
      wrapperUrl="https://repo.maven.apache.org/maven2/org/apache/maven/
wrapper/maven-wrapper/3.2.0/maven-wrapper-3.2.0.jar"
    fi
    while IFS="=" read -r key value; do
      # Remove '\r' from value to allow usage on windows as IFS does not
consider '\r' as a separator ( considers space, tab, new line ('\n'), and
custom '=' )
      safeValue=$(echo "$value" | tr -d '\r')
      case "$key" in (wrapperUrl) wrapperUrl="$safeValue"; break ;;
      esac
    done < "$MAVEN_PROJECTBASEDIR/.mvn/wrapper/maven-wrapper.properties"
    log "Downloading from: $wrapperUrl"
    if $cygwin; then
      wrapperJarPath=$(cygpath --path --windows "$wrapperJarPath")
    fi
    if command -v wget > /dev/null; then
        log "Found wget ... using wget"
        [ "$MVNW_VERBOSE" = true ] && QUIET="" || QUIET="--quiet"
        if [ -z "$MVNW_USERNAME" ] || [ -z "$MVNW_PASSWORD" ]; then
            wget $QUIET "$wrapperUrl" -O "$wrapperJarPath" || rm -f
"$wrapperJarPath"
        else
            wget $QUIET --http-user="$MVNW_USERNAME" --http-
password="$MVNW_PASSWORD" "$wrapperUrl" -O "$wrapperJarPath" || rm -f
"$wrapperJarPath"
        fi
    elif command -v curl > /dev/null; then
        log "Found curl ... using curl"
        [ "$MVNW_VERBOSE" = true ] && QUIET="" || QUIET="--silent"
        if [ -z "$MVNW_USERNAME" ] || [ -z "$MVNW_PASSWORD" ]; then
            curl $QUIET -o "$wrapperJarPath" "$wrapperUrl" -f -L || rm -f
"$wrapperJarPath"
        else
            curl $QUIET --user "$MVNW_USERNAME:$MVNW_PASSWORD" -o
"$wrapperJarPath" "$wrapperUrl" -f -L || rm -f "$wrapperJarPath"
        fi
    else
        log "Falling back to using Java to download"
        javaSource="$MAVEN_PROJECTBASEDIR/.mvn/wrapper/
MavenWrapperDownloader.java"
        javaClass="$MAVEN_PROJECTBASEDIR/.mvn/wrapper/
MavenWrapperDownloader.class"
        # For Cygwin, switch paths to Windows format before running javac
        if $cygwin; then
          javaSource=$(cygpath --path --windows "$javaSource")
          javaClass=$(cygpath --path --windows "$javaClass")
        fi
        if [ -e "$javaSource" ]; then
            if [ ! -e "$javaClass" ]; then
                log " - Compiling MavenWrapperDownloader.java ..."
                ("$JAVA_HOME/bin/javac" "$javaSource")
            fi
            if [ -e "$javaClass" ]; then
                log " - Running MavenWrapperDownloader.java ..."
                ("$JAVA_HOME/bin/java" -cp .mvn/wrapper
```

```
MavenWrapperDownloader "$wrapperUrl" "$wrapperJarPath") || rm -f
"$wrapperJarPath"
            fi
        fi
    fi
fi
##########################################################################################
############
# End of extension
##########################################################################################
############
# If specified, validate the SHA-256 sum of the Maven wrapper jar file
wrapperSha256Sum=""
while IFS="=" read -r key value; do
  case "$key" in (wrapperSha256Sum) wrapperSha256Sum=$value; break ;;
  esac
done < "$MAVEN_PROJECTBASEDIR/.mvn/wrapper/maven-wrapper.properties"
if [ -n "$wrapperSha256Sum" ]; then
  wrapperSha256Result=false
  if command -v sha256sum > /dev/null; then
    if echo "$wrapperSha256Sum  $wrapperJarPath" | sha256sum -c > /dev/null
2>&1; then
      wrapperSha256Result=true
    fi
  elif command -v shasum > /dev/null; then
    if echo "$wrapperSha256Sum  $wrapperJarPath" | shasum -a 256 -c > /dev/
null 2>&1; then
      wrapperSha256Result=true
    fi
  else
    echo "Checksum validation was requested but neither 'sha256sum' or
'shasum' are available."
    echo "Please install either command, or disable validation by removing
'wrapperSha256Sum' from your maven-wrapper.properties."
    exit 1
  fi
  if [ $wrapperSha256Result = false ]; then
    echo "Error: Failed to validate Maven wrapper SHA-256, your Maven wrapper
might be compromised." >&2
    echo "Investigate or delete $wrapperJarPath to attempt a clean download."
>&2
    echo "If you updated your Maven version, you need to update the specified
wrapperSha256Sum property." >&2
    exit 1
  fi
fi
MAVEN_OPTS="$(concat_lines "$MAVEN_PROJECTBASEDIR/.mvn/jvm.config")
$MAVEN_OPTS"
# For Cygwin, switch paths to Windows format before running java
if $cygwin; then
  [ -n "$JAVA_HOME" ] &&
    JAVA_HOME=$(cygpath --path --windows "$JAVA_HOME")
  [ -n "$CLASSPATH" ] &&
    CLASSPATH=$(cygpath --path --windows "$CLASSPATH")
  [ -n "$MAVEN_PROJECTBASEDIR" ] &&
    MAVEN_PROJECTBASEDIR=$(cygpath --path --windows "$MAVEN_PROJECTBASEDIR")
fi
# Provide a "standardized" way to retrieve the CLI args that will
# work with both Windows and non-Windows executions.
MAVEN_CMD_LINE_ARGS="$MAVEN_CONFIG $*"
export MAVEN_CMD_LINE_ARGS
```

```
WRAPPER_LAUNCHER=org.apache.maven.wrapper.MavenWrapperMain
# shellcheck disable=SC2086 # safe args
exec "$JAVACMD" \
  $MAVEN_OPTS \
  $MAVEN_DEBUG_OPTS \
  -classpath "$MAVEN_PROJECTBASEDIR/.mvn/wrapper/maven-wrapper.jar" \
  "-Dmaven.multiModuleProjectDirectory=${MAVEN_PROJECTBASEDIR}" \
  ${WRAPPER_LAUNCHER} $MAVEN_CONFIG "$@"
```

```
@REM
----------------------------------------------------------------------------
@REM Licensed to the Apache Software Foundation (ASF) under one
@REM or more contributor license agreements.  See the NOTICE file
@REM distributed with this work for additional information
@REM regarding copyright ownership.  The ASF licenses this file
@REM to you under the Apache License, Version 2.0 (the
@REM "License"); you may not use this file except in compliance
@REM with the License.  You may obtain a copy of the License at
@REM
@REM    http://www.apache.org/licenses/LICENSE-2.0
@REM
@REM Unless required by applicable law or agreed to in writing,
@REM software distributed under the License is distributed on an
@REM "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
@REM KIND, either express or implied.  See the License for the
@REM specific language governing permissions and limitations
@REM under the License.
@REM
----------------------------------------------------------------------------
@REM
----------------------------------------------------------------------------
@REM Apache Maven Wrapper startup batch script, version 3.2.0
@REM
@REM Required ENV vars:
@REM JAVA_HOME - location of a JDK home dir
@REM
@REM Optional ENV vars
@REM MAVEN_BATCH_ECHO - set to 'on' to enable the echoing of the batch
commands
@REM MAVEN_BATCH_PAUSE - set to 'on' to wait for a keystroke before ending
@REM MAVEN_OPTS - parameters passed to the Java VM when running Maven
@REM     e.g. to debug Maven itself, use
@REM set MAVEN_OPTS=-Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000
@REM MAVEN_SKIP_RC - flag to disable loading of mavenrc files
@REM
----------------------------------------------------------------------------
@REM Begin all REM lines with '@' in case MAVEN_BATCH_ECHO is 'on'
@echo off
@REM set title of command window
title %0
@REM enable echoing by setting MAVEN_BATCH_ECHO to 'on'
@if "%MAVEN_BATCH_ECHO%" == "on"  echo %MAVEN_BATCH_ECHO%
@REM set %HOME% to equivalent of $HOME
if "%HOME%" == "" (set "HOME=%HOMEDRIVE%%HOMEPATH%")
@REM Execute a user defined script before this one
if not "%MAVEN_SKIP_RC%" == "" goto skipRcPre
@REM check for pre script, once with legacy .bat ending and once with .cmd
ending
if exist "%USERPROFILE%\mavenrc_pre.bat" call "%USERPROFILE%\mavenrc_pre.bat"
%*
if exist "%USERPROFILE%\mavenrc_pre.cmd" call "%USERPROFILE%\mavenrc_pre.cmd"
%*
:skipRcPre
@setlocal
set ERROR_CODE=0
@REM To isolate internal variables from possible post scripts, we use another
```

```
setlocal
@setlocal
@REM ==== START VALIDATION ====
if not "%JAVA_HOME%" == "" goto OkJHome
echo.
echo Error: JAVA_HOME not found in your environment. >&2
echo Please set the JAVA_HOME variable in your environment to match the >&2
echo location of your Java installation. >&2
echo.
goto error
:OkJHome
if exist "%JAVA_HOME%\bin\java.exe" goto init
echo.
echo Error: JAVA_HOME is set to an invalid directory. >&2
echo JAVA_HOME = "%JAVA_HOME%" >&2
echo Please set the JAVA_HOME variable in your environment to match the >&2
echo location of your Java installation. >&2
echo.
goto error
@REM ==== END VALIDATION ====
:init
@REM Find the project base dir, i.e. the directory that contains the folder
".mvn".
@REM Fallback to current working directory if not found.
set MAVEN_PROJECTBASEDIR=%MAVEN_BASEDIR%
IF NOT "%MAVEN_PROJECTBASEDIR%"=="" goto endDetectBaseDir
set EXEC_DIR=%CD%
set WDIR=%EXEC_DIR%
:findBaseDir
IF EXIST "%WDIR%"\.mvn goto baseDirFound
cd ..
IF "%WDIR%"=="%CD%" goto baseDirNotFound
set WDIR=%CD%
goto findBaseDir
:baseDirFound
set MAVEN_PROJECTBASEDIR=%WDIR%
cd "%EXEC_DIR%"
goto endDetectBaseDir
:baseDirNotFound
set MAVEN_PROJECTBASEDIR=%EXEC_DIR%
cd "%EXEC_DIR%"
:endDetectBaseDir
IF NOT EXIST "%MAVEN_PROJECTBASEDIR%\.mvn\jvm.config" goto
endReadAdditionalConfig
@setlocal EnableExtensions EnableDelayedExpansion
for /F "usebackq delims=" %%a in ("%MAVEN_PROJECTBASEDIR%\.mvn\jvm.config")
do set JVM_CONFIG_MAVEN_PROPS=!JVM_CONFIG_MAVEN_PROPS! %%a
@endlocal & set JVM_CONFIG_MAVEN_PROPS=%JVM_CONFIG_MAVEN_PROPS%
:endReadAdditionalConfig
SET MAVEN_JAVA_EXE="%JAVA_HOME%\bin\java.exe"
set WRAPPER_JAR="%MAVEN_PROJECTBASEDIR%\.mvn\wrapper\maven-wrapper.jar"
set WRAPPER_LAUNCHER=org.apache.maven.wrapper.MavenWrapperMain
set WRAPPER_URL="https://repo.maven.apache.org/maven2/org/apache/maven/
wrapper/maven-wrapper/3.2.0/maven-wrapper-3.2.0.jar"
FOR /F "usebackq tokens=1,2 delims==" %%A IN ("%MAVEN_PROJECTBASEDIR%
\.mvn\wrapper\maven-wrapper.properties") DO (
    IF "%%A"=="wrapperUrl" SET WRAPPER_URL=%%B
)
@REM Extension to allow automatically downloading the maven-wrapper.jar from
Maven-central
@REM This allows using the maven wrapper in projects that prohibit checking
```

```
in binary data.
if exist %WRAPPER_JAR% (
    if "%MVNW_VERBOSE%" == "true" (
        echo Found %WRAPPER_JAR%
    )
) else (
    if not "%MVNW_REPOURL%" == "" (
        SET WRAPPER_URL="%MVNW_REPOURL%/org/apache/maven/wrapper/maven-
wrapper/3.2.0/maven-wrapper-3.2.0.jar"
    )
    if "%MVNW_VERBOSE%" == "true" (
        echo Couldn't find %WRAPPER_JAR%, downloading it ...
        echo Downloading from: %WRAPPER_URL%
    )
    powershell -Command "&{"^
™"$webclient = new-object System.Net.WebClient;"^
™"if (-not ([string]::IsNullOrEmpty('%MVNW_USERNAME%') -and
[string]::IsNullOrEmpty('%MVNW_PASSWORD%'))) {"^
™"$webclient.Credentials = new-object
System.Net.NetworkCredential('%MVNW_USERNAME%', '%MVNW_PASSWORD%');"^
™"}"^
™"[Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::Tls12; $webclient.DownloadFile('%WRAPPER_URL%',
'%WRAPPER_JAR%')"^
™"}"
    if "%MVNW_VERBOSE%" == "true" (
        echo Finished downloading %WRAPPER_JAR%
    )
)
@REM End of extension
@REM If specified, validate the SHA-256 sum of the Maven wrapper jar file
SET WRAPPER_SHA_256_SUM=""
FOR /F "usebackq tokens=1,2 delims==" %%A IN ("%MAVEN_PROJECTBASEDIR%
\.mvn\wrapper\maven-wrapper.properties") DO (
    IF "%%A"=="wrapperSha256Sum" SET WRAPPER_SHA_256_SUM=%%B
)
IF NOT %WRAPPER_SHA_256_SUM%=="" (
    powershell -Command "&{"^
       "$hash = (Get-FileHash \"%WRAPPER_JAR%\" -Algorithm
SHA256).Hash.ToLower();"^
       "If('%WRAPPER_SHA_256_SUM%' -ne $hash){"^
       "  Write-Output 'Error: Failed to validate Maven wrapper SHA-256, your
Maven wrapper might be compromised.';"^
       "  Write-Output 'Investigate or delete %WRAPPER_JAR% to attempt a
clean download.';"^
       "  Write-Output 'If you updated your Maven version, you need to update
the specified wrapperSha256Sum property.';"^
       "  exit 1;"^
       "}"^
       "}"
    if ERRORLEVEL 1 goto error
)
@REM Provide a "standardized" way to retrieve the CLI args that will
@REM work with both Windows and non-Windows executions.
set MAVEN_CMD_LINE_ARGS=%*
%MAVEN_JAVA_EXE% ^
  %JVM_CONFIG_MAVEN_PROPS% ^
  %MAVEN_OPTS% ^
  %MAVEN_DEBUG_OPTS% ^
  -classpath %WRAPPER_JAR% ^
  "-Dmaven.multiModuleProjectDirectory=%MAVEN_PROJECTBASEDIR%" ^
```

```
  %WRAPPER_LAUNCHER% %MAVEN_CONFIG% %*
if ERRORLEVEL 1 goto error
goto end
:error
set ERROR_CODE=1
:end
@endlocal & set ERROR_CODE=%ERROR_CODE%
if not "%MAVEN_SKIP_RC%"=="" goto skipRcPost
@REM check for post script, once with legacy .bat ending and once with .cmd
ending
if exist "%USERPROFILE%\mavenrc_post.bat" call "%USERPROFILE%
\mavenrc_post.bat"
if exist "%USERPROFILE%\mavenrc_post.cmd" call "%USERPROFILE%
\mavenrc_post.cmd"
:skipRcPost
@REM pause the script if MAVEN_BATCH_PAUSE is set to 'on'
if "%MAVEN_BATCH_PAUSE%"=="on" pause
if "%MAVEN_TERMINATE_CMD%"=="on" exit %ERROR_CODE%
cmd /C exit /B %ERROR_CODE%
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-aggregator</artifactId>
    <version>0.23.0</version>
    <packaging>pom</packaging>
    <modules>
        <module>langchain4j-parent</module>
        <module>langchain4j-bom</module>
        <module>langchain4j-core</module>
        <module>langchain4j</module>
        <module>langchain4j-spring-boot-starter</module>
        <!-- model providers -->
        <module>langchain4j-azure-open-ai</module>
        <module>langchain4j-dashscope</module>
        <module>langchain4j-hugging-face</module>
        <module>langchain4j-local-ai</module>
        <module>langchain4j-open-ai</module>
        <module>langchain4j-vertex-ai</module>
        <!-- embedding stores -->
        <module>langchain4j-cassandra</module>
        <module>langchain4j-chroma</module>
        <module>langchain4j-elasticsearch</module>
        <module>langchain4j-milvus</module>
        <module>langchain4j-opensearch</module>
        <module>langchain4j-pinecone</module>
        <module>langchain4j-pgvector</module>
        <module>langchain4j-redis</module>
        <module>langchain4j-vespa</module>
        <module>langchain4j-weaviate</module>
    </modules>
</project>
```

README.md

LangChain for Java: Supercharge your Java application with the power of LLMs
[](https://twitter.com/intent/follow?screen_name=langchain4j)
[](https://discord.gg/V5qHrWNVf)
Project goals
The goal of this project is to simplify the integration of AI/LLM capabilities into your Java application.
This can be achieved thanks to:
- **A simple and coherent layer of abstractions**, designed to ensure that your code does not depend on concrete implementations such as LLM providers, embedding store providers, etc. This allows for easy swapping of components.
- **Numerous implementations of the above-mentioned abstractions**, providing you with a variety of LLMs and embedding stores to choose from.
- **Range of in-demand features on top of LLMs, such as:**
    - The capability to **ingest your own data** (documentation, codebase, etc.), allowing the LLM to act and respond based on your data.
    - **Autonomous agents** for delegating tasks (defined on the fly) to the LLM, which will strive to complete them.
    - **Prompt templates** to help you achieve the highest possible quality of LLM responses.
    - **Memory** to provide context to the LLM for your current and past conversations.
    - **Structured outputs** for receiving responses from the LLM with a desired structure as Java POJOs.
    - **"AI Services"** for declaratively defining complex AI behavior behind a simple API.
    - **Chains** to reduce the need for extensive boilerplate code in common use-cases.
    - **Auto-moderation** to ensure that all inputs and outputs to/from the LLM are not harmful.
Code examples
Please see examples of how LangChain4j can be used in `langchain4j-examples` repo:
- [Examples in plain Java](https://github.com/langchain4j/langchain4j-examples/tree/main/other-examples/src/main/java)
- [Example with Spring Boot](https://github.com/langchain4j/langchain4j-examples/blob/main/spring-boot-example/src/test/java/dev/example/CustomerSupportApplicationTest.java)
News
29 September:
- Updates to models API: return `Response<T>` instead of `T`. `Response<T>` contains token usage and finish reason.
- All model and embedding store integrations now live in their own modules
- Integration with [Vespa](https://vespa.ai/) by [@Heezer](https://github.com/Heezer)
- Integration with [Elasticsearch](https://www.elastic.co/) by [@Martin7-1](https://github.com/Martin7-1)
- Integration with [Redis](https://redis.io/) by [@Martin7-1](https://github.com/Martin7-1)
- Integration with [Milvus](https://milvus.io/) by [@IuriiKoval](https://github.com/IuriiKoval)
- Integration with [Astra DB](https://www.datastax.com/products/datastax-astra) and [Cassandra](https://cassandra.apache.org/) by [@clun](https://github.com/clun)
- Added support for overlap in document splitters
- Some bugfixes and smaller improvements
29 August:

- Offline [text classification with embeddings](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/embedding/classification/EmbeddingModelTextClassifierExample.java)
- Integration with [Google Vertex AI](https://cloud.google.com/vertex-ai) by [@kuraleta](https://github.com/kuraleta)
- Reworked [document splitters](https://github.com/langchain4j/langchain4j/blob/main/langchain4j/src/main/java/dev/langchain4j/data/document/splitter/DocumentSplitters.java)
- In-memory embedding store can now be easily persisted
- [And more](https://github.com/langchain4j/langchain4j/releases/tag/0.22.0)

19 August:
- Integration with [Azure OpenAI](https://learn.microsoft.com/en-us/azure/ai-services/openai/overview) by [@kuraleta](https://github.com/kuraleta)
- Integration with Qwen models (DashScope) by [@jiangsier-xyz](https://github.com/jiangsier-xyz)
- [Integration with Chroma](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/embedding/store/ChromaEmbeddingStoreExample.java) by [@kuraleta](https://github.com/kuraleta)
- [Support for persistent ChatMemory](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/ServiceWithPersistentMemoryForEachUserExample.java)

10 August:
- [Integration with Weaviate](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/embedding/store/WeaviateEmbeddingStoreExample.java) by [@Heezer](https://github.com/Heezer)
- [Support for DOC, XLS and PPT document types](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/DocumentLoaderExamples.java) by [@oognuyh](https://github.com/oognuyh)
- [Separate chat memory for each user](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/ServiceWithMemoryForEachUserExample.java)
- [Custom in-process embedding models](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/embedding/model/InProcessEmbeddingModelExamples.java)
- Added lots of Javadoc
- [And more](https://github.com/langchain4j/langchain4j/releases/tag/0.19.0)

26 July:
- We've added integration with [LocalAI](https://localai.io/). Now, you can use LLMs hosted locally!
- Added support for [response streaming in AI Services](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/ServiceWithStreamingExample.java).

21 July:
- Now, you can do [text embedding inside your JVM](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/embedding/model/InProcessEmbeddingModelExamples.java).

17 July:
- You can now try out OpenAI's `gpt-3.5-turbo` and `text-embedding-ada-002` models with LangChain4j for free, without needing an OpenAI account and keys! Simply use the API key "demo".

15 July:
- Added EmbeddingStoreIngestor
- Redesigned document loaders (see FileSystemDocumentLoader)
- Simplified ConversationalRetrievalChain
- Renamed DocumentSegment into TextSegment
- Added output parsers for numeric types
- Added @UserName for AI Services
- Fixed [23](https://github.com/langchain4j/langchain4j/issues/23) and [24](https://github.com/langchain4j/langchain4j/issues/24)

11 July:
- Added ["Dynamic Tools"](https://github.com/langchain4j/langchain4j-examples/

blob/main/other-examples/src/main/java/ServiceWithDynamicToolsExample.java):
  Now, the LLM can generate code for tasks that require precise calculations, such as math and string manipulation. This will be dynamically executed in a style akin to GPT-4's code interpreter!
  We use [Judge0, hosted by Rapid API](https://rapidapi.com/judge0-official/api/judge0-ce/pricing), for code execution. You can subscribe and receive 50 free executions per day.
5 July:
- Now you can [add your custom knowledge base to "AI Services"](https://github.com/langchain4j/langchain4j-examples/blob/main/spring-boot-example/src/test/java/dev/example/CustomerSupportApplicationTest.java).
  Relevant information will be automatically retrieved and injected into the prompt. This way, the LLM will have a
  context of your data and will answer based on it!
- The current date and time can now be automatically injected into the prompt using
  special `{{current_date}}`, `{{current_time}}` and `{{current_date_time}}` placeholders.
3 July:
- Added support for Spring Boot 3
2 July:
- [Added Spring Boot Starter](https://github.com/langchain4j/langchain4j-examples/blob/main/spring-boot-example/src/test/java/dev/example/CustomerSupportApplicationTest.java)
- Added support for HuggingFace models
1 July:
- [Added "Tools"](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/ServiceWithToolsExample.java) (support for OpenAI functions)
Highlights
You can declaratively define concise "AI Services" that are powered by LLMs:
```java
interface Assistant {
    String chat(String userMessage);
}
Assistant assistant = AiServices.create(Assistant.class, model);
String answer = assistant.chat("Hello");
System.out.println(answer);
// Hello! How can I assist you today?
```
You can use LLM as a classifier:
```java
enum Sentiment {
    POSITIVE, NEUTRAL, NEGATIVE
}
interface SentimentAnalyzer {
    @UserMessage("Analyze sentiment of {{it}}")
    Sentiment analyzeSentimentOf(String text);
    @UserMessage("Does {{it}} have a positive sentiment?")
    boolean isPositive(String text);
}
SentimentAnalyzer sentimentAnalyzer =
AiServices.create(SentimentAnalyzer.class, model);
Sentiment sentiment = sentimentAnalyzer.analyzeSentimentOf("It is good!");
// POSITIVE
boolean positive = sentimentAnalyzer.isPositive("It is bad!");
// false
```
You can easily extract structured information from unstructured data:
```java
class Person {
```

```
    private String firstName;
    private String lastName;
    private LocalDate birthDate;
    public String toString() {...}
}
interface PersonExtractor {
    @UserMessage("Extract information about a person from {{it}}")
    Person extractPersonFrom(String text);
}
PersonExtractor extractor = AiServices.create(PersonExtractor.class, model);
String text = "In 1968, amidst the fading echoes of Independence Day, "
    + "a child named John arrived under the calm evening sky. "
    + "This newborn, bearing the surname Doe, marked the start of a new
journey.";
Person person = extractor.extractPersonFrom(text);
// Person { firstName = "John", lastName = "Doe", birthDate = 1968-07-04 }
```
You can define more sophisticated prompt templates using mustache syntax:
```java
interface Translator {
    @SystemMessage("You are a professional translator into {{language}}")
    @UserMessage("Translate the following text: {{text}}")
    String translate(@V("text") String text, @V("language") String language);
}
Translator translator = AiServices.create(Translator.class, model);
String translation = translator.translate("Hello, how are you?", "Italian");
// Ciao, come stai?
```
You can provide tools that LLMs can use! Can be anything: retrieve
information from DB, call APIs, etc.
See example [here](https://github.com/langchain4j/langchain4j-examples/blob/
main/other-examples/src/main/java/ServiceWithToolsExample.java).
Compatibility
- Java: 8 or higher
- Spring Boot: 2 or 3
Getting started
1. Add LangChain4j OpenAI dependency to your project:
    - Maven:
      ```
      <dependency>
          <groupId>dev.langchain4j</groupId>
          <artifactId>langchain4j-open-ai</artifactId>
          <version>0.23.0</version>
      </dependency>
      ```
    - Gradle:
      ```
      implementation 'dev.langchain4j:langchain4j-open-ai:0.23.0'
      ```
2. Import your OpenAI API key:
    ```java
    String apiKey = System.getenv("OPENAI_API_KEY");
    ```
    You can use the API key "demo" to test OpenAI, which we provide for free.
    [How to gen an API key?](https://github.com/langchain4j/langchain4j#how-
to-get-an-api-key)
3. Create an instance of a model and start interacting:
    ```java
    OpenAiChatModel model = OpenAiChatModel.withApiKey(apiKey);
    String answer = model.generate("Hello world!");
    System.out.println(answer); // Hello! How can I assist you today?
```

```
## Disclaimer
Please note that the library is in active development and:
- Many features are still missing. We are working hard on implementing them
ASAP.
- API might change at any moment. At this point, we prioritize good design in
the future over backward compatibility
  now. We hope for your understanding.
- We need your input! Please [let us know](https://github.com/langchain4j/
langchain4j/issues/new/choose) what features you need and your concerns about
the current implementation.
## Current capabilities:
- AI Services:
    - [Simple](https://github.com/langchain4j/langchain4j-examples/blob/main/
other-examples/src/main/java/SimpleServiceExample.java)
    - [With Memory](https://github.com/langchain4j/langchain4j-examples/blob/
main/other-examples/src/main/java/ServiceWithMemoryExample.java)
    - [With Tools](https://github.com/langchain4j/langchain4j-examples/blob/
main/other-examples/src/main/java/ServiceWithToolsExample.java)
    - [With Streaming](https://github.com/langchain4j/langchain4j-examples/
blob/main/other-examples/src/main/java/ServiceWithStreamingExample.java)
    - [With Retriever](https://github.com/langchain4j/langchain4j-examples/
blob/main/other-examples/src/main/java/ServiceWithRetrieverExample.java)
    - [With Auto-Moderation](https://github.com/langchain4j/langchain4j-
examples/blob/main/other-examples/src/main/java/
ServiceWithAutoModerationExample.java)
    - [With Structured Outputs, Structured Prompts, etc](https://github.com/
langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/
OtherServiceExamples.java)
- Integration with [OpenAI](https://platform.openai.com/docs/introduction)
and [Azure OpenAI](https://learn.microsoft.com/en-us/azure/ai-services/openai/
overview) for:
    - [Chats](https://platform.openai.com/docs/guides/chat) (sync + streaming
+ functions)
    - [Completions](https://platform.openai.com/docs/guides/completion) (sync
+ streaming)
    - [Embeddings](https://platform.openai.com/docs/guides/embeddings)
- Integration with [Google Vertex AI](https://cloud.google.com/vertex-ai) for:
    - [Chats](https://cloud.google.com/vertex-ai/docs/generative-ai/chat/chat-
prompts)
    - [Completions](https://cloud.google.com/vertex-ai/docs/generative-ai/
text/text-overview)
    - [Embeddings](https://cloud.google.com/vertex-ai/docs/generative-ai/
embeddings/get-text-embeddings)
- Integration with [HuggingFace Inference API](https://huggingface.co/docs/
api-inference/index) for:
    - [Chats](https://huggingface.co/docs/api-inference/
detailed_parameters#text-generation-task)
    - [Completions](https://huggingface.co/docs/api-inference/
detailed_parameters#text-generation-task)
    - [Embeddings](https://huggingface.co/docs/api-inference/
detailed_parameters#feature-extraction-task)
- Integration with [LocalAI](https://localai.io/) for:
  - Chats (sync + streaming + functions)
  - Completions (sync + streaming)
  - Embeddings
- Integration with [DashScope](https://dashscope.aliyun.com/) for:
    - Chats (sync + streaming)
    - Completions (sync + streaming)
    - Embeddings
- [Chat memory](https://github.com/langchain4j/langchain4j-examples/blob/main/
```

other-examples/src/main/java/ChatMemoryExamples.java)
- [Persistent chat memory](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/ServiceWithPersistentMemoryForEachUserExample.java)
- [Chat with Documents](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/ChatWithDocumentsExamples.java)
- Integration with [Astra DB](https://www.datastax.com/products/datastax-astra) and [Cassandra](https://cassandra.apache.org/)
- [Integration](https://github.com/langchain4j/langchain4j-examples/blob/main/chroma-example/src/main/java/ChromaEmbeddingStoreExample.java) with [Chroma](https://www.trychroma.com/)
- [Integration](https://github.com/langchain4j/langchain4j-examples/blob/main/elasticsearch-example/src/main/java/ElasticsearchEmbeddingStoreExample.java) with [Elasticsearch](https://www.elastic.co/)
- [Integration](https://github.com/langchain4j/langchain4j-examples/blob/main/milvus-example/src/main/java/MilvusEmbeddingStoreExample.java) with [Milvus](https://milvus.io/)
- [Integration](https://github.com/langchain4j/langchain4j-examples/blob/main/pinecone-example/src/main/java/PineconeEmbeddingStoreExample.java) with [Pinecone](https://www.pinecone.io/)
- [Integration](https://github.com/langchain4j/langchain4j-examples/blob/main/redis-example/src/main/java/RedisEmbeddingStoreExample.java) with [Redis](https://redis.io/)
- [Integration](https://github.com/langchain4j/langchain4j-examples/blob/main/vespa-example/src/main/java/VespaEmbeddingStoreExample.java) with [Vespa](https://vespa.ai/)
- [Integration](https://github.com/langchain4j/langchain4j-examples/blob/main/weaviate-example/src/main/java/WeaviateEmbeddingStoreExample.java) with [Weaviate](https://weaviate.io/)
- [In-memory embedding store](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/embedding/store/InMemoryEmbeddingStoreExample.java) (can be persisted)
- [Structured outputs](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/OtherServiceExamples.java)
- [Prompt templates](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/PromptTemplateExamples.java)
- [Structured prompt templates](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/StructuredPromptTemplateExamples.java)
- [Streaming of LLM responses](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/StreamingExamples.java)
- [Loading txt, html, pdf, doc, xls and ppt documents from the file system and via URL](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/DocumentLoaderExamples.java)
- [Splitting documents into segments](https://github.com/langchain4j/langchain4j-examples/blob/main/other-examples/src/main/java/ChatWithDocumentsExamples.java):
    - by paragraphs, lines, sentences, words, etc
    - recursively
    - with overlap
- Token count estimation (so that you can predict how much you will pay)

Coming soon:
- Extending "AI Service" features
- Integration with more LLM providers (commercial and free)
- Integrations with more embedding stores (commercial and free)
- Support for more document types
- Long-term memory for chatbots and agents
- Chain-of-Thought and Tree-of-Thought

Request features
Please [let us know](https://github.com/langchain4j/langchain4j/issues/new/choose) what features you need!

Contribute
Please help us make this open-source library better by contributing.
Some guidelines:
1. Follow [Google's Best Practices for Java Libraries](https://jlbp.dev/).
2. Keep the code compatible with Java 8.
3. Avoid adding new dependencies as much as possible. If absolutely necessary, try to (re)use the same libraries which are already present.
4. Follow existing code styles present in the project.
5. Ensure to add Javadoc where necessary.
6. Provide unit and/or integration tests for your code.
7. Large features should be discussed with maintainers before implementation.
Use cases
You might ask why would I need all of this?
Here are a couple of examples:
- You want to implement a custom AI-powered chatbot that has access to your data and behaves the way you want it:
    - Customer support chatbot that can:
        - politely answer customer questions
        - take /change/cancel orders
    - Educational assistant that can:
        - Teach various subjects
        - Explain unclear parts
        - Assess user's understanding/knowledge
- You want to process a lot of unstructured data (files, web pages, etc) and extract structured information from them.
  For example:
    - extract insights from customer reviews and support chat history
    - extract interesting information from the websites of your competitors
    - extract insights from CVs of job applicants
- You want to generate information, for example:
    - Emails tailored for each of your customers
    - Content for your app/website:
        - Blog posts
        - Stories
- You want to transform information, for example:
    - Summarize
    - Proofread and rewrite
    - Translate
Best practices
We highly recommend watching [this amazing 90-minute tutorial](https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/) on prompt engineering best practices, presented by Andrew Ng (DeepLearning.AI) and Isa Fulford (OpenAI).
This course will teach you how to use LLMs efficiently and achieve the best possible results. Good investment of your time!
Here are some best practices for using LLMs:
- Be responsible. Use AI for Good.
- Be specific. The more specific your query, the best results you will get.
- Add a ["Let’s think step by step" instruction](https://arxiv.org/pdf/2205.11916.pdf) to your prompt.
- Specify steps to achieve the desired goal yourself. This will make the LLM do what you want it to do.
- Provide examples. Sometimes it is best to show LLM a few examples of what you want instead of trying to explain it.
- Ask LLM to provide structured output (JSON, XML, etc). This way you can parse response more easily and distinguish
  different parts of it.
- Use unusual delimiters, such as \```triple backticks``` to help the LLM distinguish

  data or input from instructions.
How to get an API key
You will need an API key from OpenAI (paid) or HuggingFace (free) to use LLMs hosted by them.
We recommend using OpenAI LLMs (`gpt-3.5-turbo` and `gpt-4`) as they are by far the most capable and are reasonably priced.
It will cost approximately $0.01 to generate 10 pages (A4 format) of text with `gpt-3.5-turbo`. With `gpt-4`, the cost will be $0.30 to generate the same amount of text. However, for some use cases, this higher cost may be justified.
[How to get OpenAI API key](https://www.howtogeek.com/885918/how-to-get-an-openai-api-key/).
For embeddings, we recommend using one of the models from the [HuggingFace MTEB leaderboard](https://huggingface.co/spaces/mteb/leaderboard).
You'll have to find the best one for your specific use case.
Here's how to get a HuggingFace API key:
- Create an account on https://huggingface.co
- Go to https://huggingface.co/settings/tokens
- Generate a new access token

chroma-example\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>chroma-example</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-chroma</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</project>
```

chroma-example\src\main\java\ChromaEmbeddingStoreExample.java

```
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.chroma.ChromaEmbeddingStore;
import java.util.List;
import static dev.langchain4j.internal.Utils.randomUUID;
public class ChromaEmbeddingStoreExample {
    /**
     * To run this example, ensure you have Chroma running locally. If not,
then:
     * - Execute "docker pull ghcr.io/chroma-core/chroma:0.4.6"
     * - Execute "docker run -d -p 8000:8000 ghcr.io/chroma-core/chroma:0.4.6"
     * - Wait until Chroma is ready to serve (may take a few minutes)
     */
    public static void main(String[] args) {
        EmbeddingStore<TextSegment> embeddingStore =
ChromaEmbeddingStore.builder()
                .baseUrl("http://localhost:8000")
                .collectionName(randomUUID())
                .build();
        EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
        TextSegment segment1 = TextSegment.from("I like football.");
        Embedding embedding1 = embeddingModel.embed(segment1).content();
        embeddingStore.add(embedding1, segment1);
        TextSegment segment2 = TextSegment.from("The weather is good today.");
        Embedding embedding2 = embeddingModel.embed(segment2).content();
        embeddingStore.add(embedding2, segment2);
        Embedding queryEmbedding = embeddingModel.embed("What is your
favourite sport?").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(queryEmbedding, 1);
        EmbeddingMatch<TextSegment> embeddingMatch = relevant.get(0);
        System.out.println(embeddingMatch.score()); // 0.8144288493114709
        System.out.println(embeddingMatch.embedded().text()); // I like
football.
    }
}
```

devoxx\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>devoxx</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-open-ai</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.tinylog</groupId>
            <artifactId>tinylog-impl</artifactId>
            <version>2.6.2</version>
        </dependency>
        <dependency>
            <groupId>org.tinylog</groupId>
            <artifactId>slf4j-tinylog</artifactId>
            <version>2.6.2</version>
        </dependency>
        <dependency>
            <groupId>org.mapdb</groupId>
            <artifactId>mapdb</artifactId>
            <version>3.0.9</version>
            <exclusions>
                <exclusion>
                    <groupId>org.jetbrains.kotlin</groupId>
                    <artifactId>kotlin-stdlib</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
</project>
```

devoxx\src\main\java\ApiKeys.java

```
public class ApiKeys {
    public static final String OPENAI_API_KEY =
System.getenv("OPENAI_API_KEY");
    public static final String RAPID_API_KEY = System.getenv("RAPID_API_KEY");
}
```

devoxx\src\main\java_00_HelloDevoxx.java

```
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
public class _00_HelloDevoxx {
    public static void main(String[] args) {
        ChatLanguageModel model =
OpenAiChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
        String response = model.generate("Say Hello Devoxx");
        System.out.println(response);
    }
}
```

devoxx\src\main\java_01_ModelParameters.java

```
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static java.time.Duration.ofSeconds;
public class _01_ModelParameters {
    public static void main(String[] args) {
    ’òò   & ÖWFW" ÖV æ–ær f÷" ÷ Vä ’ W‡ Æ –æVB †W&R ‡GG 3¢òð
platform.openai.com/docs/api-reference/chat/create
        ChatLanguageModel model = OpenAiChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .modelName(GPT_3_5_TURBO)
                .temperature(0.3)
                .timeout(ofSeconds(60))
                .logRequests(true)
                .logResponses(true)
                .build();
        String prompt = "Explain in three lines how to make a beautiful
painting";
        String response = model.generate(prompt);
        System.out.println(response);
    }
}
```

devoxx\src\main\java_02_PromptTemplate.java

```
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.PromptTemplate;
import dev.langchain4j.model.openai.OpenAiChatModel;
import java.util.HashMap;
import java.util.Map;
import static java.time.Duration.ofSeconds;
public class _02_PromptTemplate {
    public static void main(String[] args) {
        ChatLanguageModel model = OpenAiChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .timeout(ofSeconds(60))
                .build();
        String template = "Create a recipe for a {{dishType}} with the
following ingredients: {{ingredients}}";
        PromptTemplate promptTemplate = PromptTemplate.from(template);
        Map<String, Object> variables = new HashMap<>();
        variables.put("dishType", "oven dish");
        variables.put("ingredients", "potato, tomato, feta, olive oil");
        Prompt prompt = promptTemplate.apply(variables);
        String response = model.generate(prompt.text());
        System.out.println(response);
    }
}
```

devoxx\src\main\java_03_StructuredPromptTemplate.java

```
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.structured.StructuredPrompt;
import dev.langchain4j.model.input.structured.StructuredPromptProcessor;
import dev.langchain4j.model.openai.OpenAiChatModel;
import java.util.List;
import static java.time.Duration.ofSeconds;
import static java.util.Arrays.asList;
public class _03_StructuredPromptTemplate {
    @StructuredPrompt({
            "Create a recipe of a {{dish}} that can be prepared using only
{{ingredients}}.",
            "Structure your answer in the following way:",
            "Recipe name: ...",
            "Description: ...",
            "Preparation time: ...",
            "Required ingredients:",
            "- ...",
            "- ...",
            "Instructions:",
            "- ...",
            "- ..."
    })
    static class CreateRecipePrompt {
        String dish;
        List<String> ingredients;
        CreateRecipePrompt(String dish, List<String> ingredients) {
            this.dish = dish;
            this.ingredients = ingredients;
        }
    }
    public static void main(String[] args) {
        ChatLanguageModel model = OpenAiChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .timeout(ofSeconds(60))
                .build();
        CreateRecipePrompt createRecipePrompt = new CreateRecipePrompt(
                "salad",
                asList("cucumber", "tomato", "feta", "onion", "olives")
        );
        Prompt prompt =
StructuredPromptProcessor.toPrompt(createRecipePrompt);
        String recipe = model.generate(prompt.text());
        System.out.println(recipe);
    }
}
```

devoxx\src\main\java_04_Streaming.java

```
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.openai.OpenAiStreamingChatModel;
import dev.langchain4j.model.output.Response;
public class _04_Streaming {
    public static void main(String[] args) {
        OpenAiStreamingChatModel model =
OpenAiStreamingChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
        String prompt = "Write a short 1 paragraph funny poem about
developers and null-pointers";
        System.out.println("Nr of chars: " + prompt.length());
        System.out.println("Nr of tokens: " +
model.estimateTokenCount(prompt));
        model.generate(prompt, new StreamingResponseHandler<AiMessage>() {
            @Override
            public void onNext(String token) {
                System.out.print(token);
            }
            @Override
            public void onComplete(Response<AiMessage> response) {
                System.out.println("\n\nDone streaming");
            }
            @Override
            public void onError(Throwable error) {
                System.out.println("Something went wrong: " +
error.getMessage());
            }
        });
    }
}
```

devoxx\src\main\java_05_Memory.java

```
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.SystemMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.TokenWindowChatMemory;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.Tokenizer;
import dev.langchain4j.model.openai.OpenAiStreamingChatModel;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import dev.langchain4j.model.output.Response;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
public class _05_Memory {
    public static void main(String[] args) throws ExecutionException,
InterruptedException {
        OpenAiStreamingChatModel model =
OpenAiStreamingChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
        Tokenizer tokenizer = new OpenAiTokenizer(GPT_3_5_TURBO);
        ChatMemory chatMemory = TokenWindowChatMemory.withMaxTokens(1000,
tokenizer);
        UserMessage userMessage1 = userMessage(
                "How do I optimize database queries for a large-scale e-
commerce platform? "
                        + "Answer short in three to five lines maximum.");
        chatMemory.add(userMessage1);
        System.out.println("[User]: " + userMessage1.text());
        System.out.print("[LLM]: ");
        CompletableFuture<AiMessage> futureAiMessage = new
CompletableFuture<>();
        StreamingResponseHandler<AiMessage> handler = new
StreamingResponseHandler<AiMessage>() {
            @Override
            public void onNext(String token) {
                System.out.print(token);
            }
            @Override
            public void onComplete(Response<AiMessage> response) {
                futureAiMessage.complete(response.content());
            }
            @Override
            public void onError(Throwable throwable) {
            }
        };
        model.generate(chatMemory.messages(), handler);
        chatMemory.add(futureAiMessage.get());
        UserMessage userMessage2 = userMessage(
                "Give a concrete example implementation of the first point? "
+
                        "Be short, 10 lines of code maximum.");
        chatMemory.add(userMessage2);
        System.out.println("\n\n[User]: " + userMessage2.text());
        System.out.print("[LLM]: ");
        model.generate(chatMemory.messages(), handler);
    }
}
//SystemMessage systemMessage = SystemMessage.from(
```

```
//"You are a senior developer explaining to another senior developer, "
//      + "the project you are working on is an e-commerce platform with Java
back-end, " +
//      "Oracle database,and Spring Data JPA");
//chatMemory.add(systemMessage);
```

```
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.data.message.UserMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.openai.OpenAiStreamingChatModel;
import java.util.ArrayList;
import java.util.List;
import static java.time.Duration.ofSeconds;
public class _06_FewShot {
    public static void main(String[] args) {
        OpenAiStreamingChatModel model = OpenAiStreamingChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .timeout(ofSeconds(100))
                .build();
        List<ChatMessage> fewShotHistory = new ArrayList<>();
        // Adding positive feedback example to history
        fewShotHistory.add(UserMessage.from(
                "I love the new update! The interface is very user-friendly
and the new features are amazing!"));
        fewShotHistory.add(AiMessage.from(
                "Action: forward input to positive feedback storage\nReply:
Thank you very much for this great feedback! We have transmitted your message
to our product development team who will surely be very happy to hear this.
We hope you continue enjoying using our product."));
        // Adding negative feedback example to history
        fewShotHistory.add(UserMessage
                .from("I am facing frequent crashes after the new update on
my Android device."));
        fewShotHistory.add(AiMessage.from(
                "Action: open new ticket - crash after update Android\nReply:
We are so sorry to hear about the issues you are facing. We have reported the
problem to our development team and will make sure this issue is addressed as
fast as possible. We will send you an email when the fix is done, and we are
always at your service for any further assistance you may need."));
        // Adding another positive feedback example to history
        fewShotHistory.add(UserMessage
                .from("Your app has made my daily tasks so much easier! Kudos
to the team!"));
        fewShotHistory.add(AiMessage.from(
                "Action: forward input to positive feedback storage\nReply:
Thank you so much for your kind words! We are thrilled to hear that our app
is making your daily tasks easier. Your feedback has been shared with our
team. We hope you continue to enjoy using our app!"));
        // Adding another negative feedback example to history
        fewShotHistory.add(UserMessage
                .from("The new feature is not working as expected. It’s
causing data loss."));
        fewShotHistory.add(AiMessage.from(
                "Action: open new ticket - data loss by new feature\nReply:We
apologize for the inconvenience caused. Your feedback is crucial to us, and
we have reported this issue to our technical team. They are working on it on
priority. We will keep you updated on the progress and notify you once the
issue is resolved. Thank you for your patience and support."));
        // Adding real user's message
        ChatMessage customerComplaint = UserMessage
                .from("How can your app be so slow? Please do something about
it!");
        fewShotHistory.add(customerComplaint);
```

```
        System.out.println("[User]: " + customerComplaint.text());
        System.out.print("[LLM]: ");
        model.generate(fewShotHistory, new
StreamingResponseHandler<AiMessage>() {
            @Override
            public void onNext(String token) {
                System.out.print(token);
            }
            @Override
            public void onError(Throwable throwable) {
            }
        });
        // Extract reply and send to customer
        // Perform necessary action in back-end
    }
}
```

devoxx\src\main\java_07_ConversationalChain.java

```
import dev.langchain4j.chain.ConversationalChain;
import dev.langchain4j.model.openai.OpenAiChatModel;
import static java.time.Duration.ofSeconds;
public class _07_ConversationalChain {
    public static void main(String[] args) {
        OpenAiChatModel model = OpenAiChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .timeout(ofSeconds(60))
                .build();
        ConversationalChain chain = ConversationalChain.builder()
                .chatLanguageModel(model)
                // .chatMemory(...) // you can override default chat memory
                .build();
        String userMessage1 = "Can you give a brief explanation of the Agile
methodology, 3 lines max?";
        System.out.println("[User]: " + userMessage1);
        String answer1 = chain.execute(userMessage1);
        System.out.println("[LLM]: " + answer1);
        String userMessage2 = "What are good tools for that? 3 lines max.";
        System.out.println("[User]: " + userMessage2);
        String answer2 = chain.execute(userMessage2);
        System.out.println("[LLM]: " + answer2);
    }
}
```

devoxx\src\main\java_08_AIServiceExamples.java

```
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.input.structured.StructuredPrompt;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.output.structured.Description;
import dev.langchain4j.service.AiServices;
import dev.langchain4j.service.MemoryId;
import dev.langchain4j.service.SystemMessage;
import dev.langchain4j.service.UserMessage;
import dev.langchain4j.service.V;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.List;
import static java.time.Duration.ofSeconds;
import static java.util.Arrays.asList;
public class _08_AIServiceExamples {
    static ChatLanguageModel model = OpenAiChatModel.builder()
            .apiKey(ApiKeys.OPENAI_API_KEY)
            .timeout(ofSeconds(60))
            .build();
    ////////////////// SIMPLE EXAMPLE //////////////////////
    static class Simple_AI_Service_Example {
        interface Assistant {
            String chat(String message);
        }
        public static void main(String[] args) {
            Assistant assistant = AiServices.create(Assistant.class, model);
            String userMessage = "Translate 'Plus-Values des cessions de
valeurs mobilières, de droits sociaux et gains assimilés'";
            String answer = assistant.chat(userMessage);
            System.out.println(answer);
        }
    }
    ////////////////// WITH MESSAGE AND VARIABLES //////////////////////
    static class AI_Service_with_System_Message_Example {
        interface Chef {
            @SystemMessage("You are a professional chef. You are friendly,
polite and concise.")
            String answer(String question);
        }
        public static void main(String[] args) {
            Chef chef = AiServices.create(Chef.class, model);
            String answer = chef.answer("How long should I grill chicken?");
            System.out.println(answer); // Grilling chicken usually takes
around 10-15 minutes per side ...
        }
    }
    static class AI_Service_with_System_and_User_Messages_Example {
        interface TextUtils {
            @SystemMessage("You are a professional translator into
{{language}}")
            @UserMessage("Translate the following text: {{text}}")
            String translate(@V("text") String text, @V("language") String
language);
```

```
            @SystemMessage("Summarize every message from user in {{n}} bullet
points. Provide only bullet points.")
            List<String> summarize(@UserMessage String text, @V("n") int n);
        }
        public static void main(String[] args) {
            TextUtils utils = AiServices.create(TextUtils.class, model);
            String translation = utils.translate("Hello, how are you?",
"italian");
            System.out.println(translation); // Ciao, come stai?
            String text = "AI, or artificial intelligence, is a branch of
computer science that aims to create "
                    + "machines that mimic human intelligence. This can range
from simple tasks such as recognizing "
                    + "patterns or speech to more complex tasks like making
decisions or predictions.";
            List<String> bulletPoints = utils.summarize(text, 3);
            System.out.println(bulletPoints);
            // [
            // "- AI is a branch of computer science",
            // "- It aims to create machines that mimic human intelligence",
            // "- It can perform simple or complex tasks"
            // ]
        }
    }
    ////////////////////// EXTRACTING DIFFERENT DATA TYPES //////////////////////
    static class Sentiment_Extracting_AI_Service_Example {
        enum Sentiment {
            POSITIVE, NEUTRAL, NEGATIVE;
        }
        interface SentimentAnalyzer {
            @UserMessage("Analyze sentiment of {{it}}")
            Sentiment analyzeSentimentOf(String text);
            @UserMessage("Does {{it}} have a positive sentiment?")
            boolean isPositive(String text);
        }
        public static void main(String[] args) {
            SentimentAnalyzer sentimentAnalyzer =
AiServices.create(SentimentAnalyzer.class, model);
            Sentiment sentiment = sentimentAnalyzer.analyzeSentimentOf("It is
good!");
            System.out.println(sentiment); // POSITIVE
            boolean positive = sentimentAnalyzer.isPositive("It is bad!");
            System.out.println(positive); // false
        }
    }
    static class Number_Extracting_AI_Service_Example {
        interface NumberExtractor {
            @UserMessage("Extract number from {{it}}")
            int extractInt(String text);
            @UserMessage("Extract number from {{it}}")
            long extractLong(String text);
            @UserMessage("Extract number from {{it}}")
            BigInteger extractBigInteger(String text);
            @UserMessage("Extract number from {{it}}")
            float extractFloat(String text);
            @UserMessage("Extract number from {{it}}")
            double extractDouble(String text);
            @UserMessage("Extract number from {{it}}")
            BigDecimal extractBigDecimal(String text);
        }
        public static void main(String[] args) {
```

```
            NumberExtractor extractor =
AiServices.create(NumberExtractor.class, model);
            String text = "After countless millennia of computation, the
supercomputer Deep Thought finally announced "
                    + "that the answer to the ultimate question of life, the
universe, and everything was forty two.";
            int intNumber = extractor.extractInt(text);
            System.out.println(intNumber); // 42
            long longNumber = extractor.extractLong(text);
            System.out.println(longNumber); // 42
            BigInteger bigIntegerNumber = extractor.extractBigInteger(text);
            System.out.println(bigIntegerNumber); // 42
            float floatNumber = extractor.extractFloat(text);
            System.out.println(floatNumber); // 42.0
            double doubleNumber = extractor.extractDouble(text);
            System.out.println(doubleNumber); // 42.0
            BigDecimal bigDecimalNumber = extractor.extractBigDecimal(text);
            System.out.println(bigDecimalNumber); // 42.0
        }
    }
    static class Date_and_Time_Extracting_AI_Service_Example {
        interface DateTimeExtractor {
            @UserMessage("Extract date from {{it}}")
            LocalDate extractDateFrom(String text);
            @UserMessage("Extract time from {{it}}")
            LocalTime extractTimeFrom(String text);
            @UserMessage("Extract date and time from {{it}}")
            LocalDateTime extractDateTimeFrom(String text);
        }
        public static void main(String[] args) {
            DateTimeExtractor extractor =
AiServices.create(DateTimeExtractor.class, model);
            String text = "The tranquility pervaded the evening of 1968, just
fifteen minutes shy of midnight,"
                    + " following the celebrations of Independence Day.";
            LocalDate date = extractor.extractDateFrom(text);
            System.out.println(date); // 1968-07-04
            LocalTime time = extractor.extractTimeFrom(text);
            System.out.println(time); // 23:45
            LocalDateTime dateTime = extractor.extractDateTimeFrom(text);
            System.out.println(dateTime); // 1968-07-04T23:45
        }
    }
    static class POJO_Extracting_AI_Service_Example {
        static class Person {
            private String firstName;
            private String lastName;
            private LocalDate birthDate;
            @Override
            public String toString() {
                return "Person {" +
                        " firstName = \"" + firstName + "\"" +
                        ", lastName = \"" + lastName + "\"" +
                        ", birthDate = " + birthDate +
                        " }";
            }
        }
        interface PersonExtractor {
            @UserMessage("Extract information about a person from {{it}}")
            Person extractPersonFrom(String text);
        }
```

```
        public static void main(String[] args) {
            PersonExtractor extractor =
AiServices.create(PersonExtractor.class, model);
            String text = "In 1968, amidst the fading echoes of Independence
Day, "
                    + "a child named John arrived under the calm evening sky.
"
                    + "This newborn, bearing the surname Doe, marked the
start of a new journey.";
            Person person = extractor.extractPersonFrom(text);
            System.out.println(person); // Person { firstName = "John",
lastName = "Doe", birthDate = 1968-07-04 }
        }
    }
    ////////////////////// DESCRIPTIONS ////////////////////////
    static class POJO_With_Descriptions_Extracting_AI_Service_Example {
        static class Recipe {
            @Description("short title, 3 words maximum")
            private String title;
            @Description("short description, 2 sentences maximum")
            private String description;
            @Description("each step should be described in 6 to 8 words,
steps should rhyme with each other")
            private List<String> steps;
            private Integer preparationTimeMinutes;
            @Override
            public String toString() {
                return "Recipe {" +
                        " title = \"" + title + "\"" +
                        ", description = \"" + description + "\"" +
                        ", steps = " + steps +
                        ", preparationTimeMinutes = " +
preparationTimeMinutes +
                        " }";
            }
        }
        @StructuredPrompt("Create a recipe of a {{dish}} that can be prepared
using only {{ingredients}}")
        static class CreateRecipePrompt {
            private String dish;
            private List<String> ingredients;
        }
        interface Chef {
            Recipe createRecipeFrom(String... ingredients);
            Recipe createRecipe(CreateRecipePrompt prompt);
        }
        public static void main(String[] args) {
            Chef chef = AiServices.create(Chef.class, model);
            Recipe recipe = chef.createRecipeFrom("cucumber", "tomato",
"feta", "onion", "olives", "lemon");
            System.out.println(recipe);
            // Recipe {
            // title = "Greek Salad",
            // description = "A refreshing mix of veggies and feta cheese in
a zesty
            // dressing.",
            // steps = [
            // "Chop cucumber and tomato",
            // "Add onion and olives",
            // "Crumble feta on top",
            // "Drizzle with dressing and enjoy!"
```

```
            // ],
            // preparationTimeMinutes = 10
            // }
            CreateRecipePrompt prompt = new CreateRecipePrompt();
            prompt.dish = "oven dish";
            prompt.ingredients = asList("cucumber", "tomato", "feta",
"onion", "olives", "potatoes");
            Recipe anotherRecipe = chef.createRecipe(prompt);
            System.out.println(anotherRecipe);
            // Recipe ...
        }
    }
    ////////////////////////// WITH MEMORY ////////////////////////
    static class ServiceWithMemoryExample {
        interface Assistant {
            String chat(String message);
        }
        public static void main(String[] args) {
            ChatMemory chatMemory =
MessageWindowChatMemory.withMaxMessages(10);
            Assistant assistant = AiServices.builder(Assistant.class)
                    .chatLanguageModel(model)
                    .chatMemory(chatMemory)
                    .build();
            String answer = assistant.chat("Hello! My name is Klaus.");
            System.out.println(answer); // Hello Klaus! How can I assist you
today?
            String answerWithName = assistant.chat("What is my name?");
            System.out.println(answerWithName); // Your name is Klaus.
        }
    }
    static class ServiceWithMemoryForEachUserExample {
        interface Assistant {
            String chat(@MemoryId int memoryId, @UserMessage String
userMessage);
        }
        public static void main(String[] args) {
            Assistant assistant = AiServices.builder(Assistant.class)
                    .chatLanguageModel(model)
                    .chatMemoryProvider(memoryId ->
MessageWindowChatMemory.withMaxMessages(10))
                    .build();
            System.out.println(assistant.chat(1, "Hello, my name is Klaus"));
            // Hi Klaus! How can I assist you today?
            System.out.println(assistant.chat(2, "Hello, my name is
Francine"));
            // Hello Francine! How can I assist you today?
            System.out.println(assistant.chat(1, "What is my name?"));
            // Your name is Klaus.
            System.out.println(assistant.chat(2, "What is my name?"));
            // Your name is Francine.
        }
    }
}
```

devoxx\src\main\java_09_ServiceWithPersistentMemoryForEachUserExample.java

```
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.memory.chat.ChatMemoryProvider;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
import dev.langchain4j.service.MemoryId;
import dev.langchain4j.service.UserMessage;
import dev.langchain4j.store.memory.chat.ChatMemoryStore;
import org.mapdb.DB;
import org.mapdb.DBMaker;
import java.util.List;
import java.util.Map;
import static
dev.langchain4j.data.message.ChatMessageDeserializer.messagesFromJson;
import static
dev.langchain4j.data.message.ChatMessageSerializer.messagesToJson;
import static org.mapdb.Serializer.INTEGER;
import static org.mapdb.Serializer.STRING;
public class _09_ServiceWithPersistentMemoryForEachUserExample {
    interface Assistant {
        String chat(@MemoryId int memoryId, @UserMessage String userMessage);
    }
    public static void main(String[] args) {
        PersistentChatMemoryStore store = new PersistentChatMemoryStore();
        ChatMemoryProvider chatMemoryProvider = memoryId ->
MessageWindowChatMemory.builder()
                .id(memoryId)
                .maxMessages(10)
                .chatMemoryStore(store)
                .build();
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPENAI_A
PI_KEY))
                .chatMemoryProvider(chatMemoryProvider)
                .build();
        System.out.println(assistant.chat(1, "Hello, my name is Klaus"));
        System.out.println(assistant.chat(2, "Hi, my name is Francine"));
        // Now, comment out the two lines above, uncomment the two lines
below, and run again.
        // System.out.println(assistant.chat(1, "What is my name?"));
        // System.out.println(assistant.chat(2, "What is my name?"));
    }
    // You can create your own implementation of ChatMemoryStore and store
chat memory whenever you'd like
    static class PersistentChatMemoryStore implements ChatMemoryStore {
        private final DB db = DBMaker.fileDB("multi-user-chat-
memory.db").transactionEnable().make();
        private final Map<Integer, String> map = db.hashMap("messages",
INTEGER, STRING).createOrOpen();
        @Override
        public List<ChatMessage> getMessages(Object memoryId) {
            String json = map.get((int) memoryId);
            return messagesFromJson(json);
        }
        @Override
        public void updateMessages(Object memoryId, List<ChatMessage>
messages) {
            String json = messagesToJson(messages);
```

```
            map.put((int) memoryId, json);
            db.commit();
        }
        @Override
        public void deleteMessages(Object memoryId) {
            map.remove((int) memoryId);
            db.commit();
        }
    }
}
```

devoxx\src\main\java_10_ServiceWithToolsExample.java

```
import dev.langchain4j.agent.tool.Tool;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
public class _10_ServiceWithToolsExample {
    // Please also check CustomerSupportApplication and
CustomerSupportApplicationTest
    // from spring-boot-example module
    static class Calculator {
        @Tool("Calculates the length of a string")
        int stringLength(String s) {
            return s.length();
        }
        @Tool("Calculates the sum of two numbers")
        int add(int a, int b) {
            return a + b;
        }
        @Tool("Calculates the square root of a number")
        double sqrt(int x) {
            return Math.sqrt(x);
        }
    }
    interface Assistant {
        String chat(String userMessage);
    }
    public static void main(String[] args) {
        ChatLanguageModel model = OpenAiChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .logRequests(false)
                .build();
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(model)
                .tools(new Calculator())
                .chatMemory(MessageWindowChatMemory.withMaxMessages(10))
                .build();
        String question = "What is the square root of the sum of the numbers
of letters in the words \"hello\" and \"world\"?";
        String answer = assistant.chat(question);
        System.out.println(answer);
        // The square root of the sum of the number of letters in the words
"hello" and "world" is approximately 3.162.
    }
}
```

devoxx\src\main\java_11_ServiceWithDynamicToolsExample.java

```
import dev.langchain4j.code.Judge0JavaScriptExecutionTool;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
import static java.time.Duration.ofSeconds;
public class _11_ServiceWithDynamicToolsExample {
    interface Assistant {
        String chat(String message);
    }
    public static void main(String[] args) {
        Judge0JavaScriptExecutionTool judge0Tool = new
Judge0JavaScriptExecutionTool(ApiKeys.RAPID_API_KEY);
        ChatLanguageModel chatLanguageModel = OpenAiChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .temperature(0.0)
                .timeout(ofSeconds(60))
                .build();
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(MessageWindowChatMemory.withMaxMessages(20))
                .tools(judge0Tool)
                .build();
        interact(assistant, "What is the square root of 49506838032859?");
        interact(assistant, "Capitalize every third letter: abcabc");
        interact(assistant, "What is the number of hours between 17:00 on 21
Feb 1988 and 04:00 on 12 Apr 2014?");
    }
    private static void interact(Assistant assistant, String userMessage) {
        System.out.println("[User]: " + userMessage);
        String answer = assistant.chat(userMessage);
        System.out.println("[Assistant]: " + answer);
        System.out.println();
        System.out.println();
    }
}
```

devoxx\src\main\java_12_ChatWithDocumentsExamples.java

```
import dev.langchain4j.chain.ConversationalRetrievalChain;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.document.splitter.DocumentSplitters;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.PromptTemplate;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import dev.langchain4j.retriever.EmbeddingStoreRetriever;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.EmbeddingStoreIngestor;
import dev.langchain4j.store.embedding.inmemory.InMemoryEmbeddingStore;
import java.net.URISyntaxException;
import java.net.URL;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static
dev.langchain4j.data.document.FileSystemDocumentLoader.loadDocument;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static java.util.stream.Collectors.joining;
public class _12_ChatWithDocumentsExamples {
    static class IfYouNeedSimplicity {
        public static void main(String[] args) throws Exception {
            EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
            EmbeddingStore<TextSegment> embeddingStore = new
InMemoryEmbeddingStore<>();
            EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
                    .documentSplitter(DocumentSplitters.recursive(500, 0))
                    .embeddingModel(embeddingModel)
                    .embeddingStore(embeddingStore)
                    .build();
            Document document = loadDocument(toPath("story-about-happy-
carrot.txt"));
            ingestor.ingest(document);
            ConversationalRetrievalChain chain =
ConversationalRetrievalChain.builder()
                    .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPEN
AI_API_KEY))
                    .retriever(EmbeddingStoreRetriever.from(embeddingStore,
embeddingModel))
                    // .chatMemory() // you can override default chat memory
                    // .promptTemplate() // you can override default prompt
template
                    .build();
            String answer = chain.execute("Who is Charlie?");
            System.out.println(answer); // Charlie is a cheerful carrot
living in VeggieVille...
        }
```

```
    }
    static class If_You_Need_More_Control {
        public static void main(String[] args) {
            // Load the document that includes the information you'd like to
"chat" about with the model.
            Document document = loadDocument(toPath("story-about-happy-
carrot.txt"));
            // Split document into segments 100 tokens each
            DocumentSplitter splitter = DocumentSplitters.recursive(
                    100,
                    0,
                    new OpenAiTokenizer(GPT_3_5_TURBO)
            );
            List<TextSegment> segments = splitter.split(document);
            // Embed segments (convert them into vectors that represent the
meaning) using embedding model
            EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
            List<Embedding> embeddings =
embeddingModel.embedAll(segments).content();
            // Store embeddings into embedding store for further search /
retrieval
            EmbeddingStore<TextSegment> embeddingStore = new
InMemoryEmbeddingStore<>();
            embeddingStore.addAll(embeddings, segments);
            // Specify the question you want to ask the model
            String question = "Who is Charlie?";
            // Embed the question
            Embedding questionEmbedding =
embeddingModel.embed(question).content();
            // Find relevant embeddings in embedding store by semantic
similarity
            // You can play with parameters below to find a sweet spot for
your specific use case
            int maxResults = 3;
            double minScore = 0.7;
            List<EmbeddingMatch<TextSegment>> relevantEmbeddings
                    = embeddingStore.findRelevant(questionEmbedding,
maxResults, minScore);
            // Create a prompt for the model that includes question and
relevant embeddings
            PromptTemplate promptTemplate = PromptTemplate.from(
                    "Answer the following question to the best of your
ability:\n"
                            + "\n"
                            + "Question:\n"
                            + "{{question}}\n"
                            + "\n"
                            + "Base your answer on the following information:
\n"
                            + "{{information}}");
            String information = relevantEmbeddings.stream()
                    .map(match -> match.embedded().text())
                    .collect(joining("\n\n"));
            Map<String, Object> variables = new HashMap<>();
            variables.put("question", question);
            variables.put("information", information);
            Prompt prompt = promptTemplate.apply(variables);
            // Send the prompt to the OpenAI chat model
            ChatLanguageModel chatModel =
OpenAiChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
            AiMessage aiMessage =
```

```
chatModel.generate(prompt.toUserMessage()).content();
            // See an answer from the model
            String answer = aiMessage.text();
            System.out.println(answer); // Charlie is a cheerful carrot
living in VeggieVille...
        }
    }
    private static Path toPath(String fileName) {
        try {
            URL fileUrl =
_12_ChatWithDocumentsExamples.class.getResource(fileName);
            return Paths.get(fileUrl.toURI());
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
    }
}
```

devoxx\src\main\resources\story-about-happy-carrot.txt

Once upon a time in the town of VeggieVille, there lived a cheerful carrot named Charlie.
Charlie was a radiant carrot, always beaming with joy and positivity.
His vibrant orange skin and lush green top were a sight to behold, but it was his infectious laughter and warm personality that really set him apart.
Charlie had a diverse group of friends, each a vegetable with their own unique characteristics.
There was Bella the blushing beetroot, always ready with a riddle or two; Timmy the timid tomato, a gentle soul with a heart of gold; and Percy the prankster potato, whose jokes always brought a smile to everyone's faces.
Despite their differences, they shared a close bond, their friendship as robust as their natural goodness.
Their lives were filled with delightful adventures, from playing hide-and-seek amidst the leafy lettuce to swimming in the dewy droplets that pooled on the cabbage leaves.
Their favorite place, though, was the sunlit corner of the vegetable patch, where they would bask in the warmth of the sun, share stories, and have hearty laughs.
One day, a bunch of pesky caterpillars invaded VeggieVille.
The vegetables were terrified, fearing they would be nibbled to nothingness.
But Charlie, with his usual sunny disposition, had an idea.
He proposed they host a grand feast for the caterpillars, with the juiciest leaves from the outskirts of the town.
Charlie's optimism was contagious, and his friends eagerly joined in to prepare the feast.
When the caterpillars arrived, they were pleasantly surprised.
They enjoyed the feast and were so impressed with the vegetables' hospitality that they promised not to trouble VeggieVille again.
In return, they agreed to help pollinate the flowers, contributing to a more lush and vibrant VeggieVille.
Charlie's idea had saved the day, but he humbly attributed the success to their teamwork and friendship.
They celebrated their victory with a grand party, filled with laughter, dance, and merry games.
That night, under the twinkling stars, they made a pact to always stand by each other, come what may.
From then on, the story of the happy carrot and his friends spread far and wide, a tale of friendship, unity, and positivity.
Charlie, Bella, Timmy, and Percy continued to live their joyful lives, their laughter echoing through VeggieVille.
And so, the tale of the happy carrot and his friends serves as a reminder that no matter the challenge, with optimism, teamwork, and a bit of creativity, anything is possible.

devoxx\src\main\resources\tinylog.properties

```
writer.level = info
```

elasticsearch-example\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>elasticsearch-example</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-elasticsearch</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</project>
```

elasticsearch-example\src\main\java\ElasticsearchEmbeddingStoreExample.java

```
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import
dev.langchain4j.store.embedding.elasticsearch.ElasticsearchEmbeddingStore;
import java.util.List;
public class ElasticsearchEmbeddingStoreExample {
    /**
     * To run this example, ensure you have Elasticsearch running locally. If
not, then:
     * - Execute "docker pull docker.elastic.co/elasticsearch/
elasticsearch:8.9.0"
     * - Execute "docker run -d -p 9200:9200 -p 9300:9300 -e
discovery.type=single-node -e xpack.security.enabled=false docker.elastic.co/
elasticsearch/elasticsearch:8.9.0"
     * - Wait until Elasticsearch is ready to serve (may take a few minutes)
     */
    public static void main(String[] args) throws InterruptedException {
        EmbeddingStore<TextSegment> embeddingStore =
ElasticsearchEmbeddingStore.builder()
                .serverUrl("http://localhost:9200")
                .build();
        EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
        TextSegment segment1 = TextSegment.from("I like football.");
        Embedding embedding1 = embeddingModel.embed(segment1).content();
        embeddingStore.add(embedding1, segment1);
        TextSegment segment2 = TextSegment.from("The weather is good today.");
        Embedding embedding2 = embeddingModel.embed(segment2).content();
        embeddingStore.add(embedding2, segment2);
        Thread.sleep(1000); // to be sure that embeddings were persisted
        Embedding queryEmbedding = embeddingModel.embed("What is your
favourite sport?").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(queryEmbedding, 1);
        EmbeddingMatch<TextSegment> embeddingMatch = relevant.get(0);
        System.out.println(embeddingMatch.score()); // 0.81442887
        System.out.println(embeddingMatch.embedded().text()); // I like
football.
    }
}
```

LICENSE

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.
5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!) The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

milvus-example\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>milvus-example</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-milvus</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</project>
```

milvus-example\src\main\java\MilvusEmbeddingStoreExample.java

```
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.milvus.MilvusEmbeddingStore;
import java.util.List;
public class MilvusEmbeddingStoreExample {
    /**
     * First run Milvus locally:
     * Run "docker compose up -d" inside "../resources" directory.
     * If you want to create a fresh Milvus instance, don't forget to remove
"../resources/volumes" directory.
     */
    public static void main(String[] args) {
        EmbeddingStore<TextSegment> embeddingStore =
MilvusEmbeddingStore.builder()
                .host("localhost")
                .port(19530)
                .dimension(384)
                .build();
        EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
        TextSegment segment1 = TextSegment.from("I like football.");
        Embedding embedding1 = embeddingModel.embed(segment1).content();
        embeddingStore.add(embedding1, segment1);
        TextSegment segment2 = TextSegment.from("The weather is good today.");
        Embedding embedding2 = embeddingModel.embed(segment2).content();
        embeddingStore.add(embedding2, segment2);
        Embedding queryEmbedding = embeddingModel.embed("What is your
favourite sport?").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(queryEmbedding, 1);
        EmbeddingMatch<TextSegment> embeddingMatch = relevant.get(0);
        System.out.println(embeddingMatch.score()); // 0.8144287765026093
        System.out.println(embeddingMatch.embedded().text()); // I like
football.
    }
}
```

other-examples\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>other-examples</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-open-ai</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-hugging-face</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-vertex-ai</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.mapdb</groupId>
            <artifactId>mapdb</artifactId>
            <version>3.0.9</version>
            <exclusions>
                <exclusion>
                    <groupId>org.jetbrains.kotlin</groupId>
                    <artifactId>kotlin-stdlib</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.tinylog</groupId>
```

```
            <artifactId>tinylog-impl</artifactId>
            <version>2.6.2</version>
        </dependency>
        <dependency>
            <groupId>org.tinylog</groupId>
            <artifactId>slf4j-tinylog</artifactId>
            <version>2.6.2</version>
        </dependency>
    </dependencies>
</project>
```

other-examples\src\main\java\ApiKeys.java

```
public class ApiKeys {
    // You can use "demo" api key for demonstration purposes.
    // You can get your own OpenAI API key here: https://platform.openai.com/
account/api-keys
    public static final String OPENAI_API_KEY = "demo";
    // You can get your own HuggingFace API key here: https://huggingface.co/
settings/tokens
    public static final String HF_API_KEY = System.getenv("HF_API_KEY");
    // You can get your own Judge0 RapidAPI key here: https://rapidapi.com/
judge0-official/api/judge0-ce
    public static final String RAPID_API_KEY = System.getenv("RAPID_API_KEY");
}
```

other-examples\src\main\java\ChatMemoryExamples.java

```
import dev.langchain4j.chain.ConversationalChain;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.TokenWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import java.io.IOException;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
public class ChatMemoryExamples {
    // See also ServiceWithMemoryExample and
ServiceWithMemoryForEachUserExample
    public static class ConversationalChain_Example {
        public static void main(String[] args) throws IOException {
            ConversationalChain chain = ConversationalChain.builder()
                    .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPEN
AI_API_KEY))
                    // .chatMemory() // you can override default chat memory
                    .build();
            String answer = chain.execute("Hello, my name is Klaus");
            System.out.println(answer); // Hello Klaus! How can I assist you
today?
            String answerWithName = chain.execute("What is my name?");
            System.out.println(answerWithName); // Your name is Klaus.
        }
    }
    public static class If_You_Need_More_Control {
        public static void main(String[] args) {
            ChatLanguageModel model =
OpenAiChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
            ChatMemory chatMemory = TokenWindowChatMemory.withMaxTokens(300,
new OpenAiTokenizer(GPT_3_5_TURBO));
            // You have full control over the chat memory.
            // You can decide if you want to add a particular message to the
memory
            // (e.g. you might not want to store few-shot examples to save on
tokens).
            // You can process/modify the message before saving if required.
            chatMemory.add(userMessage("Hello, my name is Klaus"));
            AiMessage answer =
model.generate(chatMemory.messages()).content();
            System.out.println(answer.text()); // Hello Klaus! How can I
assist you today?
            chatMemory.add(answer);
            chatMemory.add(userMessage("What is my name?"));
            AiMessage answerWithName =
model.generate(chatMemory.messages()).content();
            System.out.println(answerWithName.text()); // Your name is Klaus.
            chatMemory.add(answerWithName);
        }
    }
}
```

other-examples\src\main\java\ChatWithDocumentsExamples.java

```
import dev.langchain4j.chain.ConversationalRetrievalChain;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.document.splitter.DocumentSplitters;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.PromptTemplate;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import dev.langchain4j.retriever.EmbeddingStoreRetriever;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.EmbeddingStoreIngestor;
import dev.langchain4j.store.embedding.inmemory.InMemoryEmbeddingStore;
import java.net.URISyntaxException;
import java.net.URL;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static
dev.langchain4j.data.document.FileSystemDocumentLoader.loadDocument;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
import static java.util.stream.Collectors.joining;
public class ChatWithDocumentsExamples {
    // Please also check ServiceWithRetrieverExample
    static class IfYouNeedSimplicity {
        public static void main(String[] args) throws Exception {
            EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
            EmbeddingStore<TextSegment> embeddingStore = new
InMemoryEmbeddingStore<>();
            EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
                    .documentSplitter(DocumentSplitters.recursive(500, 0))
                    .embeddingModel(embeddingModel)
                    .embeddingStore(embeddingStore)
                    .build();
            Document document = loadDocument(toPath("example-files/story-
about-happy-carrot.txt"));
            ingestor.ingest(document);
            ConversationalRetrievalChain chain =
ConversationalRetrievalChain.builder()
                    .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPEN
AI_API_KEY))
                    .retriever(EmbeddingStoreRetriever.from(embeddingStore,
embeddingModel))
                    // .chatMemory() // you can override default chat memory
                    // .promptTemplate() // you can override default prompt
template
                    .build();
            String answer = chain.execute("Who is Charlie?");
            System.out.println(answer); // Charlie is a cheerful carrot
living in VeggieVille...
```

```
        }
    }
    static class If_You_Need_More_Control {
        public static void main(String[] args) {
            // Load the document that includes the information you'd like to
"chat" about with the model.
            Document document = loadDocument(toPath("example-files/story-
about-happy-carrot.txt"));
            // Split document into segments 100 tokens each
            DocumentSplitter splitter = DocumentSplitters.recursive(
                    100,
                    0,
                    new OpenAiTokenizer(GPT_3_5_TURBO)
            );
            List<TextSegment> segments = splitter.split(document);
            // Embed segments (convert them into vectors that represent the
meaning) using embedding model
            EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
            List<Embedding> embeddings =
embeddingModel.embedAll(segments).content();
            // Store embeddings into embedding store for further search /
retrieval
            EmbeddingStore<TextSegment> embeddingStore = new
InMemoryEmbeddingStore<>();
            embeddingStore.addAll(embeddings, segments);
            // Specify the question you want to ask the model
            String question = "Who is Charlie?";
            // Embed the question
            Embedding questionEmbedding =
embeddingModel.embed(question).content();
            // Find relevant embeddings in embedding store by semantic
similarity
            // You can play with parameters below to find a sweet spot for
your specific use case
            int maxResults = 3;
            double minScore = 0.7;
            List<EmbeddingMatch<TextSegment>> relevantEmbeddings
                    = embeddingStore.findRelevant(questionEmbedding,
maxResults, minScore);
            // Create a prompt for the model that includes question and
relevant embeddings
            PromptTemplate promptTemplate = PromptTemplate.from(
                    "Answer the following question to the best of your
ability:\n"
                            + "\n"
                            + "Question:\n"
                            + "{{question}}\n"
                            + "\n"
                            + "Base your answer on the following information:
\n"
                            + "{{information}}");
            String information = relevantEmbeddings.stream()
                    .map(match -> match.embedded().text())
                    .collect(joining("\n\n"));
            Map<String, Object> variables = new HashMap<>();
            variables.put("question", question);
            variables.put("information", information);
            Prompt prompt = promptTemplate.apply(variables);
            // Send the prompt to the OpenAI chat model
            ChatLanguageModel chatModel =
OpenAiChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
```

```
            AiMessage aiMessage =
chatModel.generate(prompt.toUserMessage()).content();
            // See an answer from the model
            String answer = aiMessage.text();
            System.out.println(answer); // Charlie is a cheerful carrot
living in VeggieVille...
        }
    }
    private static Path toPath(String fileName) {
        try {
            URL fileUrl =
ChatWithDocumentsExamples.class.getResource(fileName);
            return Paths.get(fileUrl.toURI());
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
    }
}
```

other-examples\src\main\java\DocumentLoaderExamples.java

```
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.FileSystemDocumentLoader;
import java.net.URISyntaxException;
import java.net.URL;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
public class DocumentLoaderExamples {
    static class Load_Pdf_From_File_System_Example {
        public static void main(String[] args) {
            Path filePath = toPath("example-files/story-about-happy-
carrot.pdf");
            Document document =
FileSystemDocumentLoader.loadDocument(filePath);
            System.out.println(document);
        }
    }
    static class Load_Docx_From_File_System_Example {
        public static void main(String[] args) {
            Path filePath = toPath("example-files/story-about-happy-
carrot.docx");
            Document document =
FileSystemDocumentLoader.loadDocument(filePath);
            System.out.println(document);
        }
    }
    static class Load_The_Whole_Directory_Example {
        public static void main(String[] args) {
            Path directoryPath = toPath("example-files");
            List<Document> documents =
FileSystemDocumentLoader.loadDocuments(directoryPath);
            documents.forEach(System.out::println);
        }
    }
    private static Path toPath(String fileName) {
        try {
            URL fileUrl =
ChatWithDocumentsExamples.class.getResource(fileName);
            return Paths.get(fileUrl.toURI());
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
    }
}
```

other-examples\src\main\java\embedding\classification\EmbeddingModelTextClassifierExample.java

```
package embedding.classification;
import dev.langchain4j.classification.EmbeddingModelTextClassifier;
import dev.langchain4j.classification.TextClassifier;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import static embedding.classification.EmbeddingModelTextClassifierExample.CustomerServiceCategory.*;
import static java.util.Arrays.asList;
public class EmbeddingModelTextClassifierExample {
    enum CustomerServiceCategory {
        BILLING_AND_PAYMENTS,
        TECHNICAL_SUPPORT,
        ACCOUNT_MANAGEMENT,
        PRODUCT_INFORMATION,
        ORDER_STATUS,
        RETURNS_AND_EXCHANGES,
        FEEDBACK_AND_COMPLAINTS
    }
    public static void main(String[] args) {
        Map<CustomerServiceCategory, List<String>> examples = new HashMap<>();
        examples.put(BILLING_AND_PAYMENTS, asList(
                "Can I pay using PayPal?",
                "Do you accept Bitcoin?",
                "Is it possible to pay via wire transfer?",
                "I keep getting an error message when I try to pay.",
                "My card was charged twice, can you help?",
                "Why was my payment declined?",
                "How can I request a refund?",
                "When will I get my refund?",
                "Can I get a refund if I cancel my subscription?",
                "Can you send me an invoice for my last order?",
                "I didn't receive a receipt for my purchase.",
                "Is the invoice sent to my email automatically?",
                "How do I upgrade my subscription?",
                "What are the differences between the Basic and Premium plans?",
                "How do I cancel my subscription?",
                "Can I switch to a monthly plan from an annual one?",
                "I want to downgrade my subscription, how do I go about it?",
                "Is there a penalty for downgrading my plan?"
        ));
        examples.put(TECHNICAL_SUPPORT, asList(
                "The app keeps crashing whenever I open it.",
                "I can't save changes in the settings.",
                "Why is the search function not working?",
                "The installer is stuck at 50%.",
                "I keep getting an ‘Installation Failed' message.",
                "How do I install this on a Mac?",
                "I can't connect to the server.",
                "Why am I constantly getting disconnected?",
                "My Wi-Fi works, but your app says no internet connection.",
                "Why is the app so slow?",
                "I'm experiencing lag during video calls.",
                "The website keeps freezing on my browser.",
```

```
            "I get a ‘404 Not Found' error.",
            "What does the ‘Permission Denied' error mean?",
            "Why am I seeing an ‘Insufficient Storage' warning?",
            "Is this compatible with Windows 11?",
            "The app doesn't work on my Android phone.",
            "Do you have a browser extension for Safari?"
    ));
    examples.put(ACCOUNT_MANAGEMENT, asList(
            "I forgot my password, how can I reset it?",
            "I didn't receive a password reset email.",
            "Is there a way to change my password from within the app?",
            "How do I set up two-factor authentication?",
            "I lost my phone, how can I log in now?",
            "Why am I not getting the 2FA code?",
            "My account has been locked, what do I do?",
            "Is there a limit on login attempts?",
            "I've been locked out for no reason, can you help?",
            "How do I change my email address?",
            "Can I update my profile picture?",
            "How do I edit my shipping address?",
            "Can I share my account with family?",
            "How do I give admin access to my team member?",
            "Is there a guest access feature?",
            "How do I delete my account?",
            "What happens to my data if I deactivate my account?",
            "Can I reactivate my account later?"
    ));
    examples.put(PRODUCT_INFORMATION, asList(
            "What does the ‘Sync' feature do?",
            "How does the privacy mode work?",
            "Can you explain the real-time tracking feature?",
            "When will the new model be in stock?",
            "Do you have this item in a size medium?",
            "Are you restocking the sold-out items soon?",
            "What's the difference between version 1.0 and 2.0?",
            "Is the Pro version worth the extra cost?",
            "Do older versions support the new update?",
            "Is this product compatible with iOS?",
            "Will this work with a 220V power supply?",
            "Do you have options for USB-C?",
            "Are there any accessories included?",
            "Do you sell protective cases for this model?",
            "What add-ons would you recommend?",
            "What does the warranty cover?",
            "How do I claim the warranty?",
            "Is the warranty international?"
    ));
    examples.put(ORDER_STATUS, asList(
            "Where is my order right now?",
            "Can you give me a tracking number?",
            "How do I know my order has been shipped?",
            "Can I change the shipping method?",
            "Do you offer overnight shipping?",
            "Is pickup from the store an option?",
            "When will my order arrive?",
            "Why is my delivery delayed?",
            "Can I specify a delivery date?",
            "It's past the delivery date, where is my order?",
            "Will I be notified if there's a delay?",
            "How long will the weather delay my shipment?",
            "I received my order, but an item is missing.",
```

```
                "The package was empty when it arrived.",
                "I got the wrong item, what should I do?",
                "Will all my items arrive at the same time?",
                "Why did I receive only part of my order?",
                "Is it possible to get the remaining items faster?"
        ));
        examples.put(RETURNS_AND_EXCHANGES, asList(
                "What's your return policy?",
                "Is the return shipping free?",
                "Do I need the original packaging to return?",
                "How do I get a return label?",
                "Do I need to call customer service for a return?",
                "Is an RMA number required?",
                "I need to exchange for a different size.",
                "Can I exchange a gift?",
                "How long does the exchange process take?",
                "My item arrived damaged, what do I do?",
                "The product doesn't work as described.",
                "There's a part missing, can you send it?",
                "I received the wrong item, how can I get it corrected?",
                "I didn't order this, why did I receive it?",
                "You sent me two of the same item by mistake.",
                "Is there a restocking fee for returns?",
                "Will I get a full refund?",
                "How much will be deducted for restocking?"
        ));
        examples.put(FEEDBACK_AND_COMPLAINTS, asList(
                "The material quality is not as advertised.",
                "The product broke after a week of use.",
                "The colors faded after the first wash.",
                "The representative was rude to me.",
                "I was on hold for 30 minutes, this is unacceptable.",
                "Your customer service resolved my issue quickly, thank you!",
                "Your website is hard to navigate.",
                "The app keeps crashing, it's frustrating.",
                "The checkout process is confusing.",
                "You should offer a chat feature for quicker help.",
                "Can you add a wishlist feature?",
                "Please make a mobile-friendly version of the website.",
                "I found a bug in your software.",
                "There's a typo on your homepage.",
                "The payment page has a glitch.",
                "Can you start offering this in a gluten-free option?",
                "Please add support for Linux.",
                "I wish you had more colors to choose from."
        ));
        EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
        TextClassifier<CustomerServiceCategory> classifier = new
EmbeddingModelTextClassifier<>(embeddingModel, examples);
        List<CustomerServiceCategory> categories = classifier.classify("Yo
where is my package?");
        System.out.println(categories); // [ORDER_STATUS]
    }
}
```

other-examples\src\main\java\embedding\model\HuggingFaceEmbeddingModelExample.java

```
package embedding.model;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.huggingface.HuggingFaceEmbeddingModel;
import dev.langchain4j.model.output.Response;
import static java.time.Duration.ofSeconds;
public class HuggingFaceEmbeddingModelExample {
    public static void main(String[] args) {
        EmbeddingModel embeddingModel = HuggingFaceEmbeddingModel.builder()
                .accessToken(System.getenv("HF_API_KEY"))
                .modelId("sentence-transformers/all-MiniLM-L6-v2")
                .waitForModel(true)
                .timeout(ofSeconds(60))
                .build();
        Response<Embedding> response = embeddingModel.embed("Hello, how are
you?");
        System.out.println(response);
    }
}
```

other-examples\src\main\java\embedding\model\InProcessEmbeddingModelExamples.java

```
package embedding.model;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.embedding.OnnxEmbeddingModel;
import java.io.IOException;
public class InProcessEmbeddingModelExamples {
    static class Pre_Packaged_In_Process_Embedding_Model_Example {
        public static void main(String[] args) throws IOException {
            String text = "Let's demonstrate that embedding can be done
within a Java process and entirely offline.";
            // requires "langchain4j-embeddings-all-minilm-l6-v2" Maven/
Gradle dependency, see pom.xml
            EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
            Embedding inProcessEmbedding =
embeddingModel.embed(text).content();
            System.out.println(inProcessEmbedding);
            // Uncomment to compare with embedding generated by HuggingFace
            // EmbeddingModel huggingFaceEmbeddingModel =
HuggingFaceEmbeddingModel.builder()
            //         .accessToken(System.getenv("HF_API_KEY"))
            //         .modelId("sentence-transformers/all-MiniLM-L6-v2")
            //         .build();
            //Embedding huggingFaceEmbedding =
huggingFaceEmbeddingModel.embed(text).content();
            //System.out.println(CosineSimilarity.between(inProcessEmbedding,
huggingFaceEmbedding));
            // 1.000000001963221 <- this indicates that the embedding created
by the offline in-process all-MiniLM-L6-v2 model
            // is practically identical to that generated using the
HuggingFace API.
        }
    }
    static class Custom_In_Process_Embedding_Model_Example {
        public static void main(String[] args) throws IOException {
            String text = "Let's demonstrate that embedding can be done
within a Java process and entirely offline.";
            // requires "langchain4j-embeddings" Maven/Gradle dependency, see
pom.xml
            EmbeddingModel embeddingModel = new OnnxEmbeddingModel("/home/me/
model.onnx");
            Embedding inProcessEmbedding =
embeddingModel.embed(text).content();
            System.out.println(inProcessEmbedding);
        }
    }
}
```

other-examples\src\main\java\embedding\model\OpenAiEmbeddingModelExample.java

```
package embedding.model;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.openai.OpenAiEmbeddingModel;
import dev.langchain4j.model.output.Response;
public class OpenAiEmbeddingModelExample {
    public static void main(String[] args) {
        EmbeddingModel embeddingModel =
OpenAiEmbeddingModel.withApiKey("demo");
        Response<Embedding> response = embeddingModel.embed("Hello, how are
you?");
        System.out.println(response);
    }
}
```

other-examples\src\main\java\embedding\model\VertexAiEmbeddingModelExample.java

```
package embedding.model;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.output.Response;
import dev.langchain4j.model.vertexai.VertexAiEmbeddingModel;
public class VertexAiEmbeddingModelExample {
    public static void main(String[] args) {
        EmbeddingModel embeddingModel = VertexAiEmbeddingModel.builder()
                .endpoint("us-central1-aiplatform.googleapis.com:443")
                .project("langchain4j")
                .location("us-central1")
                .publisher("google")
                .modelName("textembedding-gecko@001")
                .build();
        Response<Embedding> response = embeddingModel.embed("Hello, how are
you?");
        System.out.println(response);
    }
}
```

other-examples\src\main\java\embedding\store\InMemoryEmbeddingStoreExample.java

```
package embedding.store;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.inmemory.InMemoryEmbeddingStore;
import java.util.List;
public class InMemoryEmbeddingStoreExample {
    public static void main(String[] args) {
        InMemoryEmbeddingStore<TextSegment> embeddingStore = new
InMemoryEmbeddingStore<>();
        EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
        TextSegment segment1 = TextSegment.from("I like football.");
        Embedding embedding1 = embeddingModel.embed(segment1).content();
        embeddingStore.add(embedding1, segment1);
        TextSegment segment2 = TextSegment.from("The weather is good today.");
        Embedding embedding2 = embeddingModel.embed(segment2).content();
        embeddingStore.add(embedding2, segment2);
        Embedding queryEmbedding = embeddingModel.embed("What is your
favourite sport?").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(queryEmbedding, 1);
        EmbeddingMatch<TextSegment> embeddingMatch = relevant.get(0);
        System.out.println(embeddingMatch.score()); // 0.8144288515898701
        System.out.println(embeddingMatch.embedded().text()); // I like
football.
        // In-memory embedding store can be serialized and deserialized to/
from JSON
        // String serializedStore = embeddingStore.serializeToJson();
        // InMemoryEmbeddingStore<TextSegment> deserializedStore =
InMemoryEmbeddingStore.fromJson(serializedStore);
        // In-memory embedding store can be serialized and deserialized to/
from file
        // String filePath = "/home/me/embedding.store";
        // embeddingStore.serializeToFile(filePath);
        // InMemoryEmbeddingStore<TextSegment> deserializedStore =
InMemoryEmbeddingStore.fromFile(filePath);
    }
}
```

other-examples\src\main\java\HelloWorldExample.java

```
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
public class HelloWorldExample {
    public static void main(String[] args) {
        // Create an instance of a model
        ChatLanguageModel model =
OpenAiChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
        // Start interacting
        String answer = model.generate("Hello world!");
        System.out.println(answer); // Hello! How can I assist you today?
    }
}
```

other-examples\src\main\java\OtherServiceExamples.java

```
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.input.structured.StructuredPrompt;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.output.structured.Description;
import dev.langchain4j.service.*;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.List;
import static java.util.Arrays.asList;
public class OtherServiceExamples {
    static ChatLanguageModel chatLanguageModel =
OpenAiChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
    static class Sentiment_Extracting_AI_Service_Example {
        enum Sentiment {
            POSITIVE, NEUTRAL, NEGATIVE;
        }
        interface SentimentAnalyzer {
            @UserMessage("Analyze sentiment of {{it}}")
            Sentiment analyzeSentimentOf(String text);
            @UserMessage("Does {{it}} have a positive sentiment?")
            boolean isPositive(String text);
        }
        public static void main(String[] args) {
            SentimentAnalyzer sentimentAnalyzer =
AiServices.create(SentimentAnalyzer.class, chatLanguageModel);
            Sentiment sentiment = sentimentAnalyzer.analyzeSentimentOf("It is
good!");
            System.out.println(sentiment); // POSITIVE
            boolean positive = sentimentAnalyzer.isPositive("It is bad!");
            System.out.println(positive); // false
        }
    }
    static class Number_Extracting_AI_Service_Example {
        interface NumberExtractor {
            @UserMessage("Extract number from {{it}}")
            int extractInt(String text);
            @UserMessage("Extract number from {{it}}")
            long extractLong(String text);
            @UserMessage("Extract number from {{it}}")
            BigInteger extractBigInteger(String text);
            @UserMessage("Extract number from {{it}}")
            float extractFloat(String text);
            @UserMessage("Extract number from {{it}}")
            double extractDouble(String text);
            @UserMessage("Extract number from {{it}}")
            BigDecimal extractBigDecimal(String text);
        }
        public static void main(String[] args) {
            NumberExtractor extractor =
AiServices.create(NumberExtractor.class, chatLanguageModel);
            String text = "After countless millennia of computation, the
supercomputer Deep Thought finally announced " +
                    "that the answer to the ultimate question of life, the
universe, and everything was forty two.";
            int intNumber = extractor.extractInt(text);
```

```
            System.out.println(intNumber); // 42
            long longNumber = extractor.extractLong(text);
            System.out.println(longNumber); // 42
            BigInteger bigIntegerNumber = extractor.extractBigInteger(text);
            System.out.println(bigIntegerNumber); // 42
            float floatNumber = extractor.extractFloat(text);
            System.out.println(floatNumber); // 42.0
            double doubleNumber = extractor.extractDouble(text);
            System.out.println(doubleNumber); // 42.0
            BigDecimal bigDecimalNumber = extractor.extractBigDecimal(text);
            System.out.println(bigDecimalNumber); // 42.0
        }
    }
    static class Date_and_Time_Extracting_AI_Service_Example {
        interface DateTimeExtractor {
            @UserMessage("Extract date from {{it}}")
            LocalDate extractDateFrom(String text);
            @UserMessage("Extract time from {{it}}")
            LocalTime extractTimeFrom(String text);
            @UserMessage("Extract date and time from {{it}}")
            LocalDateTime extractDateTimeFrom(String text);
        }
        public static void main(String[] args) {
            DateTimeExtractor extractor =
AiServices.create(DateTimeExtractor.class, chatLanguageModel);
            String text = "The tranquility pervaded the evening of 1968, just
fifteen minutes shy of midnight," +
                    " following the celebrations of Independence Day.";
            LocalDate date = extractor.extractDateFrom(text);
            System.out.println(date); // 1968-07-04
            LocalTime time = extractor.extractTimeFrom(text);
            System.out.println(time); // 23:45
            LocalDateTime dateTime = extractor.extractDateTimeFrom(text);
            System.out.println(dateTime); // 1968-07-04T23:45
        }
    }
    static class POJO_Extracting_AI_Service_Example {
        static class Person {
            private String firstName;
            private String lastName;
            private LocalDate birthDate;
            @Override
            public String toString() {
                return "Person {" +
                        " firstName = \"" + firstName + "\"" +
                        ", lastName = \"" + lastName + "\"" +
                        ", birthDate = " + birthDate +
                        " }";
            }
        }
        interface PersonExtractor {
            @UserMessage("Extract information about a person from {{it}}")
            Person extractPersonFrom(String text);
        }
        public static void main(String[] args) {
            PersonExtractor extractor =
AiServices.create(PersonExtractor.class, chatLanguageModel);
            String text = "In 1968, amidst the fading echoes of Independence
Day, "
                    + "a child named John arrived under the calm evening sky.
"
```

```
                    + "This newborn, bearing the surname Doe, marked the
start of a new journey.";
            Person person = extractor.extractPersonFrom(text);
            System.out.println(person); // Person { firstName = "John",
lastName = "Doe", birthDate = 1968-07-04 }
        }
    }
    static class POJO_With_Descriptions_Extracting_AI_Service_Example {
        static class Recipe {
            @Description("short title, 3 words maximum")
            private String title;
            @Description("short description, 2 sentences maximum")
            private String description;
            @Description("each step should be described in 4 words, steps
should rhyme")
            private List<String> steps;
            private Integer preparationTimeMinutes;
            @Override
            public String toString() {
                return "Recipe {" +
                        " title = \"" + title + "\"" +
                        ", description = \"" + description + "\"" +
                        ", steps = " + steps +
                        ", preparationTimeMinutes = " +
preparationTimeMinutes +
                        " }";
            }
        }
        @StructuredPrompt("Create a recipe of a {{dish}} that can be prepared
using only {{ingredients}}")
        static class CreateRecipePrompt {
            private String dish;
            private List<String> ingredients;
        }
        interface Chef {
            Recipe createRecipeFrom(String... ingredients);
            Recipe createRecipe(CreateRecipePrompt prompt);
        }
        public static void main(String[] args) {
            Chef chef = AiServices.create(Chef.class, chatLanguageModel);
            Recipe recipe = chef.createRecipeFrom("cucumber", "tomato",
"feta", "onion", "olives");
            System.out.println(recipe);
            // Recipe {
            //     title = "Greek Salad",
            //     description = "A refreshing mix of veggies and feta cheese
in a zesty dressing.",
            //     steps = [
            //         "Chop cucumber and tomato",
            //         "Add onion and olives",
            //         "Crumble feta on top",
            //         "Drizzle with dressing and enjoy!"
            //     ],
            //     preparationTimeMinutes = 10
            // }
            CreateRecipePrompt prompt = new CreateRecipePrompt();
            prompt.dish = "salad";
            prompt.ingredients = asList("cucumber", "tomato", "feta",
"onion", "olives");
            Recipe anotherRecipe = chef.createRecipe(prompt);
            System.out.println(anotherRecipe);
```

```
            // Recipe ...
        }
    }
    static class AI_Service_with_System_Message_Example {
        interface Chef {
            @SystemMessage("You are a professional chef. You are friendly,
polite and concise.")
            String answer(String question);
        }
        public static void main(String[] args) {
            Chef chef = AiServices.create(Chef.class, chatLanguageModel);
            String answer = chef.answer("How long should I grill chicken?");
            System.out.println(answer); // Grilling chicken usually takes
around 10-15 minutes per side, depending on ...
        }
    }
    static class AI_Service_with_System_and_User_Messages_Example {
        interface TextUtils {
            @SystemMessage("You are a professional translator into
{{language}}")
            @UserMessage("Translate the following text: {{text}}")
            String translate(@V("text") String text, @V("language") String
language);
            @SystemMessage("Summarize every message from user in {{n}} bullet
points. Provide only bullet points.")
            List<String> summarize(@UserMessage String text, @V("n") int n);
        }
        public static void main(String[] args) {
            TextUtils utils = AiServices.create(TextUtils.class,
chatLanguageModel);
            String translation = utils.translate("Hello, how are you?",
"italian");
            System.out.println(translation); // Ciao, come stai?
            String text = "AI, or artificial intelligence, is a branch of
computer science that aims to create " +
                    "machines that mimic human intelligence. This can range
from simple tasks such as recognizing " +
                    "patterns or speech to more complex tasks like making
decisions or predictions.";
            List<String> bulletPoints = utils.summarize(text, 3);
            System.out.println(bulletPoints);
            // [
            //     "- AI is a branch of computer science",
            //     "- It aims to create machines that mimic human
intelligence",
            //     "- It can perform simple or complex tasks"
            // ]
        }
    }
    static class AI_Service_with_UserName_Example {
        interface Assistant {
            String chat(@UserName String name, @UserMessage String message);
        }
        public static void main(String[] args) {
            Assistant assistant = AiServices.create(Assistant.class,
chatLanguageModel);
            String answer = assistant.chat("Klaus", "Hi, tell me my name if
you see it.");
            System.out.println(answer); // Hello! Your name is Klaus. How can
I assist you today?
        }
```

```
    }
}
```

other-examples\src\main\java\PromptTemplateExamples.java

```
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.PromptTemplate;
import java.util.HashMap;
import java.util.Map;
public class PromptTemplateExamples {
    static class PromptTemplate_with_One_Variable_Example {
        public static void main(String[] args) {
            PromptTemplate promptTemplate = PromptTemplate.from("Say 'hi' in
{{it}}.");
            Prompt prompt = promptTemplate.apply("German");
            System.out.println(prompt.text()); // Say 'hi' in German.
        }
    }
    static class PromptTemplate_With_Multiple_Variables_Example {
        public static void main(String[] args) {
            PromptTemplate promptTemplate = PromptTemplate.from("Say
'{{text}}' in {{language}}.");
            Map<String, Object> variables = new HashMap<>();
            variables.put("text", "hi");
            variables.put("language", "German");
            Prompt prompt = promptTemplate.apply(variables);
            System.out.println(prompt.text()); // Say 'hi' in German.
        }
    }
}
```

other-examples\src\main\java\ProxyExample.java

```
import dev.langchain4j.model.openai.OpenAiChatModel;
import java.net.InetSocketAddress;
import java.net.Proxy;
import static java.net.Proxy.Type.HTTP;
public class ProxyExample {
    public static void main(String[] args) {
        OpenAiChatModel model = OpenAiChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .proxy(new Proxy(HTTP, new InetSocketAddress("39.175.77.7",
30001)))
                .build();
        String answer = model.generate("hello");
        System.out.println(answer);
    }
}
```

other-examples\src\main\java\ServiceWithAutoModerationExample.java

```
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.model.openai.OpenAiModerationModel;
import dev.langchain4j.service.AiServices;
import dev.langchain4j.service.Moderate;
import dev.langchain4j.service.ModerationException;
public class ServiceWithAutoModerationExample {
    interface Chat {
        @Moderate
        String chat(String text);
    }
    public static void main(String[] args) {
        OpenAiModerationModel moderationModel =
OpenAiModerationModel.withApiKey(ApiKeys.OPENAI_API_KEY);
        Chat chat = AiServices.builder(Chat.class)
                .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPENAI_A
PI_KEY))
                .moderationModel(moderationModel)
                .build();
        try {
            chat.chat("I WILL KILL YOU!!!");
        } catch (ModerationException e) {
            System.out.println(e.getMessage());
            // Text "I WILL KILL YOU!!!" violates content policy
        }
    }
}
```

other-examples\src\main\java\ServiceWithDynamicToolsExample.java

```
import dev.langchain4j.code.Judge0JavaScriptExecutionTool;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
import static java.time.Duration.ofSeconds;
public class ServiceWithDynamicToolsExample {
    interface Assistant {
        String chat(String message);
    }
    public static void main(String[] args) {
        Judge0JavaScriptExecutionTool judge0Tool = new
Judge0JavaScriptExecutionTool(ApiKeys.RAPID_API_KEY);
        ChatLanguageModel chatLanguageModel = OpenAiChatModel.builder()
                .apiKey(ApiKeys.OPENAI_API_KEY)
                .temperature(0.0)
                .timeout(ofSeconds(60))
                .build();
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(MessageWindowChatMemory.withMaxMessages(20))
                .tools(judge0Tool)
                .build();
        interact(assistant, "What is the square root of 49506838032859?");
        interact(assistant, "Capitalize every third letter: abcabc");
        interact(assistant, "What is the number of hours between 17:00 on 21
Feb 1988 and 04:00 on 12 Apr 2014?");
    }
    private static void interact(Assistant assistant, String userMessage) {
        System.out.println("[User]: " + userMessage);
        String answer = assistant.chat(userMessage);
        System.out.println("[Assistant]: " + answer);
        System.out.println();
        System.out.println();
    }
}
```

other-examples\src\main\java\ServiceWithMemoryExample.java

```
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
public class ServiceWithMemoryExample {
    interface Assistant {
        String chat(String message);
    }
    public static void main(String[] args) {
        ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(10);
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPENAI_A
PI_KEY))
                .chatMemory(chatMemory)
                .build();
        String answer = assistant.chat("Hello! My name is Klaus.");
        System.out.println(answer); // Hello Klaus! How can I assist you
today?
        String answerWithName = assistant.chat("What is my name?");
        System.out.println(answerWithName); // Your name is Klaus.
    }
}
```

other-examples\src\main\java\ServiceWithMemoryForEachUserExample.java

```
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
import dev.langchain4j.service.MemoryId;
import dev.langchain4j.service.UserMessage;
public class ServiceWithMemoryForEachUserExample {
    interface Assistant {
        String chat(@MemoryId int memoryId, @UserMessage String userMessage);
    }
    public static void main(String[] args) {
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPENAI_A
PI_KEY))
                .chatMemoryProvider(memoryId ->
MessageWindowChatMemory.withMaxMessages(10))
                .build();
        System.out.println(assistant.chat(1, "Hello, my name is Klaus"));
        // Hi Klaus! How can I assist you today?
        System.out.println(assistant.chat(2, "Hello, my name is Francine"));
        // Hello Francine! How can I assist you today?
        System.out.println(assistant.chat(1, "What is my name?"));
        // Your name is Klaus.
        System.out.println(assistant.chat(2, "What is my name?"));
        // Your name is Francine.
    }
}
```

other-examples\src\main\java\ServiceWithPersistentMemoryExample.java

```
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.memory.ChatMemory;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
import dev.langchain4j.store.memory.chat.ChatMemoryStore;
import org.mapdb.DB;
import org.mapdb.DBMaker;
import java.util.List;
import java.util.Map;
import static
dev.langchain4j.data.message.ChatMessageDeserializer.messagesFromJson;
import static
dev.langchain4j.data.message.ChatMessageSerializer.messagesToJson;
import static org.mapdb.Serializer.STRING;
public class ServiceWithPersistentMemoryExample {
    interface Assistant {
        String chat(String message);
    }
    public static void main(String[] args) {
        ChatMemory chatMemory = MessageWindowChatMemory.builder()
                .maxMessages(10)
                .chatMemoryStore(new PersistentChatMemoryStore())
                .build();
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPENAI_A
PI_KEY))
                .chatMemory(chatMemory)
                .build();
        String answer = assistant.chat("Hello! My name is Klaus.");
        System.out.println(answer); // Hello Klaus! How can I assist you
today?
        // Now, comment out the two lines above, uncomment the two lines
below, and run again.
        // String answerWithName = assistant.chat("What is my name?");
        // System.out.println(answerWithName); // Your name is Klaus.
    }
    // You can create your own implementation of ChatMemoryStore and store
chat memory whenever you'd like
    static class PersistentChatMemoryStore implements ChatMemoryStore {
        private final DB db = DBMaker.fileDB("chat-
memory.db").transactionEnable().make();
        private final Map<String, String> map = db.hashMap("messages",
STRING, STRING).createOrOpen();
        @Override
        public List<ChatMessage> getMessages(Object memoryId) {
            String json = map.get((String) memoryId);
            return messagesFromJson(json);
        }
        @Override
        public void updateMessages(Object memoryId, List<ChatMessage>
messages) {
            String json = messagesToJson(messages);
            map.put((String) memoryId, json);
            db.commit();
        }
        @Override
        public void deleteMessages(Object memoryId) {
```

```
            map.remove((String) memoryId);
            db.commit();
        }
    }
}
```

other-examples\src\main\java\ServiceWithPersistentMemoryForEachUserExample.java

```
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.memory.chat.ChatMemoryProvider;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
import dev.langchain4j.service.MemoryId;
import dev.langchain4j.service.UserMessage;
import dev.langchain4j.store.memory.chat.ChatMemoryStore;
import org.mapdb.DB;
import org.mapdb.DBMaker;
import java.util.List;
import java.util.Map;
import static
dev.langchain4j.data.message.ChatMessageDeserializer.messagesFromJson;
import static
dev.langchain4j.data.message.ChatMessageSerializer.messagesToJson;
import static org.mapdb.Serializer.INTEGER;
import static org.mapdb.Serializer.STRING;
public class ServiceWithPersistentMemoryForEachUserExample {
    interface Assistant {
        String chat(@MemoryId int memoryId, @UserMessage String userMessage);
    }
    public static void main(String[] args) {
        PersistentChatMemoryStore store = new PersistentChatMemoryStore();
        ChatMemoryProvider chatMemoryProvider = memoryId ->
MessageWindowChatMemory.builder()
                .id(memoryId)
                .maxMessages(10)
                .chatMemoryStore(store)
                .build();
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPENAI_A
PI_KEY))
                .chatMemoryProvider(chatMemoryProvider)
                .build();
        System.out.println(assistant.chat(1, "Hello, my name is Klaus"));
        System.out.println(assistant.chat(2, "Hi, my name is Francine"));
        // Now, comment out the two lines above, uncomment the two lines
below, and run again.
        // System.out.println(assistant.chat(1, "What is my name?"));
        // System.out.println(assistant.chat(2, "What is my name?"));
    }
    // You can create your own implementation of ChatMemoryStore and store
chat memory whenever you'd like
    static class PersistentChatMemoryStore implements ChatMemoryStore {
        private final DB db = DBMaker.fileDB("multi-user-chat-
memory.db").transactionEnable().make();
        private final Map<Integer, String> map = db.hashMap("messages",
INTEGER, STRING).createOrOpen();
        @Override
        public List<ChatMessage> getMessages(Object memoryId) {
            String json = map.get((int) memoryId);
            return messagesFromJson(json);
        }
        @Override
        public void updateMessages(Object memoryId, List<ChatMessage>
messages) {
```

```
            String json = messagesToJson(messages);
            map.put((int) memoryId, json);
            db.commit();
        }
        @Override
        public void deleteMessages(Object memoryId) {
            map.remove((int) memoryId);
            db.commit();
        }
    }
}
```

other-examples\src\main\java\ServiceWithRetrieverExample.java

```
public class ServiceWithRetrieverExample {
    // Please check CustomerSupportApplication and
CustomerSupportApplicationTest
    // from spring-boot-example module
}
```

other-examples\src\main\java\ServiceWithStreamingExample.java

```
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiStreamingChatModel;
import dev.langchain4j.service.AiServices;
import dev.langchain4j.service.TokenStream;
public class ServiceWithStreamingExample {
    interface Assistant {
        TokenStream chat(String message);
    }
    public static void main(String[] args) {
        // Sorry, "demo" API key does not support streaming (yet). Please use
your own key.
        StreamingChatLanguageModel model =
OpenAiStreamingChatModel.withApiKey(System.getenv("OPENAI_API_KEY"));
        Assistant assistant = AiServices.create(Assistant.class, model);
        TokenStream tokenStream = assistant.chat("Tell me a joke");
        tokenStream.onNext(System.out::println)
                .onComplete(System.out::println)
                .onError(Throwable::printStackTrace)
                .start();
    }
}
```

other-examples\src\main\java\ServiceWithToolsExample.java

```
import dev.langchain4j.agent.tool.Tool;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
public class ServiceWithToolsExample {
    // Please also check CustomerSupportApplication and
CustomerSupportApplicationTest
    // from spring-boot-example module
    static class Calculator {
        @Tool("Calculates the length of a string")
        int stringLength(String s) {
            return s.length();
        }
        @Tool("Calculates the sum of two numbers")
        int add(int a, int b) {
            return a + b;
        }
        @Tool("Calculates the square root of a number")
        double sqrt(int x) {
            return Math.sqrt(x);
        }
    }
    interface Assistant {
        String chat(String userMessage);
    }
    public static void main(String[] args) {
        Assistant assistant = AiServices.builder(Assistant.class)
                .chatLanguageModel(OpenAiChatModel.withApiKey(ApiKeys.OPENAI_A
PI_KEY))
                .tools(new Calculator())
                .chatMemory(MessageWindowChatMemory.withMaxMessages(10))
                .build();
        String question = "What is the square root of the sum of the numbers
of letters in the words \"hello\" and \"world\"?";
        String answer = assistant.chat(question);
        System.out.println(answer);
        // The square root of the sum of the number of letters in the words
"hello" and "world" is approximately 3.162.
    }
}
```

other-examples\src\main\java\SimpleServiceExample.java

```
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.openai.OpenAiChatModel;
import dev.langchain4j.service.AiServices;
public class SimpleServiceExample {
    interface Assistant {
        String chat(String message);
    }
    public static void main(String[] args) {
        ChatLanguageModel chatLanguageModel =
OpenAiChatModel.withApiKey(ApiKeys.OPENAI_API_KEY);
        Assistant assistant = AiServices.create(Assistant.class,
chatLanguageModel);
        String answer = assistant.chat("Hello");
        System.out.println(answer); // Hello! How can I assist you today?
    }
}
```

other-examples\src\main\java\StreamingExamples.java

```
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.data.message.ChatMessage;
import dev.langchain4j.model.StreamingResponseHandler;
import dev.langchain4j.model.chat.StreamingChatLanguageModel;
import dev.langchain4j.model.language.StreamingLanguageModel;
import dev.langchain4j.model.openai.OpenAiStreamingChatModel;
import dev.langchain4j.model.openai.OpenAiStreamingLanguageModel;
import dev.langchain4j.model.output.Response;
import java.util.List;
import static dev.langchain4j.data.message.SystemMessage.systemMessage;
import static dev.langchain4j.data.message.UserMessage.userMessage;
import static java.util.Arrays.asList;
public class StreamingExamples {
    static class StreamingChatLanguageModel_Example {
        public static void main(String[] args) {
            // Sorry, "demo" API key does not support streaming (yet). Please
use your own key.
            StreamingChatLanguageModel model =
OpenAiStreamingChatModel.withApiKey(System.getenv("OPENAI_API_KEY"));
            List<ChatMessage> messages = asList(
                    systemMessage("You are a very sarcastic assistant"),
                    userMessage("Tell me a joke")
            );
            model.generate(messages, new
StreamingResponseHandler<AiMessage>() {
                @Override
                public void onNext(String token) {
                    System.out.println("New token: '" + token + "'");
                }
                @Override
                public void onComplete(Response<AiMessage> response) {
                    System.out.println("Streaming completed: " + response);
                }
                @Override
                public void onError(Throwable error) {
                    error.printStackTrace();
                }
            });
        }
    }
    static class StreamingLanguageModel_Example {
        public static void main(String[] args) {
            // Sorry, "demo" API key does not support streaming (yet). Please
use your own key.
            StreamingLanguageModel model =
OpenAiStreamingLanguageModel.withApiKey(System.getenv("OPENAI_API_KEY"));
            model.generate("Tell me a joke", new
StreamingResponseHandler<String>() {
                @Override
                public void onNext(String token) {
                    System.out.println("New token: '" + token + "'");
                }
                @Override
                public void onComplete(Response<String> response) {
                    System.out.println("Streaming completed: " + response);
                }
                @Override
                public void onError(Throwable error) {
```

```
                    error.printStackTrace();
                }
            });
        }
    }
}
```

other-examples\src\main\java\StructuredPromptTemplateExamples.java

```
import dev.langchain4j.data.message.AiMessage;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.input.Prompt;
import dev.langchain4j.model.input.structured.StructuredPrompt;
import dev.langchain4j.model.input.structured.StructuredPromptProcessor;
import dev.langchain4j.model.openai.OpenAiChatModel;
import java.util.List;
import static java.time.Duration.ofSeconds;
import static java.util.Arrays.asList;
public class StructuredPromptTemplateExamples {
    static ChatLanguageModel model = OpenAiChatModel.builder()
            .apiKey(ApiKeys.OPENAI_API_KEY)
            .timeout(ofSeconds(60))
            .build();
    static class Simple_Structured_Prompt_Example {
        @StructuredPrompt("Create a recipe of a {{dish}} that can be prepared
using only {{ingredients}}")
        static class CreateRecipePrompt {
            private String dish;
            private List<String> ingredients;
        }
        public static void main(String[] args) {
            CreateRecipePrompt createRecipePrompt = new CreateRecipePrompt();
            createRecipePrompt.dish = "salad";
            createRecipePrompt.ingredients = asList("cucumber", "tomato",
"feta", "onion", "olives");
            Prompt prompt =
StructuredPromptProcessor.toPrompt(createRecipePrompt);
            AiMessage aiMessage =
model.generate(prompt.toUserMessage()).content();
            System.out.println(aiMessage.text());
        }
    }
    static class Multi_Line_Structured_Prompt_Example {
        @StructuredPrompt({
                "Create a recipe of a {{dish}} that can be prepared using
only {{ingredients}}.",
                "Structure your answer in the following way:",
                "Recipe name: ...",
                "Description: ...",
                "Preparation time: ...",
                "Required ingredients:",
                "- ...",
                "- ...",
                "Instructions:",
                "- ...",
                "- ..."
        })
        static class CreateRecipePrompt {
            private String dish;
            private List<String> ingredients;
        }
        public static void main(String[] args) {
            CreateRecipePrompt createRecipePrompt = new CreateRecipePrompt();
            createRecipePrompt.dish = "salad";
            createRecipePrompt.ingredients = asList("cucumber", "tomato",
"feta", "onion", "olives");
            Prompt prompt =
```

```
StructuredPromptProcessor.toPrompt(createRecipePrompt);
            AiMessage aiMessage =
model.generate(prompt.toUserMessage()).content();
            System.out.println(aiMessage.text());
        }
    }
}
```

other-examples\src\main\resources\example-files\story-about-happy-carrot.docx

BASE64:

UEsDBAoAAAAAAIdO4kAAAAAAAAAAAAAAAAAJAAAAZG9jUHJvcHMvUEsDBBQAAAAIAIdO4kCXfTO6Zg
EAAHkCAAAQAAAAZG9jUHJvcHMvYXBwLnhtbJ1RQU7DMBC8I/GHKPfETtq0BW1dlUBPCCo10GNlOdvG
IrEt21T09zgUteHKbWdGO5rdgcVX10ZHtE5qNY+zlMYRKqFrqQ7z+K1aJbM4cp6rmrda4Tw+oYsX7P
YG1lYbtF6ii4KFcvO48d7cE+JEgx13aZBVUPbadtwHaA9E7/dS4KMWnx0qT3JKJwS/PKoa68RcDOOz
4/3R/9e01qLP596rkwmBGVTYmZZ7ZC99nDatte+AXFhY8wM6lgM5D7DVtnZsTGdAziOUDbdc+PAplt
PZBMiAgGepwjoFch6CneUHy03zQw4QVNrztpIdsmBxBbARvMUyZGZ73joEciV69w/3Zir92F/
wq/8lB/G20jcbw0V/z3gaTroGHUiwNKaVgvvQO9uuN9HrTze7LEvzlKZZVoymu1X2NMqnD2WST+7KZ
Dwq6mSZFXlCi7IIv6E0L5dAhk4Qyt2g+LTSn/p/DGF47qVi9g1QSwMEFAAAAAgAh07iQHQvpfZOAQA
AjwIAABEAAABkb2NQcm9wcy9jb3JlLnhtbI2Sy27CMBBF95X6D5H3ifBCFlJkFrECqRKpWrVnWsPY
BE/ZBtC/r5JgDRVWXQ5uneO73gmn59lFZzAOqFVgZIoRgEoprQuwK9bZbhDAXOU8VppRUUqAGH5uX
jQ84MYdrCi9UGrBfggpakHGGmQHvvDcHYsT1I6qLWoVpxq62kvi3tDhvKDnQHOI3jKZbgKaee4g4Ym
oGIrkjOBqQ52qoHcIahAgnKO5xECf7xerDS3W3olZFTCt+YdqZr3DGbs4s4uM9ODMa6rqM662O0+RP
8sV699qOGQnV/xQCVOWf9c4RZoB540ALI5bmb8p49LzZLVKZxmoXxLEziTZKSbEri+DPHN9e1vwNeW
NqWC9l4q4OVOH5R6w6N6OyD2q2mos6v2y1uBfCn5l7DX9MQWV4b/515kpLJZJT5Bii7KBZOoruuMsn
xuOyr3ydUfgNQSwMEFAAAAAgAh07iQIRkhqMlAQAADgIAABMAAABkb2NQcm9wcy9jdXN0b20ueG1sp
ZFBS8MwFMfvgt8h5J4maVa7jrZjbTYQDwrO3UuaboUmKUk6HeJ3N0OnePCi7/b4h9/7vZd8+aIGcJT
W9UYXkEYEAqmFaXu9L+DTdoPmEDjf6LYZjJYFPEkHl+X1Vf5gzSit76UDAaFdAQ/ejwuMnThI1bgox
DoknbGq8aG1e2y6rheSGzEpqT2OCbnBYnLeKDR+4eAHb3H0f0W2Rpzt3G57GoNumX/
CT6BTvm8L+MqTmvOEJCheZzWihFYoY1mKyJyQuIrrTbZav0Ewnh/HEOhGhdXvHu8Dtp2Er6Z+aHfSB
vTRL4bx2XlbUsIYojSKIxJRmrA0x99hji8O/7RhF5vbevdjfFUnCeUrltKaz0JVM0qybEXZJmWc8+o
3G3w+1cdHlu9QSwMECgAAAAAh07iQAAAAAAAAAAAAAAAAAUAAAB3b3JkL1BLAwQUAAAACACHTuJA/
ScuwJIJAAAoZQAADwAAAHdvcmQvc3R5bGVzLnhtbLVdW3ejOBJ+33P2P3Pi7ntOdz40tntOdy6b
PJDmZcbLzLINsaxsQKyejOBJ+33P2P3Pi7kL1BLAwQUAAAACACHTuJA/
vcSR98xVJmQy948Pj3yPJ4EMRbKe+0+P1we/+l6WsyRkkUz43H/lmf/
bl7//7fN2luWvEc88AEiyWRzM/U2ep7PJJAs2PGbZoUx5AgdXUsUsh49qPYmZ+lGkB4GMU5aLpYhE/
jo5OTqa+hWMnPuFSmYVxEEsAiUzucr1JTO5WomAV3/qK9Q+vOWVlzIoYp7khnGieAQ2yCTbiDSr0WI
sGtzipgZ5tt3EcxzV5233IdtKFaZKBjzLoE3iqDQ+ZiJpYI7PdoAaxx2C4ybl7U80FFx+fGT+a9lxf
GSzuHK7vrqmzKIdxo7WLlvxViwVU2UzQwC07E6ziyLLZXzJctbgbbfbw22aHQZJZXar1Y5PJ3Do7SL
fi4PZ93UiFVtGEJzb4zP/C0RmKINLvmJFlGf6o3pQ1cfqk/lzLZM887YzlgVCzP1HEUMw3/Ot96eMG
Th3O9t8TbLuI5xl+ddMgNULES8Kc3aQ7YJMvnyeGP76b8uOtLGqPOuD0RCcEKqLMse2s0AWST73T6a
QoHCDfPXHtcmruV9/8ZRsRMj/2vDkKeMh5HJ14oLH4kaEIdf5XX339P1BCakg++b+p0/
Vl7cy+MHDRQ7EGlU7KcrCq5eApzpPgPa/NafBKT4QGkMK8YZsvsha9OaLhMWAf6+tj3y4dUqWDWday
bzjfYg+Wl4aWkOcDIc4HQ5xNhzil+EQ0+EQ58Mhfh0O8akTYlBgiyTkLz0BNwJwdxiOANwdnCMAd4f
sCMDdgTwCcHd4jwDcHfQjAHenwgjABAmSy4AiPTQsQXJoWILU0LAEiaFhCdJCwxIkhYYlSAkNS5AQG
pYgHcqOkPcdHhtJPv7jaCVlnsicezl/IYBnCYCbYRQRge5+cUXjFwrc8vFfdRo7zf7Qie7udQbM9Lk
7AQbJeq5HSp5ceSuxLhQM5rt64IMYePLMIxgNeiwMgYCSQfEcqgXj30KTNIqvuIJ6CB+fo5U5hCyRS
LiXFPGSItZTtqYD50lodIvQOTUFjTQ2mcaKfKPH2oIi22IGFbLxozOXzLOJ2CCFuBUZwaNIo3rfiij
iVOD3RHlkLCfouxpcgs6rwT0bP+gMLkH31eCWkUExRGjDU3m7sp7K6RU8le/LxCHzfQVP5fsKnsr3F
Xy37z90FodUXB9FHhH0Yy4iqV9MjK8GC7FOGPTvhthclnCrorv3wBRbK5ZuPF37H9/
ibzJ89R5JxlkNNNlA0ajYBfhFJMUQl/fU+9/BkylBbT9Fyeb9HVCpQXMH3XowqN91B6Mx3WW/6R9Tj

6g3i2KZ00jOgkVFWXQYP4fhXSRB9L+l77VQ0E2lqvZ081Ck272uKelAItH+t/sg6Ba/
gRP0GN7Ayzam8P0OB8V9RPA2mOhhdvOacgWViR/jp++1jCK55aGdYkyVy5Xs6f2MyHIVpxuWiWx8h1
1Wc2K8O5aOj/4QwXwPoii6OoDJJJE3Rjexp89Sleb/8Rdf/
nN839w83t16X6F6k7zGVOhUFVhj+4WgeFSW0DIkeAobaBgMiASKcZKgbmwIfuevS8lgrtTolXUD/
wAFUTPzJ+dUFAsWpxRDRWP/I4j/FqqhFAVpQ/BvpoR+yTHA/+0pSt7jQLAecWnV/7Ni+R8eEAxIjem
gkLo5Kd65v8Mn6Km9wyfo5JT4FxGDuZUkkxLeE5B5qL4DchcRjHorF8lIqlUR0UXpRc1A1wo1A10zy
KiIk4zUSYaA0keGgNxFlJFq7oCgPlOmwr+UCOla2KCTNa9BJ2tbg07WsAadtlUJ5lC1YoZgKlULnWB
GVYlu6qrdE7QHVTtb6GTxbmwni3eDThbvBp0s3g06WbwbdLJ4N+hk8X566fHVCvr7hM/xFgdZ7Lc4y
DJAvw3hcQrre9TrgPFczxCsVIiriK8ZxevAEv5ByZVeuyWTnuUuI4icftVCOsIr8clCCQpsdD0eDU5
qOUH0f2NQxoYlVzQvk8vA1P0dqpx6hBWgyAJMucCtJ2NvxXqTe4vNgNdOU7N2zYqv9R/
nmemJXktnBce7fXpqAb/joSji2jXYbJqe7U+BzKnpLz+nMA9gZJdwCmuJe5ug8pLBx9p//
nN8M5TA2g8Lrn9mv8HH2m+We/aEaOUfg4+UtXPbqs9LWP/
t4dPr3Ja7TYFnkDyc2zK4oRhwC7YkbvAHiITN/e/
kE949BbDkBC0VtrYoiUyaDWSxNUfJYoJ1IIutUT4q61C/uUjsUK69tXYo0d6iO5Rob/UdSrS3DA8l2
luPBxLtJ8xDSWyq0MhbpdBDuWza0HCNIELnNnloiMbQIWfxxnYCbM20K95YFlsD7Yo3lsXWOn3ijeX
CiDeWy1m8sUTO4o0lchZvLJGzeGOJnMUbSeQm3lgSmyo0OvdBvLFcNm1ouNrijSWyyUND1BZvJNH+h
Yv6uYccYe1ZwRjKYmugXfHG3outdfrEG8uFEW8sl7N4Y4mcxRtL5CzeWCJn8cYSOYs3kshNvLEkGPH
Gctm0odHUtnhjiWzy0BC1xRtJ5C7eyPeyjuKNZbE10K54Y1lsrdMn3lgujHhjuZzFG0vkLN5YImfxx
hI5izeWyFm8kURu4o0lwYg3lsumDY2mtsUbS2STh4aoLd5IInfxRk57cRRvLIutgXbFG8tia50+8cZ
yYcQby+Us3lgiZ/HGEjmLN5bIWbyxRM7ijSRyE28sCUa8sVw2bWg0tS3eWCKbPDREbfFGErmLN3JWo
aN4Y1lsDbQr3lgWW+v0iTeWCyPeWC5n8cYSOYs3lshZvLFEzuKNJXIWbySRm3hjSTDijeWyaUOjqW3
xxhLZ5KEhaou3IYKt6tu7zuut2c1vPMB8pBxWbc79tN5AR09Rgj3o9Q771aby5sTvZtt5fZ1eQQnnP
DP4oYD2Vu/VmnizNPZtq/
j6zKNy1h3s168x1Lsd+s2Uwrkfi0SqG70xv7bB7NDfeaTeob998KratV9fGWS5nqNYAX4Toahm/
DHYLr4ynCcHTwt9cg029/+3Obi4118t4Yq5z9TB4qu+sLXLv/HEru+CDTgv0Mte4eoe352YTffbvrN
tXfQ2h8vmTrCsx6LcLGztt+Z0x5pqQbyZP1q6az8bIH6WUdmo8M8Fj6I7Zpo4lynww+9gmBffZZCFL
6xqCr7Ky6PHR0bcPhxfyhx+PKL/emUmixr4LgBwTNuY8qM28s1j9X/Zl/8DUEsDBBQAAAAIAIdO4kA
rYcZzFgQAAAAKAAARAAAAd29yZC9zZXR0aW5ncy54bWylVltT2zgUft+/Q8ZvwfHEGjXQ+hAIIUuU
AbD9lmWT2ItungkOSb8+j2SrDg7UNppnyyfy3fuRzr+9Cz4aA3aMCVnSbY3SUYgqaqYXM2Sx4fF+GM
yMpbIinAlYZZswCSfTv7847jLDViLYmaEENLkgs6S2tomT1NDaxDE7KkGJDKXSgti8VevUkH0U9uMq
RINsaxknNlNuj+ZHCU9jJolrZZ5DzEWjGpl1NI6lVwtl4xC/4ka+mfsBs1zRVsB0nqLqQaOPihpata
YiCZ+FQ1DrCPI+r0g1oJHuS6bvCfZh9spXW01fsY9p9BoRcEYLJDgIVxBmNzCZNNXQNtU72Gq02A7d
VConk38afDc8Ff6b1Q7VPGalZroUGZsgB0vGjNvjVXinFiyxeu6bq9rzB6VvRM7VcsOUmQNSslI0Px
qJZUmJcf27LJpcoK9+aKUGHV5A5piubGxJ5MkdQwQJVTFxlgQCyWt8cQSg8QROFe3yhat1qqV1SUQp
CHGmmCovfZbggul7CvBqm+0O41M6poMkUDiJFBwDbhFrGBJWm4fSFlY1URzH/Z7g5UmHZbws2bVP6A
to4QXDaFIiqLZ4VGIrGKm4WRzqTR7wcgIPx90L3DCN1uN/8lH2B9IS3XXSmpbPzB/g5a7LgRAWhNNK
Oai93COXmjFo9nKZXeOc6+xLXsNvwVcWQwGBQulH699QbzsNZA1nBH6ZDgx9anbP57Z8gdNmM9DIHj
pi+cGt1RRs6W9B4srxMuS6l9ssGsm4RLYqrZXEjPNd9S+adI8wLP9xmztIxzsPxq4IMaeGkbkmQbyd
N9yCA3jLSL/SlZYzFNz22JbaUxJX0cPEnZcEbYkZkESgR0aqP3mu1EVJMhqNdu2f1ye3x1Hp+CbEnv
9HUMKa45tAz7gwm445lfagr3Aqay+YFYY7lJfzt/w4D0HQLpUf8U74GHTwAIIVgVvj18P9z1jviILz
pobhvOrQ2F+11ja5aFcrkXxiqxMPNzj0McyfNhfHB5+/CsL7jmxtziI1SOI3F0Vd/
rkOJxcWUYilHRORKkZGd24ywSLK/JSP50xGfkl4AqBXU7RlpE5HgeGEYTzBc5iZPhlInK3Ic5h6WH5
DdGrAbeX0G9ScUd92WK5fQr6M+7IJljrcIJCuqO5bDrt8ZjE2RORbtqyiFoSL4QdFi7cr2vtANMhPV
1u8R3h2/aaDBsP5PixcG0E/XTOkpd6PL912lgwrgv3/

IAb0jRhSZWrbJZwN/6ZU7P4V+EzxP+Uq/2et+95+Od4/odQFyxK9wcnEI4o1R8G2kGkHQw0vGSD3HS
gHUba4UA7ijR8BnV5jeOiOZNPuBPi0dGXinPVQXUZibPkFSkkwU/DlaS8rQAbBG8jcyULi48wl+Hh6
XbyH1BLAwQKAAAAAACHTuJAAAAAAAAAAAAAAAAACwAAAHdvcmQvdGhlbWUvUEsDBBQAAAAIAIdO4kD
Rru/E+AUAACQZAAAVAAAAd29yZC90aGVtZS90aGVtZTEueG1s7VlNbxs3EL0X6EL0Y7L2xZOsjMiIHt
j7i1nYSREqKHCkttcuIu1yQlB3diuRYoEDRtMihAYpeeijaBkiAFmj6a5ymSFMgf6FD7mpFSlTtGD4
YReyLxH0zfJwZviFXV67ej6l3iLkgLGn65Usl38PJkAUkCZv+7X73o8u+JyRKAkRZgpv+FAv/6taHH
1xBmzLCMfbAPhGbqOlHUqaba2tiCMNIXGIpTuDZiPEYSfjKw7WAoyPwG9O19VKpthYjkvhegmJwe2M
0IkPsvfzt99ffP/
a3Zt47FKZIpFADQ8p7yje2TDQ2GJcVQkxFi3LvENGmDxMF7KiP70vfo0hIeND0S/
rPX9u6soY2cyMqV9gadl39l9vlBsF4Xc/Jw0ExaaVSrdS2C/8aQOUyrlPv1Dq1wp8GoOEQVppxMX1W
dxo77WqONUDZR4fvdr29Ubbwhv+NJc7bVfVv4TUo819Zwne7LYiihdegDF9dwlcq9fVWxcJrUIavLe
Hrpe12pW7hNSiiJBkvoUvV2kZrttoCMmJ01wlvVCvd+nrufI6CaiiqS00xYolcVWsxusd4FwAKSJEk
iSenKR6hIZRxC1Ey4MTbJ2Ek1TRoEyPjeTY0FEtDakZPDDlJZdP/
JEWwMeZe37746e2LZ97xg+fHD349fvjw+MEvmSPLahcloWn15ocv/3nymff3s+/ePPrajRcm/
s+fP3/5x1duIGyiOZ1X3zz96/nTV4+/
eP3jIwd8m6OBCe+TGAvvOj7ybrEYFqajYjPHA/5uFv0IEdNiOwkFSpCaxeG/IyMLfX2KKHLgdrAdwT
scRMQFvDa5ZxHuRXwiicPjXhRbwAPG6A7jzijsqbmMMPcnSeienE9M3C2EDl1zt1Bi5bczSUE9ictl
K8IWzZsUJRKFOMHSU8/YGGPH6u4SYsX1gAw5E2wkvbvE20HEGZI+GVjVNDfaJTHkZeoiCPm2YnNwx9
th1LXqNj60kbArEHWQ72NqhfEamkgUu1z2UUzNgO8jGblI9qZ8aOI6QkKmQ0yZ1wmwEC6bGxzWayR9
DwTEnfYDOo1tJJdk7PK5jxgzkW02bkUoTl3YHkkiE/
uxGEOJIu8mky74AbN3iPoOeUDJynTfIdhK98lqcBu006Q0LxD1ZMIdubyGmVW/vSkdIaylBqTdUuyY
JCfKdzbD+Qk3SOWrb584eF9Uyd7mxLlndheEehVuUZ5bjAfk4qtzG02Smxg2xHKLei/
O78XZ/9+L86r9fP6SPFdhEGh1GMyO2/rwHa88e48IpT05pXhf6OO3gN4TdGFQ2emLJy7uYmkEH9VOh
gksXMiRtvE4k58SGfUilMLRvewrJ6HIXYfCS5mAK6MedvpWeDqJD1iQXTnLZXW9zMRDIDkfL1WLcbg
uyAxdq8+vUYV7zTbU190ZAWX7LiSMyWwSGw4S9dmgCpK+XEPQHCT0ys6FRcPB4rJyP0vVEgugVmQFD
kceHKmafrUCJmAEdyZEcaDylKV6ll2dzPPM9KpgWhVQghcbeQXMM91QXFcuT60uK7VTZNoiYZSbTUJ
HRvcwEaEA59WpRk9D411z3Zin1KKnQpHHwqBRv/xfLM6aa7Bb1AaamEpBE++o6dc2qlAyQ5Q2/RFc3
eFjnELtCHWoRTSEF2BDybMNfxZlSbmQbSSiLOBadDI1iInE3KMkbvpq+UUaaKI1RHMrr4MgXFhyDZC
Vi0YOkm4nGY9GeCjNtBsjKtLZV1D4TCucT7X52cHKkk0g3b0oOPIGdMJvISixar2sAhgQAe93ylk0A
wKvJAshm9ffQmPKZdd8J6hrKBtHNI1Q3lFMMc/gWsoLOvpbEQPjW75mCKgRkrwRDkLVYM2gWt206Bo
Zh5Vd92QjFTlDNOc901IV1TXdKmbNMGsDC7E8W5M3WM1CDO3S7PCZdC9KbmOmdQvnhKJLQMCL+Dm67
ikagkFtPplFTTFelmGl2fmo3TtmCzyB2mmahKH6tZnbhbgVPcI5HQyeqfOD3WLVwtBodq7UkdY/
Xpg/L7DBPRCPNrzInVApMoHQoK1/AVBLAwQUAAAACACHTuJANlRuMaoIAACrMwAAEQAAAHdvcmQvZG
9jdW1lbnQueG1s7Vtdc9s2Fn3e2bsRbbsJE2zapVOA42zSPHSaGTvtYwckQRIRCGABUCz31+
+5ACjJ60ymTceeWqMXSyJBEOS9OOfcD3/3/e+9YlvhvDR6XVyeXxRM6MrUUrfr4sPNm7MXBfOB65or
o8W6mIQvvn/5z398N65qUw290IFhCu1Xo63WRReCXS2XvupEz/15LytnvGnCeWX6pWkaWYnlaFy9fH
JxeRG/WWcq4T3ud8X1lvsiT9ffnc1YoXGvxrieB39uXLvsudsM9gyzWx5kKZUME+a+eD5PY9bF4PQq
L+hstyC6ZJUWlD/mK9ydp/jEfdOVr/MbiHdcOqGwBqN9J+3+Mb50NjxiNy9p+7mH2PZqHjfay2d37r
d75D9ig9eOjzDFfsI7033iZdTpol6l90D23Vv1/2f8IxPenmGet+dS7xb2ZQ968KouLz73UrNn0EL2
t/z6zto/+26fRP8+uKXFlvorG+StM4PdLcfKvzbbO73ZzUU7+0+s7OL5nUfzf2qCO3v/uuNW7JZj/
dXgg+lf88B3847jeD5af17pDCQHu+/
y6RKn9hcVrK9W71ptHC8Vnm28fMbGy68ZbZDiJbCrNPVEnzb+ee/
ow+WPN0YHz8ZVJ3VYF7Vo+KBCgQPcVxIv/YorWTpJR7oftL99pMKLmAcsaVb/
X4zbcrUunrwo8pErmv7WMcV1Ox8T+uzDNQ1d5jXh0+a1/a3XOa7Cy591JdhgjWacBdkLJjULnWDBjJ

```
qZhv0i2laKX6RSYkEnnGBKbkWN4eAM4ZpBsYo7ZwLTvMfxq447JcU5vY2Q3kn8myx3Mp+cnWp2ny92
M3q/+W2zkXtYxPFachB8MsiCcTXyybNS8B4kwUYZOvbRTAzygFnjZZBb8O/JVHsIgFHuAxHIVD9Kz7
YAIjIQcE63gvkNdhsZQw2+Y60TApvPWDbSNuPMy7YLOAADdkbVC1YOgckQjd1hNqkbUUHBDJ4pPmCs
cHG2kbueWWhEowF9YcK+5YE5wZWamBeBdbJn3HIXbpueYNZsSKJdB5zEu5D1uoDCHFe0t9fFb2/
NK15tkgfPY/+N9c8jEwg+vq1OiB2h6YRP94NPHSfGqMEczgt4OoQRsUvjpNC1XzDBqw4DtqIVgSRAw
irwjXSMmGjQ8j+DAOVwxyv4ufRBVv62+56MeA9ahpDrJtI+UcwroRR0AvRBSZBFpFIKEZwxYcc2wJl
6SvYDI8m6hjWNY2E037Ib2fcER9AXspc1sA1hk8GlrEVgioHeQE9EnuKsEwRC8JIW4PdtRLb3wlXpe
gsE3XhCPGtCnGPsDFzro9kIUGEmPvgZQSigtJeYHVAq4IETIuOvPGs4AtmTCz0I+b0W3soAC8QdXcu
mAcVBemLr4xBYCRs7iUpFViyNBt2lwRkjEKQzeKAzJUIN+pbOah4GxxV8xNQaeYkjsOcJx+4Px0AnF
L4gniOJ1VBYA/VCurgWiuQWxTO83gKNBkfe2TgDLaX4RFjXyVqcQa+deSE2DKK6hivCD5kSvJnwN4Q
B4RRgxo9AOrokh1O1GCdWO2MxhlwXeswaQzdH5EUzVLwsOSQhZsLyjsCLEQv/7YPfm4hGDd8aR+AEM
1cxyiXWWESZTabxg4aKZpVxGnQDPqKDe6WCRGZFoyNJ4hS4DyRWs5L7zWx+UuTwsXwtJlwkxEOqFrc
mP6MooIPpE+vBl0jQH4MfnNDsntDsZy1YzScST+WgIZ/hXVb4zQQsgS6ywDbuKEjc8ho4c5DIOaHLg
2geoMseJjLjwDBONlJA3TSQt8QQh5AhmJYlwh9SxkybQAr7SFTNY+CDV8iv5IwaAJ1UAaVZBj9AYAK
0NShc+pQ4M4DwGNRqBlHAT1vqQbbUjyBpiCiECNghRLWI+Si8axEN1rSj8Avlxqyo9iiYrYlL2MdBV
mDckJVWEnh0wgwB+TgHeZZpmlLgJ7s+iF3zpkNQbixlBnwf1VeFyg5vKcGZBRJ2Yw4HkTFqhUM+86O
RGt5AOtvAOQTSmjHMTN5wBPY7Cah7ElC/dpRux86/pZdQy6L6Vs5LxDARQRv3SNtT9nxwFslHUR+BY
z0GRr4hkBcaRauE+BnjCe2jabxhsseu98QIkbHJoLvgzH9FDIHM00EdBAMmYhGgDK6ByCLgQApzoLT
zgUpmvEUDw8nOD0IA7zTqU0i5QFVF+3AqhkUV3AllkStRSiLVl5FdGRgfnED84CSqYlFHGwiB3lCBm
qpp5CJzue3AqkdgzxMh3BMh7FUISfoo7z1SMlFqplCbCrDAl27oS5ABD8n78gg/VNQZF+EkJpYC6u7
oSEKmkMRpKnUhjX0ELvhoqKMSSqDkHpKJkP3dygopt119KgUOVApHIuUwHTwX1BcwPOoUSX/2iN4n1
qIWfgyZucdhRGTKNSXmF6gB18i/Er8H9DxuQAgttboSEUTO6JHnAgVYFIhpD+YaYGyGZSVUBJWYkVI
RjogDnU4jZeF7fgz9LydKuCdKeEPlH/gMeuGSNIkZ+2lOEaAL0lK+Nfa/
EcpTvmiOUD10KdoeGp67ckAqlKyFFkWyYdf7AEIg10abTsKYU1/
WA3ZqHrTQLVJzwyL1KCRbpHYDUplSD0mOUvWQHAJUgqCEqoWxnjgXq2faYKLqkJig7G7sQTgMLE78/
yAhxQ/Yjh69JTDWbs/R98/vWeGoOpy6KQUquIlyiIQMuCIgpRgnrDo00wn08OXM4py2wu2y6EsehNo
MSofY7RWwILVb0okpZvYZwALb3UtEnkfgFDMJeTQlpvZn215TN/WIf1S5/Bf9jwc6sPH9+YunubXat
j8BHbELjcXxZ89ir6EjusfPFxfxZ2kC2oT2p5VoDs6iSQgmWhffPImDG3QiHfxsB7IY9TDGTu7KKOr
k9pAI6GbMl+iBJk8D8C8yb9GyhDFQF+K9RGV3XTx9HqeGyd11ujJOh8ednxRfU4M6vsz/ZfPyf1BLA
wQUAAAACACHTuJALUNf3xgDAAA4DAAAEgAAAHdvcmQvZm9udFRhYmxlLnhtbM2W0W7TMBSG75F4h8j
3W5wsW7NqbBSNVeIGITbEtZs6rUVsR3aysGfgCvEevADibeCt+DYTtqsTaoWNCBV2/TUPnG+/Oc/
Prv4wHPvnirNpEhQcIiRR0UqZ0zME/T2bnIQI0+XRMxILgVN0APV6OL8+bOzepxJUWoP5gs95mmCFm
VZjH1fpwvKiT6UBRXwZyYVJyX8VHOfE/W+Kg5SyQtSsinLWfnghxifoCaN2iWLzDKW0hcyrTgVpZ3v
K5pDRin0ghW6zVbvkq2WalYomVKt4Z557vJxwsQyTRBtJOIsVVLLrDyEm/
HdinyTCqYH2J7xHHk8Hb+cC6nINAd2dRCh8wacV48F4RC8Y5xq7xWtvTeSE2EHFERITQMYc0/
yBOEQXif4CB/
jCN4hnEXIN5nSBVGalsuB2IUzwln+0EaVzWvHF6xMF238nihmFubmaDaHPyo9xQmCR4JHl/EIuUiQo
Bgi5mgiISzKHSAPO+toGbFjUpvHDgkmEzMGIpCnmWXX6TsJbRD58eXj92+fLQiSl6+AEky3IG4Zv62
EW+8mowAYYWATtK9eRvFJHyNSlbLJuxui5kaOVohwjG9MdB1R0EaGEEUwKdgPUQPCoPsrHGY0I1VeO
kJdpVgMIMgVhjCOJ30YTJFvVcpvYLgEAecDFXMFaohs5ZjqCfeqGF0zrXvkMIjBPPjwxgkdSgiEfg2
RUXx8ta4GfLoNg2GAHxXMBA4TNKsZKpifXz/9LwUzSMjcAxxW547QH9VLazI7Wsq1rBSjytjsgFxGY
BynVijGYKO95MLljKrWlrr2MUhjSC9RW0grg30CvbyDRmcavO5lceyeVPfTlcJ6swld+HGzGTDSQRL
tZVYGshZZkdjqo90GtXuruSY5myrWyyHEE9tsrXmAOp7aQqD21ywE2nAYjfa0ENO7H1vIEugWC+ESd
hwFSalFsd526VxS77VioukA6z2nvULnu18yvfuTfyWZdrVGertL5rYDo7//
```

wn62Te2+n5pFGNsdVmOrzbU3Gs/W8rGqwaednVo0gm0UHGbxy8bTnOjzX1BLAwQKAAAAAACHTuJAAA
AAAAAAAAAAAAABgAAAF9yZWxzL1BLAwQUAAAACACHTuJAASIiH/0AAADhAgAACwAAAF9yZWxzLy5y
ZWxzrZLdSgMxEIXvBd8hzH032yoi0mxvROidSH2AIZndDd38kEy1fXuDf7iwrr3wcjJnznxzyHpzdI
N4oZRt8AqWVQ2CvA7G+k7B8+5hcQsiM3qDQ/Ck4EQZNs3lxfqJBuQylHsbsyguPivomeOdlFn35DBX
IZIvnTYkh1zK1MmIeo8dyVVd38j00wOakafYGgVpa65B7E6xbP7bO7St1XQf9MGR54kVcqwozpg6Yg
WvIRlpPgerggxymmZ1Ps3vl0pHjAYZpQ6JFjGVnBLbkuw3UGF5LM/5XTEHtDwfaHz8VDx0ZPKGzDwS
xjhHdPWfRPqQObh5ng/NF5IcfczmDVBLAwQKAAAAAACHTuJAAAAAAAAAAAAAAAAACwAAAHdvcmQvX3
JlbHMvUEsDBBQAAAAIAIdO4kDIFAZQ5wAAAKgCAAAcAAAAd29yZC9fcmVscy9kb2N1bWVudC54bWwu
cmVsc62Sz2rDMAzG74O9g9F9cdKNMUadXsag15E9gOcof5hjG0sby9tPBNq1ULpLLoZPwt/3Q9J29z
N59Y2ZxhgMVEUJCoOL7Rh6A+/N690TKGIbWutjQAMzEuzq25vtG3rL8omGMZESl0AGBub0rDW5ASdL
RUwYpNPFPFkWmXudrPu0PepNWT7qfOoB9Zmn2rcG8r59ANXMSZL/945dNzp8ie5rwsAXInQXAzf2w6
OY2twjGziWCiEFfRnifk0IluGcACxSL291jWGzJgMhs6yY/uZwqFxDqFZF4NnLMR0XQYs+xOuz+6p/
AVBLAwQUAAAACACHTuJAfMlJfmIBAAAUBQAAEwAAAFtDb250ZW50X1R5cGVzXS54bWy1lMtuwjAQRf
eV+g+Rt1Vi6KKqKgKLPpYtC/oBrjMBq37JM1D4+04CYQEUSlE3kRLb9xzfWB6Mls5mC0hogi9Fv+iJ
DLwOlfHTUrxPXvJ7kSEpXykbPJRiBShGw+urwWQVATNe7bEUM6L4ICXqGTiFRYjgeaQOySni1zSVUe
lPNQV52+vdSR08gaecmgwxHDxBreaWsuclf16bJLAossf1xIZVChWjNVoRm8qFr3Yo+YZQ8Mp2Ds5M
xBvWEPIgoRn5GbBZ98bVJFNBNlaJXpVjDVkFPU4homSh4njKAc1Q10YDZ8wdV1BAs+UKqjxyJCQysH
U+ytYhwfnwrqNm9dnEOVJw5zN3NqzbmF/
Cv0Kqmr7XXV3adZPGNWtA5OPtbLFNdsr47qgcqr31qPkwTtSH/UPvOx3siWyjT0ogELE8Xvwf9hy65
NMKtLLwHwJt7kk88R0Dsn32L26hjemQsr3Tht9QSwECFAAUAAAACACHTuJAfMlJfmIBAAAUBQAAEwA
AAAAAAAABACAAAAC+JwAAW0NvbnRlbnRfVHlwZXNdLnhtbFBLAQIUAAoAAAAAAIdO4kAAAAAAAAAAA
AAAAAAGAAAAAAAAAAAAEAAAAColAABfcmVscy9QSwECFAAUAAAACACHTuJAASIiH/0AAADhAgAACwA
AAAAAAAABACAAAABOJQAAX3JlbHMvLnJlbHNQSwECFAAKAAAAAACHTuJAAAAAAAAAAAAAAAAACQAAA
AAAAAAAABAAAAAAAAAAZG9jUHJvcHMvUEsBAhQAFAAAAAgAh07iQJd9M7pmAQAAeQIAABAAAAAAAAA
AAQAgAAAAJwAAAGRvY1Byb3BzL2FwcC54bWxQSwECFAAUAAAACACHTuJAdC+l9k4BAACPAgAAEQAAA
AAAAAABACAAAAC7AQAAZG9jUHJvcHMvY29yZS54bWxQSwECFAAUAAAACACHTuJAhGSGoyUBAAAOAgA
AEwAAAAAAAAABACAAAAA4AwAAZG9jUHJvcHMvY3VzdG9tLnhtbFBLAQIUAAoAAAAAAIdO4kAAAAAAA
AAAAAAAAAAFAAAAAAAAAAAAAEAAAAI4EAAB3b3JkL1BLAQIUAAoAAAAAAIdO4kAAAAAAAAAAAAAAAAA
LAAAAAAAAAAAAEAAAAHQmAAB3b3JkL19yZWxzL1BLAQIUABQAAAAIAIdO4kDIFAZQ5wAAAKgCAAAcA
AAAAAAAAEAIAAAAJ0mAAB3b3JkL19yZWxzL2RvY3VtZW50LnhtbC5yZWxzUEsBAhQAFAAAAAgAh07
iQDZUbjGqCAAAqzMAABEAAAAAAAAAAQAgAAAACRkAAHdvcmQvZG9jdW1lbnQueG1sUEsBAhQAFAAAA
AgAh07iQC1DX98YAwAAOAwAABIAAAAAAAAAAQAgAAAA4iEAAHdvcmQvZm9udFRhYmxlLnhtbFBLAQI
UABQAAAAIAIdO4kArYcZzFgQAAAAKAAARAAAAAAAAAAEAIAAAAHAOAAB3b3JkL3NldHRpbmdzLnhtb
FBLAQIUABQAAAAIAIdO4kD9Jy7AkgkAAChlAAAPAAAAAAAAAAEAIAAAALEEAAB3b3JkL3N0eWxlcy5
4bWxQSwECFAAKAAAAAACHTuJAAAAAAAAAAAAAAAAACwAAAAAAAAAAABAAAAC1EgAAd29yZC90aGVtZ
S9QSwECFAAUAAAACACHTuJA0a7vxPgFAAAkGQAAFQAAAAAAAAABACAAAADeEgAAd29yZC90aGVtZS9
0aGVtZTEueG1sUEsFBgAAAAAQABAA0AMAAFEpAAAAAA==

other-examples\src\main\resources\example-files\story-about-happy-carrot.txt

Once upon a time in the town of VeggieVille, there lived a cheerful carrot named Charlie.
Charlie was a radiant carrot, always beaming with joy and positivity.
His vibrant orange skin and lush green top were a sight to behold, but it was his infectious laughter and warm personality that really set him apart.
Charlie had a diverse group of friends, each a vegetable with their own unique characteristics.
There was Bella the blushing beetroot, always ready with a riddle or two; Timmy the timid tomato, a gentle soul with a heart of gold; and Percy the prankster potato, whose jokes always brought a smile to everyone's faces.
Despite their differences, they shared a close bond, their friendship as robust as their natural goodness.
Their lives were filled with delightful adventures, from playing hide-and-seek amidst the leafy lettuce to swimming in the dewy droplets that pooled on the cabbage leaves.
Their favorite place, though, was the sunlit corner of the vegetable patch, where they would bask in the warmth of the sun, share stories, and have hearty laughs.
One day, a bunch of pesky caterpillars invaded VeggieVille.
The vegetables were terrified, fearing they would be nibbled to nothingness.
But Charlie, with his usual sunny disposition, had an idea.
He proposed they host a grand feast for the caterpillars, with the juiciest leaves from the outskirts of the town.
Charlie's optimism was contagious, and his friends eagerly joined in to prepare the feast.
When the caterpillars arrived, they were pleasantly surprised.
They enjoyed the feast and were so impressed with the vegetables' hospitality that they promised not to trouble VeggieVille again.
In return, they agreed to help pollinate the flowers, contributing to a more lush and vibrant VeggieVille.
Charlie's idea had saved the day, but he humbly attributed the success to their teamwork and friendship.
They celebrated their victory with a grand party, filled with laughter, dance, and merry games.
That night, under the twinkling stars, they made a pact to always stand by each other, come what may.
From then on, the story of the happy carrot and his friends spread far and wide, a tale of friendship, unity, and positivity.
Charlie, Bella, Timmy, and Percy continued to live their joyful lives, their laughter echoing through VeggieVille.
And so, the tale of the happy carrot and his friends serves as a reminder that no matter the challenge, with optimism, teamwork, and a bit of creativity, anything is possible.

other-examples\src\main\resources\tinylog.properties

```
writer.level = debug
```

pinecone-example\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>pinecone-example</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-pinecone</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</project>
```

pinecone-example\src\main\java\PineconeEmbeddingStoreExample.java

```
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.pinecone.PineconeEmbeddingStore;
import java.util.List;
public class PineconeEmbeddingStoreExample {
    public static void main(String[] args) {
        EmbeddingStore<TextSegment> embeddingStore =
PineconeEmbeddingStore.builder()
                .apiKey(System.getenv("PINECONE_API_KEY"))
                .environment("asia-southeast1-gcp-free")
                // Project ID can be found in the Pinecone url:
                // https://app.pinecone.io/organizations/{organization}/
projects/{environment}:{projectId}/indexes
                .projectId("75dc67a")
                // Make sure the dimensions of the Pinecone index match the
dimensions of the embedding model
                // (384 for all-MiniLM-L6-v2, 1536 for text-embedding-
ada-002, etc.)
                .index("test")
                .build();
        EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
        TextSegment segment1 = TextSegment.from("I like football.");
        Embedding embedding1 = embeddingModel.embed(segment1).content();
        embeddingStore.add(embedding1, segment1);
        TextSegment segment2 = TextSegment.from("The weather is good today.");
        Embedding embedding2 = embeddingModel.embed(segment2).content();
        embeddingStore.add(embedding2, segment2);
        Embedding queryEmbedding = embeddingModel.embed("What is your
favourite sport?").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(queryEmbedding, 1);
        EmbeddingMatch<TextSegment> embeddingMatch = relevant.get(0);
        System.out.println(embeddingMatch.score()); // 0.8144288515898701
        System.out.println(embeddingMatch.embedded().text()); // I like
football.
    }
}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-examples</artifactId>
    <version>0.23.0</version>
    <packaging>pom</packaging>
    <modules>
        <module>devoxx</module>
        <module>chroma-example</module>
        <module>elasticsearch-example</module>
        <module>milvus-example</module>
        <module>pinecone-example</module>
        <module>other-examples</module>
        <module>redis-example</module>
        <module>spring-boot-example</module>
        <module>vespa-example</module>
        <module>weaviate-example</module>
    </modules>
</project>
```

redis-example\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>redis-example</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-redis</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</project>
```

redis-example\src\main\java\RedisEmbeddingStoreExample.java

```
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.redis.RedisEmbeddingStore;
import java.util.List;
public class RedisEmbeddingStoreExample {
    /**
     * To run this example, ensure you have Redis running locally. If not,
then:
     * - Execute "docker pull redis/redis-stack:latest"
     * - Execute "docker run -d -p 6379:6379 -p 8001:8001 redis/redis-
stack:latest"
     * - Wait until Redis is ready to serve (may take a few minutes)
     */
    public static void main(String[] args) {
        EmbeddingStore<TextSegment> embeddingStore =
RedisEmbeddingStore.builder()
                .host("localhost")
                .port(6379)
                .dimension(384)
                .build();
        EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
        TextSegment segment1 = TextSegment.from("I like football.");
        Embedding embedding1 = embeddingModel.embed(segment1).content();
        embeddingStore.add(embedding1, segment1);
        TextSegment segment2 = TextSegment.from("The weather is good today.");
        Embedding embedding2 = embeddingModel.embed(segment2).content();
        embeddingStore.add(embedding2, segment2);
        Embedding queryEmbedding = embeddingModel.embed("What is your
favourite sport?").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(queryEmbedding, 1);
        EmbeddingMatch<TextSegment> embeddingMatch = relevant.get(0);
        System.out.println(embeddingMatch.score()); // 0.8144288659095
        System.out.println(embeddingMatch.embedded().text()); // I like
football.
    }
}
```

spring-boot-example\.mvn\wrapper\maven-wrapper.jar

BASE64:

UEsDBAoAAAgAAOdJqFQAAAAAAAAAAAAAAAAJAAAATUVUQS1JTkYvUEsDBBQAAAgIAOdJqFT5+/Ltqg
AAAEsBAAAUAAAATUVUQS1JTkYvTUFOSUZFU1QuTUaNj8EKwjAMhu+FvkNfoMXNi+y2CYKDgbih57Bm
Wuza0nWKb+8mTMEpeAok/5d8KcCoBrvAD+g7ZU3CIrGgZNs6jS2aAGFo8koFjQkr4IqGHT04h57l4G
fB15aliERESemwVo2qp6mR1iesOiNLHdRDKW0TbuCRbWxv5DP2Sf0+nvVKS57LCx+JUX1FydojBJQ8
u09Mnu7ZTvcnZQatWMRzrbf0l4/+c6bkAVBLAwQKAAAIAADnSahUAAAAAAAAAAAAAAAABAAAAG9yZy
9QSwMECgAACAAA50moVAAAAAAAAAAAAAAAAsAAABvcmcvYXBhY2hlL1BLAwQKAAAIAADnSahUAAAA
AAAAAAAAAAAAEQAAAG9yZy9hcGFjaGUvbWF2ZW4vUEsDBAoAAAgAAOdJqFQAAAAAAAAAAAAAAAAZAA
AAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL1BLAwQKAAAIAADnSahUAAAAAAAAAAAAAAAAHQAAAG9y
Zy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvUEsDBAoAAAgAAOdJqFQAAAAAAAAAAAAAAAAPAAAATU
VUQS1JTkYvbWF2ZW4vUEsDBAoAAAgAAOdJqFQAAAAAAAAAAAAAAAAoAAAATUVUQS1JTkYvbWF2ZW4v
b3JnLmFwYWNoZS5tYXZlbi53cmFwcGVyL1BLAwQKAAAIAADnSahUAAAAAAAAAAAAAAAANgAAAE1FVE
EtSU5GL21hdmVuL29yZy5hcGFjaGUubWF2ZW4ud3JhcHBlci9tYXZlbi13cmFwcGVyL1BLAwQUAAAI
CADnSahUm0fFAXEAAAALAQAAFQAAAE1FVEEtSU5GL0RFUEVOREVOQ0lFU62OQQoCMQxF9z1FLqC9h+
BOmHWc/tEMNAlpGdDT2xnwBPp2yfshP2c6/
UzKmW7B2qTLBipwaIHOgka2UH9KIw9bMfchO6KKotASVofEfl55g5KPhcWDVd7D31/
fgbuYnvfcH8qm6/FrCnZH0IUjHXwAUEsDBBQAAAgIAOdJqFS0tOKGbQ8AAF4sAAAQAAAATUVUQS1JT
kYvTElDRU5TRd1aW3PbxhV+96/
YcqZTaQamnTRpG+eJseSGrUNpRLluJpOHJbAgtwawyC4giv31PZe9gaRkd/pWTaY1SezZs+fyne+cx
Qvxub9FL8udEu91qTqnXjzz5D+Uddp04uv560L8TXajtAfx9evX3zy5aDcM/ZtXr/b7/
VzSNnNjt68a3sq9eoEL76/vflqLxepKvL1ZXS3vlzertXh3cyc+rK8LcXd9e3dz9eEtfl3QU1fL9f3
d8ocP+A0J+GourlStOz2Acm7+wmsz8yeaCbeTTSNaJTsxwEkHZVsnZFeJ0nQVrxK1sWJ0qhBW9dZUY
4lfF14UPltpN1i9GfF7IZ2ocEtVic1BrFXJQr4C+daM2534TpgaPmh4zpRjq7rhWC9jTxQrTX+wers
bhNl3ygpQCRbq4SDkOOyM1f+m/byccyuGnRwEbLq1EhZ2W3rI2yFTQG1lI65J9IkSY4cHJO2VkCVJC
VqAGeBZL8bAA15BrRxvDQYdrGkKIa0KHxpSusDT4LdjV8Gy0rSt6bwk/6DY62HHcnjDuXhnLOnRj7Y
3EDHJqtHhwUczL2VGR3HiQl/yUrNXtgD3WfASKqE7/nchBiNKCU7H57wU/oksYEUrO7lV6Dzc143lz
itWiP1O0fHB+7SvJNm5ZfYaowmkXGjQhNzjdrpHSbWuwZq9siWKvvj29e8vaTsD5mHDB0Hj4AawOvo
A3GSVCxJB5EZ1YIRSgysn0jM9k8t/NuNMXMBa/JedXeZeh//
QJg+6GlGWFXl8eAHqEbTVDhUBvVvtHAU8xRknAbnlJNTWsFsJKQjp1R5HWm9VrayF5fRrTRb/
hFu0ptJwNElZFRysu7IZyRSQhKIzg2h0q3F38KMz9bDH8HK0ITilAuuH3CNBXgw/UIT8r/
V2tPQ7uKVRGXzcbP4FoXCquuwO/B24Y2woP2prWvix3MkOtA4JAlHROXxShoCibxr/sRZSsHlIXDE9
oJdxdExIm15jQhlSzh9zC5EAZ4CvJwfO0QtO+sDo7VAO526rKi3FcOjzY3809tMJKOzhS9KYcAgjLa
WA7sIxYgKw6fyxWlkBkDxI3chNE/I/w6UC0RQDsJQ+lGTEhYBuYAZ4OMIbWwoe1mRWOQxYW8hCQVsv
4gIOoB5l28POsBCgHcKcF+KTi75XsPMjJFNj9pfJClfK6gew4oMSaBA3O44A3OO8DfzpvSS2QVB8Ix
06r6NUrHAPjH6IHsYq3Irchbmw3+lyl4EBOGuAGgCZadWDJldiFINpfJ4IBRY2NnwCEd7NeTZ5YVjl
lINIIetL2Mw0lBSwTG91B7uc+vwUjwNO1ZP0L8Sx+bz1MJq970i8rxpWtVLH/FS9tBQpaBc6Rqusag
6QB90nMtwGogXjpJOtugxO1wBEtpYlFYkiq5HRqCdKoXWUqZPX3yKU+xp/1uPHORBTNtsvGtAnXKil
UQ8UNvEJxXDlmUiQZNg2tAp+f0r5IkuKAVHfwNZNgG03bgA7PHgE3kHRRZqTej4VaCPC8RNaEbxM5e
7ZapETFURl2h7jfaPAmDWY4mny8mXVXszimWZeFtf7CMuwSDWQgNYAGBfohY1sKI72Ftd1RD7Gzltf
YBbkRlfJUGinwaVkIfu74tlSFLEr3wP+SzoBIuoGFzdAKUFaVrIiFXIHN6jW5RAONXdUWEJKqpH+CX
Y/Vj5mK5Fr5UYvMhiZREFmbbQbcNxydFTlaceW8NLTyI+EeKk0qcdghOlZQzzCUVyvy9GMDpK3lfYT

```
Qp9N7ChQLuX0tiPsh1BEH5Fhz0YigtVsBfaWIs/V+ew0hY/4dTx2yMDPUp7cgIiP7dGmYgfKbBTEE1
BGRUgOSuf7pCR06rcR4qfBbUsD9uZyjYQ3Sz8Goq/n4q9Iq3Dbt/H4gVmJ9cjF1cfq2WYmS7MclRVU
SZEZSCCEgM7E4ogXADmEUwLD69UAlgnhB9DXVHuNXKMz3UvyvIMT48eXwHrsFhsnc5DNcHhZWwWfNB
C7B1MikJ9Uc9//4Yah24IVkGM9xvEJ0iU478cNrAUrQqD2jYRAj9+AzlxqHX3jiUXet+U0P2IxkeWT
Hc+Uc8IWdtAfMwfdSgTd/wPvXMAy1Q+YYNByDIEigYKOG6JL0fNZM+8BXQdhO/
mgiOUFhaiPNnWNPA+KgGoAfvl/AVGMHdgxEQc8UfaskGAmnAxNwD4Ku8q+b7DdNB04nayM2OVVKxup
wd78bHY4sCIJya0bcbOD7HVOWk3ZWVtAn9DRKB1qX574F+4S2mDTKV8RAf6AkURWT8uOF4QDcYfrqy
2ozyRvqpzfYo+uCLVuLpY1+j/2Qg6QCmM6OmXQW1ZBbiX+TCDnG/eLVLAit7bGuZdkMDxGaUbkT/
wZPC9FI/du1AMetVFbLgJgsaB84gRHqPgcwFFNYMWdb7WTnDI55xCOFfzRElMFMUzFppEYKFNoRn2m
hEYj5ZgveYFVcXXAFEXvhViRLhC2Cr4MwRetC9KwT6wYCr6ZizuVT4bmtHUrDwnZjlEIcFAHbjPBo2
dYHrkEaSNsNgLIURwho4H/N7EiT9tmLuFPIFmRWiEySAqtVin2cm0a6Im4vgfsehPq7IW85JOOEGlb
1BfV434D3KrhiAhaOfWN3SH+nRxUUn047iS+pzIa9txke/LgJlFp7KOwf+ehjsUQgvZBdxgn3D26bH
uEuBjSKBNb9y0ZQ7Gc6c5ltrNVAyRYEXhz1sJTdwAaHR8u2zhumAKiwAxL1bHw0V0gLFYKeVORkQkK
0SGlmz8bjyDO6HMMqfiXmBujZ5BBylWGCC1UGTwmmpMzzg6pcPFJTkv11GjVJYJW9L9v/NDVs9XN/f
Lt9QyS73Ege2Pa+T2Qcmf75NmVQcCZTDmxLPkrExVaTwk+lBX1mCno1FmzIihJnPNmYjyoETLwQegI
xZfYNRNz3sJn7UrBBjIaJR22U/mU3i9J2QrECDZ9E9SUQcdk62ShSVS5Z3X4PgfzSZDleT0dQAldJ5
zBkrlNFfBUvrHFqZVl4HrZlMv3BmesVB9lChEI6ADZWSDQVi/
xkIfomw7nc9AwI7FQEprQ+x13YYhfp2bO/
E3kgVvpOOSDHiI1r8hQpur43CLEOkxm87FsyKrCf1vsd/KIzKQE1b2FviQTCra+A0fkZ6J+CscbVaW
6amwDbZ1ETAAW7v+CO48xjQwchhhghrPJRNMq6JmYB9jxOP7YME/dW5w1UeoqiLbSsJ4JwNHgK3MFC
vHnyFXGkZxG1jphuWcYfBrtnbkyYjHZXZGpz2hTpLSpqVk8PNGK5NO5mEokD7fOpnlJgZPbqkkVjqw
bZ8lEpTGOJmOZ2KkcdQITh3xLzY6/CeBeNbFANxcfOqiijpymHmGjUmP7SxKzC5I43zgcs8hsmJWNs
Z4cXSWmjzseD3KY6m3y6fN/05p5mkVqZgHDIpi6VuH2kdevzICL4u0N1ZeN4aYM03ZL7R2WEVLNjVA
OnKoUXwRhGmQu8Rsxu+ABKVgxtkRb6Oko8A8+Q6gjU4+qzCCegDcaxKqttHyvdNx7+LuAPwEUBgLiE
BYzHl0ZQs6BKXd2I4SG9xdqTF/CNYZscW4WGQ1OvZR9wJm+/
wg6+Rjmh0PQBo1DpKQ21arfRu1vj7CgO/AJlnRyKRR+0+L1NGoDVgbeUcIBvSti04GT2pP5bMim4Dd
fDc6UALbUn+fiSjtqnfDSthYfgX+CXQ4xCaKqmwM3sNR5Y4uVYIC8SM1LmoIVyWE+911S9QJ1xaHBc
YuaP43jy4lzL3GuBZA/W6zFcj0TPyzWy3Uw7sfl/Y83H+7Fx8Xd3WJ1v7xei5u7/Fr+5p1YrH4Wf1+
uroDuaL4BfsTpqEsn0YQrVTYmTRlEc1IZcOoATS6ZihoiewqxYMz75f376wKsvnq5XL27W67+ev3T9
eq+ED9d3739EbRc/LB8v7z/mULo3fJ+db3m1wcWXsbt4g4c9uH94k7cfri7vVlfc7Xl28IGbxZA/
x421XTrQDcz3BVOwwU8Z01vNdJzOnAN0YWPUPwlxM3mpTxtdA44ER43wLV2hOzOlDq2yQzq/p6VprH
5RetpM8ux95c5fA4mxUXvtdzohi7Pl1h5BdCfbiA9WAZ81dCwE3SETjsbtYSbLAigIR8ZdGrbaGBfp
bos4m13MRnlxsnPZ+P9gokCzvQbvSFCR8ptcR4R7y3ClgO+geDodvx8fjB6TsoHDmWCyxpNG/uJALl
WtnI7neHj6vBKQHo5wPUK79az22dIKCC2fJWABIZnungh54UGhMaZG+iN42rLd+ZYxWOtxlvj40aXr
DlGjBn5G915Z2a4mk8MLp69Ew9a4bEbwwG7Naba6yafHX6Comz6XuKUEDnBiIrXUjej5Wokm3rsErm
hInjmTRC8BcDgze3BGysHgYNxiAT9eBDnZcRhuqweNF2S1v71DcgAb4TwcoMXzxnw3VwsSqwJaIWAv
LjzIhXqLCk+7pC6T9P1+LLw2eu2wELLnTE8BaVJ5+SynWauwNtqRXgCUEcayq5UfIiex6Ae/
Q4Ud6rt8NWSNBBjszZBd2E2jZ9CEW95hbCDzJevWuA8mC+
+v9IBQWOD8aPZYyfErWQ0GNkzE5zOR2+0dE12GxI5t78WoSGu/xqBNMEo6UtMJ92iJERPk6IsDPxMG
HsmXTM+Y8JzvpNt6mibStXQrvAKYMbVmdG5tC0hUSDX0YopnUdr022ZnxwDJkNXjs0qD1GL07nx5uD
JRjrQAS2QbBrJ/D6Lxow2Rl04gK9XV1hXz70GR78vbm/hkeU/36ALaVoAiHrwry/
kr+7hb6TKPt4lwd/9Fy4o/GsU02lCoNUGssZCGz6EqUaROvlaq6ZyAgoEJDuD/
gZvKRVE5uyXX2cR+Ggy4avdIQQToarv+rJOei4urkz3h/
i+QJajQfjvLgV169SmOqAXEAlA8aMevjvIynZ2N4u54g6A54/xIpSaelYAcAIWNg4vqPhpPycNKE7P
```

ctxAlCFj5baLaGYfinG4Wt2o9MoK3ZAGTRwunIFyNLhGDJ5hrZjefPqXX1BNCDwd7+O95cK9axzPpC
GHtOUOb6w5GNJl4i8H+PtV/EJ6g55Ht6y/0uM+SKqsZ5qGT5G/ECou8IH4zuXl9ygi9CMIBFy+/
Pg80Hjd+TaUoDFGVKQ4InX9ZkPTMjkZ2YVAlkMI98+9cvoeuPtqff0SVKYlX8LQn+Ie/
p0zFJON1E7fcMJLg/yBpxj4/0i/A/Ems62VmqgQgpxoDcQMHK3bjhBwQAmgLHTHb/
b5aUni6+70XPMX/wFQSwMEFAAACAgA50moVEux2AyAAAAArgAAAA8AAABNRVRBLUlORi9OT1RJQ0V9
zLEKwkAMgOE9T5FRB9t6bm4iOAhOFpxDL/YOyiWkaQ/fXhFcnf7l44cbrVzwYaTKhlcyOIu+LI/
JMXT7wy50IWCfGE9Kwyd3eXolY7zIUiJ5lgLQpzyjmsRlcMxlmJbIM84/GnnlSZQjksP/F26Sux7bt
tba0Jc1YmO7bQDgDVBLAwQUAAAICADnSahUWEym3yMHAADvDgAAMwAAAG9yZy9hcGFjaGUvbWF2ZW4
vd3JhcHBlci9Cb290c3RyYXBNYWluU3RhcnRlci5jbGFzc5VWaXsT1xV+r7WMJA8JEYZYpBCl1CAbW
0poIIns0hCHxUS2qc0Sh7TJWBpbA/KMmBnZuFtC99K9NG3pkm5paZsuIW2NHULo8jz50H7Nn+mX0vf
OSEKb3ad+rHvvnHvOuee8Z7n3n/956x0AH8Q/YkjgTAyDeFbBjILnurELZ2N4Hh+N4WN4QQ4vxhCAF
sMs8nIoRKBLylwE85K7KAcjhnM4H0EpggX5bUZgKSgreCyGbbigwI7BgRtBRcFiDEu4GMGynD/ejU/
gk934FD4th5cUvCwplxR8JoY+fFbB56L4PL4QwRdj+BK+HMMeXFbwFYHwiGEa7kGBQKr/
tEBw1CroAvfmDFOfqCzM6vZJbbZESjxn5bXSac025HeVGHSLhiOQyVn2fEYra/minlnQFnUzs2Rr5b
JuZ56yLNdx+TGuGea0q9mubg8LhBy5FBhInc2d0xa1TEkz5zPTrm2Y88M+xTSszJxR0jMnNLc47Nmm
2fM8bUsHEYGod+4xa4Fm9XTSIBDxWI5rNr3JW6arX3RHS5rj5CytoJOYqIrpbubUVK5hy1dvmB5J4L
6G8z0S92Nyf1x3i1ZBYHsDg63PlfS8m/H3yLmtGcjlcg3Mnla1IwMHpebDF/N62TUs01HwVYZmzjAL
Oa1iEmzbc2Yg1RGxdUAwGAHNtez6gRXXKGXGqlRydBXo44Mt0k8bNr2w7GUCrmsLdcBlfGlBB00jnc
6XDu3fWPW6cpuYPvnz41rZw4vJyyRX8DWmuIKvK/gGoZq2KnZeP2JIOBOdUi8tdasYQlrFYRwRuH+d
kAuojTsqvolvKfi2iiv4joJXVHwX31PxfTkM4aqCH6j4IX6k4sd4VeARlkM6z0IqahUnXS7pFznlpf
Ilyy4VnHSpGr50LY4qfoKfellETNNFZrGKn+HnKn6B1xT8UsWvcE1gc6MOZvCcQPesYWYW9nlfKn6N
37BMZCoyT1qyScVv8Tp13CVPzp4j8Ap+p+L3+AMFZ71g9jbjz05wftLLPwV/
VPEGrpPF92mowZ6hAWJLP97EnxgUFUfxZ4ZFxV+wouIGVllz7XFVsYY0K/quTSeLtrUk46viLdwU2O
FtUUYGdcJyj1gVs1CvCAVvq7iFdwRGR61KqZA0LTdZYn25etIt6slxCWeyhnby+KGppGFWqQWDyWHM
VqSe5J4+Z09awW0Vf8XfZEj/LjD0fzW1Ji/qBtYcWC/Za6g0VU5NFWXGJhtUhVzrlG0whKn+xk41Nt
ycrGM+51SujTNHzq0taVFL95553Z1edlx9obkn1jS0SsgOQJETmq2bTJr+WiOvnbSOlOzjm5vCrWvs
mJvyFVsqqn33NB3rU6Xxju6Odujc21MbHBctcV1t3X2p9rujzUEedI9rHZp1rFLF1WWesmun1uunru
XraTW6fjM1uOsDzKqlHydsi7nkLgs83sGojc2s61ZsnVYust2l1ves1WQZt9pV9WQHsbNtgPRvdKH1
rrfHx4VhLlrnad0Tjcf4fafpmCqpv51UK4a6E7IRMJLdhlOvJYF9Ha/
Asy3Eu61suP85lp6pL7XV49GOmv43tu0XZD1lWi9Ypag5E0xh781FO4Km99GcPnX3Q/mS5RDCTVqhM
F0plxlxRye2vanmCvHbZkOBecd6jw35UpuanDwpX3GtWy0JWk3l8JxlL2g06plUu8TGGdMhlPV03do
pSU/jIT6cE5B/AQh5OXPM8CvDWXAODdyAuM5FFx7mGPaIu/EIR9VnwD6+wsH1o9hPLgqLHeRTSLs0M
HgDXRM3EZgJ3kJwJjC0hpDACsLTq1BWELmBaDa4ilgiuILuRDCurmBTNhS/
Z3AN9wpeUJsFVnHfmXh8ML5lDT1daKYnQvGtVLxtJhC/f3oFvdlwIixISMwE9vJ7+5nrnl/
S8AHEOO6lYYPYTj/76OkQjT1AV47ShRdo/
gU68LJHAY75DlSdk6sDeMxz+BIexxN0NMxHfparAHU9j2GMIEieZ/
AhHCQoF7AZH8aT5DpUhcrfe4p7o6QESXmaWvgS8kHriiDi2fje3vgDnq/jg4FbeN8qdlzF64Pxnat4
cEJkg4QwKZANJUJreEjgKg7I1fsFb89d2fDQu9iVCL6LXrJ9QOAatmaVRDBBrHdfQ9QnJsJvkO3ZJr
bjVBhMhDy2gzXifnlKluTb2UgHxVFKRJsUR27fxJ6ZN5GK99eisAom0N7bxOEw/Z8gViXYqHDeScQu
V+cr9PtVvEa1faS94n0HvbjtI3YgulFy9RDZncR0NzF9mCiOUOeLnPOcL3PvCue3GU0Zv0eJ827ifZ
SRDOEBvujGcJyxIr71mL7nRZJNGf+ibTKmMlbbOPuSubrkOCUnuFeAeocLRcGkghPe/0cUTCmY5hqC
y+7Qv9Elad3HOMjfHWxBtEWAquToC9zhgYE6A7dq+XGSv1Ne6Z3+L1BLAwQUAAAICADnSahUQN2xuS
kCAAAhBAAAMgAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9EZWZhdWx0RG93bmxvYWRlciQxLmNs
YXNzlVNdTxNBFD1Da7csS6lVQPwCpWK3IAviG4akaSsxtpVQbWN4MEM7tGuWGTLd0vCv9EExmvgD/F

HGO20lKBjtw965986955yZufv9x5dvADbwxMKdcdi4a2MC8xYW4rhn4z4W40jbeIAlG1E8tJCx4DLE
wrbfSa8xrJSUbnn8iDfawjvkx0J6Pc2PjoT2CuKAd4OwoHoyULwp9Cb1PfWlH24xeJlRGt0aQzSvmo
JhquRLUeke7gv9iu8HlEmVVIMHNa59Ew+TUSOQwXkupdD5gHc6gsLVEUjT66R3riXCHWruKd3MdcO2
kKHf4KGvJMNixi2948fckyL0Li8iBLuqurohnvlG1cwFllWDQKcqykagOr5slUXYVk0LWQfLWGFw/6
q4bKL6IChzXzpwMOkgYbrm/yGMYbJcq9Tfvq4Wdyu5ctHCIwer8H7ld3LVav3lbsHCmoN1PDawG/
Tao1wfHfdMxTl2pRmy/4/DMNtQ8sBvdbX48wyRjBmMZJ8l4LLlVU86oTikMaNnE/
KYYSlTOrcbarrgTfdi6neQfo5hIlT5Ntc5rfkJDVTG3cuTmEsA9/
JuDQswPw7AkknzDuSN0ZfAFAicvC2KTcbOLn/
EWPYUkQ/9mqtkY1QDbCNFdmZQhWuYBvqeQSN+2pkdYm1TZDqmvyL6JnXlM2Ipi8wnxE8x/
v4MNoEI2Rckq4TrtBp4Z9A4hI/gRr96DjdpjeIWblOlIaXB7dPjJ1BLAwQUAAAICADnSahUm1BSwC0
CAADbBAAAUwAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9EZWZhdWx0RG93bmxvYWRlciRTeXN0Z
W1Qcm9wZXJ0aWVzUHJveHlBdXRoZW50aWNhdG9yLmNsYXNzrVPfTxNBEP62LVy9HrWe2oo/
AAGxrdijokZTY0JKTEiqkqAkhqelXdsz7V2zt6X0v9IXIZrwhC/
+UcbZa8UGmkAT7uFmduabb77d2f3958cRgFU8N5HAjAkLMwZm45gzcR/
zcSzEsajdByaW8NBEFFkDeQOPGCZfuZ6rXjNEs7lthljZrwmGqxXXE+86rV0hP/DdJkXsil/lzW0uX
b0eBGOq4QYM81u9QInWpvTbQipXBOTt99Y6qiE85Va58iWDteF5QpabPAgE1byv+LLu8DavNoTT4nv
Cc7qSt4nAWRefeaep1v2u1/R5TcjF8/
lLDNN1oTaJvevL2lDO9T2GhWyu8oXvcccTyhkNKhlYZihmx9FVLOkzi+yvMBTGq2Mwt/
yOrIo3rj7I9BlMQcu1MIUkw+w50hmSDaXahbY+lo+BkAYeWyjAobH9T/yrZYBOrhgoWniCVd3kqYFn
DG8vdSS0qxPdpxLL45wVQyrkaXKv7vTbMiT0sPvNewxL2coQREnXq5dyZ0MML0YAL1iaOh0jEcovN7
hck5KTiFg2t1NmyIxosVPW9yR/8V1jjh6yBf1FEdG3APQoaeWQpQFiIn+AyDdyIkjRfzIMHuEa9HBD
AGzcJBtDGplB8QZFdS7zE9FPduwQE/akbRwi/h1XDmB+PaFLUlfgmPr+ovLjIdpMn5ZAU5ge0L4kGx
loYqM1pfuAgSbt3cadsOxuiL+HG2Rt8hLI4TpM3Apx+AtQSwMEFAAACAgA50moVMv2XHmhDAAA7hkA
ADAAAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvRGVmYXVsdERvd25sb2FkZXIuY2xhc3OVWQl4G8
UV/seWtWt5cRzlshJIlNu3cYAE7HDYjkkCPoLlOJhAYSNvbCXySqxWOYC2tLS0tEAopAdpgd6BUtqQ
gmOgHL1CS0vv+77vg94XNP1nVpIlW4YQx7s7b9417/3vzcznZ/736BMAzhAdGu4KoA13l+MS3CMf7w
7w8R75eK+O9wXwfnxAwwcDKMfhAO7FfTo+JIn36/
iwhgcCmIWPaPhoAEfwYAWO4mMaHpKDh3WMB3AMEwGcj0d0PBrAY/i4VPu4JD8RwJN4Sg4/
IR+f1PGpAD6Nz8jHcQ1Pl+Oz+JzU80wAn8cXNDwbwGp8UfJ+SceXNXwlgHrcreOr8v01Sf+6/
PqGXMw3A1iLb+n4dgDfwXd1fE/D9yXpBzp+qONHGn6s4ScB/BQ/K8fP8Qvp4S81/
ErDr3X8RsdvNfxOx+8D6MQfpM4/Sof/pOM5HX/W8Rcdf9XxNx1/L8c/8E8d/5Ly/9bwHw3/FVgW2Z9
yrbEtTiJpOW7MSvFr3/72tDtq2W4saroJR8DYbNuW0xk3UykrJTDLTCbjci6WsHvNMUsg2L3L3GM2x
017pDniOjF7pI3EPLZBy0nxJeBfH7Nj7nkCq2umy0yn1A4K+DoTw7QxqztmW73psR2WM2DuiCuriag
ZHzSdmBxniD53NEYfG7oTzkizmTSjo1bzmLnHspv3OnTIcpo3WDvNdNzdkNhrxxPmsOXQ11OjCXtnb
CTtWFPXr7wurZGOnBJxzejuHjOZMbUgJzSVXx/OKBdYmVmnbbnNW/
s3Z9ZoxxLNO2Nxq3mL6Y6qVWrm8LBjpeh6ZaGAQMWwlXJjdkb33GIaBAJd+6JWUrJQRVXW/mbbtRzb
jNOlmM2YWuaYwDxPAxVstpNp1yNTRWnaiU8x302yj8uk3eoCeidpVlSaI0dlOmU57SOMwKAZT1sKW8
9reEHD/zSckFDl+vCsJoTAfK6zw0zFolNjVls8UNPMyWAForkxVyatb7Z3JgjUHWbKWntmlx1VkFlV
BGO1xaBaNWK5npDTY7mjCeZtYR6fY+2M01qzN0d+w1LMWV7NGzqFhdC3YxeFZHgk7aw+p8t04jHJNS
+PK5e2LOO6PEa/
B1BCLU9C1WG3h1xNlLANCMxmIUTTcdO1tuaCsWxKQIuunDmPZUJoq1LWk9S+N+FwWcECpSq/hF9NUT
0B1lauyAOSYdCyh2XzqPAGmanZk1OTPSGR8tpIeSKVT2x3oqNUFkmknah1YUyW3PxpxdskFRI9M5Z7
jxxt8wY9Zsw2cCmuMrAVgwa2yceVcrgDVzH+o66bbErKDiBXrIlSQ/hwvUDPyTeTFS/dUOmBKNOE3x
Ca0NlWegZ7t125NdLV39ve02WIcmnRI25pj0S29fVvYD97GR60SAMBVmxho2D/3N2noKaJCkMY4hSW

```
qCEqxSx2qUJW02Ved6Rdq1mGvT07MkSVmC1QJsOUYokbIog7DDFHzBW4YmuKSAibdjhmp6wom2J4sk
jDbiKcbUlhRiKskhIejqU8zWRpCm+JW6xeKZWKcRnhtFK4aWBgS6RJE/MMMV8sMLALw4aoFiH2FkMs
FIsMXIvrDNyOOwgVmbVGBVT2HUOciiFDnCYWTwtFZyK5PxuKJYYIi6WGWCaWs00ZYoVEwpxJiA+MOo
m9suEbYqVYZeAW3Mr0yIQmnNg1mea1YGpJdKRjcVW5ftXswoZYLWoMvBrXG6JWBq1O1BuiQTSy+UjZ
JgYh3tSh2heL2hBNopkLmuxL3ACntABDnC5aMkvLF1+Rk6j0+tJAwvNJoGR7R8ZefovSxBpDnCFo9y
xDrBXrDHG2OKeALys/
p0jfEghL6r6mfWPxph0xe7hpg+ma7v6kxaa9hzUgPZmdpALX86+DW5mznx12Ull/mrUxZuXpvDyLZ4
mB6WhJhffG3FEFjrAX4MLtJBxLhe2EG06lk8kEXRgOk7afjSR80WBPk6yPVpmRNkOsl8BRPappT6Zj
GZlRphPN8oZjkxQtkWryumWAXwVkkz1LE+ca4jxxvsCilanmlalwzcpUm/pfm/
dpiAtEu0DdyRe2wPKZmfO4jPy2zzY+/bxQyNPNvlp8s81mvPCokE/
ty8vZvKJ7VCGQVHNkwAnsTIskElpqXsZpbUWLOgJMejzlsBpIWW5GiAnI2wML+JQOP72w7D0CzS/
HASU6pzCosk3y3OW39hGk/FhTU+yUtr17po7cVnsZtz8ZFNNR2+z87DY79Zw3O8oUuNaGmMMUsQNJs
/0nZW2Gpt42gyHpTYThkDD3W1enzTgtzauZfsCRri+ZMXzdiZERCQLfXtOxpyjIO+SXuQkFxKrcunP
nz0rCxM7H5MIpPAVn0SDT3093eWSeBFgl11IA4AVZFdMOweHCYPDIbw+bzvDkjkEf+7u2dLd3dl3Zd
enmyMDm3o08a3W/lJh3kE7SmY01RU2fTArz1NVexKBF44kU03MKz9QRdjp5f7B4cKvOD3Ju//IgL7N
ik6fxpM7GmW2MzutubhfRGU1V4CyDgtOgIrZ54S7oAKHpbN3Zq1e5rKqMupUv4VTWgMRm9vB9QRGZ7
dNk8rVMO85XzzTHeMXsPYndjPE5RZC/
vUgxFLsALM1lMTpqOgToJD68MZdetnXgwivPzt078pkzTG1e5Dv2u7LmF9fMzFgrt/
m6F79gF8Li1JrOF0u+bNeTFwuvn8sTR7O6gctbd39f30Du+pM3RWH/zoQzZrKnXVwzffrFc1ckqBku
LMUlaAN7H0oQlKd5gO9t6l1O+qUY4vMyjmLkKOV7Qd0xlNTVP4zSuoaH4eOojL/+BzlVgu18LoSfzz
Xw4QyqOBOVOIsq12I+1uFyzoQ9NbgCrwDUlzQr1Jc0XMIfXiYyZvvILefmBrUJ6MdR+RjKh8QxBCZQ
IW2WKpuVSmcbdJyLOThf2TE8uYwdU/GKOVS+I6c8mlG+ImhM4JTjWBysVO/5ytjT0tisIa6uqsBYkE
sDOvnegGpciOXYkmdwRaHB5SRGMUyz0uDtJElP1zSUPo7ZEwjeCd+DDY9gDq+Qj2PuBOZtC86vH8eC
cVQfgh4MTWBhXX3DMSyaNL8YGp+DqGByDGpdSOOraLKeJlu4uMkYr8nFeA0s7FTurcEIRumNdM+HEm
MxEyu/2AFI3ZX1VDzBLFZw5iC9ObW3cRyntfrq6kO+Y1jMgCxpLQv5guFQ2TiWhnzjWNbqF61ayN/g
exzLh0qPYkVkAisvDfmPY2tIO44FIf84Vh3GplY9pIX0cdQcxvoM7UzSdArrT7aWS4Gl+QJzWwMUCC
gBXdFC5U8+iDKG/WL0YjVXvZ5r9t676e9eXMfrwWrSxtTYi1mTWssI5Ub5FWPqdjF1uwn/
MUomKJvMSCRxB65WMdzEONeQshtxheiDuWgezEXzYCaaZThAaZvWKnAT9SUZ0Wp6cjXzVKZibUM/
QYd1DY4G3vp4tUpr2KNhr4Z9GvZD8FGxkY9N/D1BFAdmZM2x+07Qtn86nypXL6vXkINXuwzgD9AbCd
7auvpjqO1tfFrCL1j3GOqHjqEhyDQ31TUeQzPf4zidyZ2E3XyKAmmGYg8C2MeFXsPwXKtCVecpzQWo
NhegWrySwaCTrJJXMRilKhg6xPOokI6WyLuc55y4kQuTgT5QN46WcazpaQieMY4zg2exNNaOY11vQ/
BsOT6HIFs7VBpsjZDY6msUnG8bx3qJyVAZJ9uGSuuP4txxnEeO9U/h/CMUbZeiHYWiIZ8oyt7qewyd
Q8ENEu5dT9JJnW2ynxUSobOv4dsLSQeq+LyJszcz27dgEW4l0g7gHNyGjXgLEXobpe6g1Nu59HcwJ4
cYvHcycIeo5S7cgHtU+C5nn6xGK2mvpeYmtpAb+OWjnhBeh9cz8DHOe7QbWPM34g38ihC1b+RXqcpr
NvQHVJi5x3HmJrwpU+n1KDlBV0oVVAijN0v83HyCtnw5EgmKShFezjOAOcSRzG2L6pA9qj/2NhzHks
bjWJSBTQPRUtU6jgsbPdgcYcvYeGQKcA4ztffS9/
vYke9HAx7IA05LzvsW+i+B46fHt6m9wMBKRjILnAqIF9Cg/
JXYuZ3B9fx8jiI638N1D8HXHdzE9t0T3Mxnb/AiPlt9wYvlqyzYLV/
+YI98aUfRG+yr0hUEKFgaGfLVR4bKQmWRIX/IHxnSQlpkSG+IDFX5G/nQQj62tS1HclvdMmXzKBf4E
F17GLPBvg22R0ywJT+CRjyqltnv+ZZZZhm3t4MqSRXcXd+qljmb6XybWmY1uhVcfNTSRdDIBrIK5+F
OfvmpcR1zMshKKSGQpB/vwoDak0og/44TgfzbjPr3f1BLAwQUAAAICADnSahUdMw7gLcAAADpAAAAK
QAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9Eb3dubG9hZGVyLmNsYXNzRY3BasJQEEXv2NTYtKD
gJxSh3XQWLrOtBUFQFLsf41QT4nvhmarf5sIP8KOkLxKagWHmDPfOvd7OFwBDvIR4ChEROmt7NLmVN
WHwNsnkIGy05OV8HNeUWv5Jc+WZlNv4/ZsQjU6JFmVqzT7Es+eF/XWJfnkRoftZ/1P3UfkJr9ZtWAp
```

```
Jtso7Oajho5OiUMeNlNC7h+ViNjxdZZqUhH5z+k9sEwgtVEUB4QGB50dP1Wz7Dv3eQucPUEsDBBQAA
AgIAOdJqFRwfgAq4gIAAO4GAAAqAAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL0luc3RhbGxlciQ
xLmNsYXNzrVRbU9NAFP6WlqaNoVQu4g1BrNwhIChisYqAIwwXR5Bnt+nSLoakk6SI/0odr4yDvuiDP
8rxJC2VgQLFcTI52T2X79tz2fz6vbsHYAwPY4hiQEUMgyqGMByDjhEFoyqpb/
liTMG4gtsMES8v3eQIQ3LRdnI6L3AjL/Qtvi0s/ZXDCwXh6POW63HTFE6K/
KekJb00Q09vLQF96wzhGTsrGBoXpSWWi1sZ4azxjEmapkXb4OY6d6S/LyvD/oEYtHnLEs6MyV1X0La
7BrLkKJ2vqWC73rp0pTcrHWF4tvOaYaF3cZNvc92Str4hTaE/5V4+VdKRan5lbscQBU/aVqrvkOdjE
gHcM+EWTY8YQlnpMLRUQySj2DEYWqtCMzSsetx4ucQL5VzVitFVcIchtu0z+ZQMmRPOXNFxz3Nkpug
J/RF3peFHTu+r3BpyCW8EXPU+DlV56EzwDE/KZzS5ldNXMptU73+t6otjof5bsrFVmbO4V3Qo44lD3
qtyq2CKSoztTFWrfppQ1FW76Bii1KR4ZfiGfXca8jnLMG1XWrkl4eXtrIIJDXcxqUHFOQ0a7ilIaZj
CpIL7GtJ4QCN7lElDC1o1tOMaXcxaJp+h45SEGJqrdIah63R4Kl1WmMITs/7kt1UdTP+ih3p92Xy0D
zRakRICQ/uJbWKIzqwsr80vP5+jlE5pKTrpTxajHx5LJPz60qqeXg0NoNbQKo06egC1f+ADWP8n1L2
lXR0aSUbIB3iPBMkLJS+cRzMQrHw0Rg+1gWw+FqddiL7xgc8IDf5E/eC3dwi/IU0owGtCmOQXkrt0q
q+EsBdgd5aiKthxYmsLuOO4iEuEfjnAYCqu0CJMpqv0UuvLxAtl4qhPXKYspaAGx/5BMT8PUEUrVFF
0lKmiZLtO3hWGxDSl1lVmGKe4UKlM3ymvj4j85ThYJi1Y75fpIFY7bhyHpZwJi2Qy8L+J7uDbg97A1
od+KrEfQSMSZIc/UEsDBBQAAAgIAOdJqFQAjFIInxEAAJ0lAAAoAAAAb3JnL2FwYWNoZS9tYXZlbi9
3cmFwcGVyL0luc3RhbGxlci5jbGFzc6VYCWCU5Zl+3mSSf2byEyCcIx7DHRIggoAyIDYkQYJJiIQQA
igOyR8YmMuZCUk8a2u1tfay1QrWqx60ilYUQiJq7CFtbbdu23Xtutu627pV2z3c3brd9co+7z9HZnI
gbjn+//u+/72+93je75sXP3z6OQDnyUk3fHjVhS78vT7+wY1f4zcGXnPiH934J/
zWjTh+Z+B1N1x41Yl/1vfv9fGGG2/iLTcm4w9u/BFvOPEvSvuv+vg3A/+uX9524j/0/
Z8q9b/08Sc33sF/u3E7/mzgf9z4X7zrxHs6f78IS/
CBSv7QiUFOBE4RfeeRTPKLxCEFhhS6xBCnrrj04XajS4qKxMTv+JBxRVIs4/UxwZCJTinRz5OcMtkp
U3Q41ZBpLpkuHqecYcgM2ipnunE+3lbms5TibB2d4xSvITPdqMSrqmVWkcyWcTqao5/n6nSeIfPdWC
+lyrXAKWVqarlTFrplkSxW2gpDznXKEqcspavkPKcso0dluepfoZrPd8oF+l5piM+NbbJKmVarmAsN
WaPvi9y4XD6hj8oiWStV+qg2pMaNdlmnBBcbst4ptW7sxtu6uEEXLzGkTkXVK2ODjja6pVHm6XSeUy
41ZJNTmtyyWZoZctliSIvArA2HrVhV0B+PW3GBp7pmXWVz3ead1bVNmzfVrm3eXLuxYWdj5eb1gsl1
e/37/RXhQKSiIxC0Khr9iT2rGK32SFc4GPG3C+bWRWK7K/xRf9seqyLk32+FK7pi/mjUilVUp4isGF
nGRclaSY2hXUErJigdm68xm5KshasD4UBijaDxFExZyk5X8oItAkdVpN0SjK8LhK2GztAuK7bZz4+C
krpImz+4xR8L6Dy16EjsCdBjc8bWUBuOJ/
zBpN3utpjlT1jVgXhCUHMK21uS76pIuCOwuzPmTwQi4VULRne9Edof1qFaaBME/eHdFU2JWCC8W3Um
QtFtgei6gJqb3x6gp8e1ZcsVVHxMQ+iddm4hFtjVqfPmWFBQnDLOSlQ0b6olSRHN6tpkRSPJz/5gl7
8nXp3JEtnGtEsuNoepeZ9gYlAdXJ0l2ZCtXK4bvizwnWZE54zgpWWmrSfjE3c6dS2aZejGqtVJDrqK
kZ2Q3Bd5gxV1/Eb+qbmJ0BNNJ8O8YbSrRwvYGk39pgR3XO+P2ozES1YlC9GQVu6Y2GXINkO207Sa7j
YrqmbHDdnBKgsmraNd80pHE75gpLWODnuXLm6HOWH5Q4JzhrFSotWWiMR6kgRkWn5qijE2ZshlxEXq
agrsDvsTnTHqXX1ado7pqelxK1HTbbV1JtRTjVYsFIjH1R+CaaNLZgkXMA9Cca3hIS1NljpDaNGU5C
q5ajdm/MtvLjuP1kdCpDHtcVUkFPKHmRVrcgUNt9afSKYXTYjEA92aVkOW2n45l+ID8ZZAmLlGw/
JLFzD9CyPxBn/Ijo0VtBQWYnYCxpLVXNAZvioQFZSPus+x9u4MhNNhztpoONqZyAS3wAonYj30bdau
qKmCFVGjX9QXKrQmSWYoeUD7wrQshppwZ8jK4IFxVbqapo0Uqh9Ik2/
vxtFuKfgZ+rI3vHR0oavLx7JO/bnGkMvZCg3ZyR2vbgum2gH9She4myKdsTYraU9xBn8XqzwT1+MVE
1fjGhPX4jpWnClXiF+wYExAqddZCgfr/QGij1m/
paFl56aaxo3Nm+oM2WVKm7SbYmGA+x8OwWs7A8F27W/l1YxxW8Jq92aze63w/kAsEg7Ry979KUzxmt
Ihu03ZIwFD9pqyT4KmhIR4GCEiDbfUlKhcaUpMGCIzG4W5WSH2SoUpCek0ZT8GTOnCgCHduuseU66S
q5kHuanExrdvYzQJv9eYcq1cJzg75UXux2v7w5vdAGju9XK1KZ/UR4fcQDPSQK8MpnxK1RQsjvpjCV
M+LTea8hm5yZCbGQj57AgDqiLRnrQBnzPlFvm8KbfKF0z5olxn4hE8Sng05UvyZWKkKV/RTd0mX2X7
```

G1kSgknVWltqRnsaw7wmvotjzJNmLbCofmNeehMRfvg+fiA4YyiGmzrDiUDIyuCEIV8z5Xa5Q3DZKH
6YPzc+39seseLecCThZZdNMF+8/nDPkPLF3pruaDINqLCDiOC1uv1tiWCPd0kWFcM8ZMTGXXu5bMrX
5U5TDshBE0dxzJS75BuCHWOZkdIep5qIN5RtA2v5tK2YNFSElbGYv0fB2pS75R5ivSn3MiByn9xvyj
fxSprYNnnznlikS1PZlAfkQRPPaW0YuwLhCh4LNB1vJMjEurpji/
SfIQ+Z8rAcMuVb8u20nFyAFpRVRTqD7bZn2RRob7oreKNDbcHbEYn5GM7zFnsbg5Y/bjEcXj2jqQs6
mcE93kCHtyfS6e3ys+C4+U7S2D5cbMojWi5GJL44TFw25VGt6sNyhymPyeP80JXEb1O+I0/wzPfRR7
45S7QEj5jypDxlylGtgimjwqMpx6RXaY+beFH6iG6m9GtiP62JPXV0JDTlhARMeUZuSqdsDsmQ587h
1BsItwU7ibpMR473+4MBxlzF+Fig96quZ9W+uWO1NjWzMj0z5Tkhx/
PyXVO+J9+3cdhIbT8ndZMYyOLXBOUh5geawC+Y6MFVglkf7UHBBf/fk55g0cc61rJX5B5K0ohiL9Qm
tDFFYjlZnuXjs055YMqJe1aXy871oR7Nk8tuK5G7lwmlC4afsLPd3BNPWOQrJB8bytDxMPsusGC064
ERiNeEogn2+UJ1iZ52Fp0Wb6qv6d0vEUmHeXLpqErOGTMQdZHduzXKjkC4g71tymiq9USXiLBXjvBC
3arcjlen+wm3W90bO8aQVUvYiXfuiqfsnVJaO7pXYlY8EtzP88OssZ2RFYlAvHLY/aY4vZS+3cw7vT
wW7Pv498K/4DLkZsZkrkJTS8e4Y07KXVVqHjZcxOKabi1sHuNGPaVuH7Y4dLZYpSfgkmFpbh8IixOR
yl30PmEm2cJz8jHZC0+Zj0WUqhYmD9fFqVA2BXYF7ZiXniKiw7ZdnDyY13akdznGnYNbmX0av0MQXU
tzU2esU7w3d53XxXC7P9Y+dDJizvEAWVdZVbOzZmtt0+bahouZq3Ufxab3wVBE87r1tG8Vw2OYJW0M
txVaKXc5AxnUzKRWDppqqe3xxxusbqKtI2y/
cjEkFe802iXBWfNYf3zZtHHj5szPHlmf1AYeBEJ+irukdOTnkQmw/XRSTC8xzK0RsJGx0REPXGXZ1x
DiTEnY6hrRBpaf6jp8igt5USCeWaQCf3v7MIBLG81ULGgLRuI0YxypmjqjURZAXH/
bmJ5Nnzmg2fm28GNcaBlXd0csEkpj/uyx6ynr5j2NZ7bRha08RSZmBIyRaY4ufyycrPjGWITVpo0sr
7mJS4lIXaSLUOlXV8wbJQlGja8zfWwWzMjeV9Uef6zJurLTCrdZtpPnnwKlh37yU9+aXf7gPt315ph
l2T/7fXThKfmWQDygJTLG1gm+TPCgnXCFiUgSwDP9kcSpa/fE0twVNcmTU4y5V/
nxrMf6SMyqCVp6J6UjirQuU1PWK32dRFY3dxlLxFsCCtIuDYE/ZpNMTP/
MmblvCDadVn8Y47g5Jj5Tac65qSI7zjm/Ggz5JffHEEcb4Uxwcemo3z8uHm4Y0SqVhdtfNUqRbB+zb
nIVYia64AOQD48enDkq0V8w7Pe1uI5vJwTX45N83sBZK/JIC4wvOw4pKz+GvLKFx5B/
hEt5+JTN5uDTx+cquLAaxbgQn+aKN8mGG/EZwB6pGrFHqiiP45twM99Uk/881Y6jyPnlvXDUlxT2wW
hYdBIbFvXCeRBVJ+BqPQ53SVEvTC6ZvRjXh+KFvRjfiwk+h4d/
Syb2oqQXk3yOE5jcuug4ppRM7cU0j4OPekqdvsLBp2dFQdlR5HN0hq/
QU9iLGT5jiuMgJk8pOIjxHiP/WZzZh7MOYFZa5dlUp4Tn9IN3wV7MTOrOX+FUvmyWvWmWWWRZmKH0G
Kl1j9GP2baIkjnJPfRjbh58Lo+rD/NaaFfeQo+rH/
MZG5fHcDyL0tb8J7GgqQ9lLWk95SuctjU+V5nHdRwLfe4pdFCJmj/O4+7HIsEBHNURb9y+Ik9RPyp0
aZmOzhU8jyU+M23oUt2bmbuxMo95HOcdGvxRmmiZEhlpopLlOnUN5zE8rl6sSJuUMeTMEzi/9UlcUL
KSu/G15ntc3Myq41g9UOZx5/fjwjw16DjWKMdFrKB7h1g+MZwli+MJppHDTr87MJfPtShANYpQg6m4
GAuxHstRiypswBbUYwcasAuNCOJSxNGEz6EZ38JWPMrsfgbb8QtS/
AmX4V1cLoXYKdNxhcxGm5ShXZaiQ9Zhj1jYKzdgn9yMoNyCkBxDWJ5FRF5AVF5CTN5El/wZPXl5uDp
vBqtHS+CLLIblrLjPUqODGipwCyvORVmz8HmOTE34dIFwdCu+wKIoyJtFzi+xMIryzsSXyZuPHfITf
AW3UcouavwqRwW04wS+hq+jEHE5ijspz8Cjcg29fhuctP0DHLS1LcU7uAvfgBt3U40JSa3cw5V7uVK
Dog+wxcB9Bu4fxFI4DXzTwAPJBREDDxp4CDj7fUyS99DxAWZxbuDh95Xy0LvwvYfLimfToRoSwbdp+
CN0bb7WtfyUMxe/HD6BSmbT2vryVBofwMHyPlQ1iM+xqB/VzNUCT0EqV+frKJWrLNIkR80BFpWnsB/
r8tByaPBlosNWj+MkppOd59VD2EAJDk9BL2oPYU16cbmKJUAUDPgMcszO4Zjic5LDaXO4koseY2DhE
3TuNlyBNqbRDKZMd+qtcHgrnXs7590M8A2ZJFzM1APRzEk8m8SAzWB4VzKYNQxjAwN3Kbm6uX4D348
waJocy6hlJabgMINUiOn88hgetzHzcAYzD9sh5EkCD9ohzLNDOJXvJP13SJ/8dg/
XNJg7UDSI2Si0Y6hRMvCEgSMMFFD2LvIu5aRoPR/6f5DGOodRAvoQJSsYVOzPfKf0dIwb8aSN/0/x/
1EcS7WKP3KmvWAtW8WGA3AcKS+5pB91eagvqe9DQ8PCRX3Y2HIIcxrS0HKpAmUGWTalgb2JBmAizmI
PaUx5eJ6dRQdo3F3c8d38eh+m4X7SHCLVAyyyh1BJ+9KenUaf9uI4/TOT3/vQb3t2bcaza/

G07dmJWIITHOWliyPF+Qzp1Z/TkO8c1GfaDQ+pb571kuE5DKQ2fglneXxPKtnch+YnsaUXLXXlJVt7
0Xo40yfdNslh6njMtnJqkiVlkcmdPc8M4YlTf1BOdkWs4+6Vqrj8BLa10q3b+7Cj5cgwmU+xc/44S2
ZxZpfF+J69y6G4if4qneq513JtAmt0C0NwmYBRuby1vJ/4R79d4XOI1lMv/FqC/
cQdev8VHbVpYbaznDxGLyw7wDY/i8nJIurIY9Kye+5uTUd5j91ASJuM73EEBnS69wAKDw0+7HH2E1b
BIg/2IdTCguS3sM8lbCMuj5PrpX2IbPW4TqLZ4z7Jjs5OEz2E9Wxubk+RXbyrU2vLtOGRr2jAZyrDz
GyGyT42SM84m8Fpr3nMgUODcxUSmj0FJ5NnhStVcqGnQJusLTm5tky9QIcUDviKlWFmNsNk33gyjE9
J1jVP8cARRudxxqaXaHEna/
TF1PuXjMVreB1vcP4iXta5OGWcTODcgw9lTuq9iOsrZY1UYr3MkQqdp2phC1ED+Bkz5iVS/pz5/
UuUUdIKvELeX7HBvYoYfs3Iv8aT1m+JWa9T8++p+w1qfJ3Ub+A9vIkP8Ra1vUTJb0kz/
mBn0A7W2e3owAs4ydEKvn+IH1HTeu7kx5TopLbfUMpPWCVT5SL8FH+lNS9b0jnH0c/
snNPRS3ZlGbIRf81RfqbGkjJ+ThlaY+9g8iCXjEyJ2f9+YcMWvfMBCFp/
wzX3IC5A0ahUSQIDL2fA6+IsjBv30TwZPoeC3limpAE0W/j4kbRI/ckV6sgmzAFTJ/6WoUtiSRnBS/
+UlMR4/om35pckeP7pfAr7VazYCVBoH6PPo8xf2VDwd2wuehDPY9tK/rmdTr0Drv8DUEsDBBQAAAgI
AOdJqFTEPMC7TgIAACUEAAAlAAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL0xvZ2dlci5jbGFzc4
VS604TQRT+prddlqXAAlJBhcqtrcqqeG9jBAOGpBfDmjZCjJnWYVlsd5vtAvFRfAxqosREH8CHMp5t
F4pY4p85cy7zfed8Z379/v4TwApyCqKYHcAQkgO4iTkZ8xIWFCxiyfdTMtISMjJuKeTdVnAHyxJ0Bq
m8vrVWMtYZ2DZDLGfZlvecIZxKlxkiL50PgmE4b9mieNCoCvcNr9YpouWdGq+XuWv5fhCMeHtWiyGZ
d1xT501e2xN6gx8KWz9yebMpXD3vmKZws1Rq2bsOw0Qqv88PuV7ntqkbnmvZZtbnDTdapk/
yT5JhyPB47WOBN09Jj7hrM8i5Wj1oXTGcA7cmNiw/PdilXPaRVIxgVEUcwxLuqriH+5Tf2SxulN7NL
rQWCGWkR1iq7ouaJ2FFxQM8VPHIr47vVFa3ipvFV6cP0peOWvC9StcpcItq1UK5WHkfyK3iMZ5IeKr
iGWiqmf9I9ldrxqeWJxqkknPgkYZdlSxHf00SeSSU4A2CHOsTpgU3fW+XIddH+p38xfmz6UvQw8Klp
mKm8IR9yLDYb5H91jfai605Tl1wX5gmd1vizO37KbZpXUP0wwGGKX+FdIvQnTZKp0aeTpaRjWa+gR3
TJYQxOmOd4FWM06l2CzCBK2QHMIlE8NgI6hJthD5jvI2wFon8QPRtOGN8Razi44U7eHGywAzVJ+n93
DncBLFMkZ3u1LJJIrh2RvAiINDakDT5InS3VYUssEQTpc7BagGsjOtns853MlShKScYPIH6BaHjjja
9kafpvNHBnvkDUEsDBBQAAAgIAOdJqFTE58tbMAwAADcbAAAvAAAAb3JnL2FwYWNoZS9tYXZlbi93c
mFwcGVyL01hdmVuV3JhcHBlck1haW4uY2xhc3OtWAd8U8cd/s7ryc/PYGQMmClWsOUhAgkhhhA8BBj
Lo55xQnAf9rMRSHqOhg1dSXcSulKajrRN29CWtukgSSubuAnpStp0773bdO/
dZvS792RrWHLJrwF09+7uf//x/cfd8eiT9z8IYLvoUrEbZ1R48E7ZvEs275bNWQXvUVGEMw68V/
bvk83dKt6PD6j4ID4kac6p8OIe+XWvivvwYQUfURB3YEqFE9MKzsul+1VUYEbFEnxUwQMqKqW4+/
CgggsqVuOMgodUrJWTq/Ex+fVx2XxCDj+p4FMOPFyCR/BpFZ/Bo/
Lrs7L5nGzuduDzDnxBxQC+KMdfUvBlFV/BV1V8DV9X8Q18U8W38G0F31HwXRWX43sOfF/FD/
BDuedHxfgxfuLATxX8TI5/LikeU/EL/
NKBX0lGv1bwGwd+q6IJjyn4nbT2jGx+78Dtsv+DA39U8Sf8WcVf8FcFf1PRjr8r+IcD/5TM/uXAvyV
YpPuP7B8vJh5PKHhSCnnKIeAQQmBRV2f7UFd3Z5e3u7fV2yPg9B3TJ3RPQA+NeXqiYX9obJdAabMZi
kT1ULRfD8QMgeUt3n2Nfb7eofbGfm/HUF+Pt3voQGe7V2CpvT3kNz2j/
oDh6dKjR8lgdQbhrMjBoTbvIPllLns7+u0Vrb2/
Y2Co39vd1NnjdeAIdbFmJGlHYzunRmanuhp7egY6u1scGJ3d1+3t6uzr9jlwVKBotz/
kj+4RyK+q7hcoaDZHaMlinz9kdMSCR4xwr34kYEgAzGE90K+H/XKcmCyIHvVHBGp8ZnjMo4/
rw0cNT1CfMEKeybA+Pm6EPe1yNGAP2nV/iEYXBNkLLKu6bj6mlgp6eIxMy7MsC6gJxgf1ML00Hjb5H
fUbkX1+qY8SNs1oi18uJej6jXDEb1JcWeRkJGoEu+Z2UANbQCzqD3iS8xSyOLHZe8IYjkVNsnPntnA
gnVbq6D0xbIxHKTaiiDyBNeN6OGL0ZMjfFzaDjZalm7IikaJcuz5OtsXDZmiCew3q05xbn+GA35Mpq
9kMBvXQiHRq8ywTclwynJzvkjqS89aFOTdn7pCK9fjHQno0FqYHDl6ELbvnU8yf2UPG5frISKYpzIq
qbMkkI8edfSV77i1JmJUWEVU5iB0BRn/
UCqRFCQoj6unrbuWaoNlr0iZ7Toai+om5KJCVoieqDx+n7VbiyOIn8lkVZsWlx/hiC/

ZUvZSJuTAOGxEzFh42GiOkN/SgQIXNgxq3hsZjUXt6V9p8Z1IXRRSwKLJysuSxKlI1S1ofXXnADBqK
KKS1u+lquyqoPZY0O70qMrO5XkrQcAzHNcRwSsOtuEXDpGxO4DRdlWldU8wfGJFhVtFoRZjL4uhKsH
RpokgomnDgtCKKNaGKEkVomigVizTciJs0sViUaXg5blbEEk04RbnAhpzh2iprcyAgpblzErUYo3os
EG0xJ0MBU7dUKwhOhCZplFgqcEnOjTIwGiMRI0h/hjXcKa0/
hpvlvgqBupz7mlihIlEOJH6MCpmKmlgmlgs0/t85LVD/9LJXEytEpSZWilWaWC3WaGKtWMeQSEtXBp
3FqT5ZbRklmnCJ9YrYoImNYpMmNtN+cQlO8UA5xKgIbxIoSdmgiS2iShPVYr1A9UUfFopwa6JG1Cqi
ThP1wqOIrZq4VGxTxHZNXCYuF1i1QNoJrEyGX3csFPUHjblF6acd9LVMcuaiJq7AaU3sFFcqokETu8
RugT3NeihkRl0jBpENEjLXcECPRMbpeNeoGXYltHbxKHKNspi7hnl0HtEjhmvL5siWesKWFN955Jgx
HNXEVWKPlMywvloTe0WjgNfT7u1trGvt2JcAg+jU2+jU27DPomON6mZH42YwDd8m0SywLimxlZE/
pgcYYVEjBZKuZ0ScS+IyasZCIzRzabZTVBMtwquJfWK/
wPUdpitRwFyRcWPYP+o3Rlz+kOsZ0UURBzTRKg6ysGmiDcd5bCRR6D0aNidlydWET/
q7PEtJ5AHcbMYCI650Z8+6N6F5vfRb+2xxro8xeeqPsmByJuOWpokOsV/
DYQylhYBdAKXXO3lkpuzPrw9uS1M6RTMtNb5T1U9Wel7N5h9ZTEOpfYi3wLqqnAfy/MIsj7qoaU8xn
HJmqs8cG7NqpT80arKaZ5PRn26+Vb5YW8aMaOrRtqKqOtc1rIg2NgYCNLAq4zIkeVdd7I2MMpiuA5m
nvX2k7c1xZXga170tWYzPDkdXVW6+yRNoV26itEPHYqkYliK0xJ/
t5pWb1dzpuABNtrPKkiqvoqP+MevGt30Bo7LfGCWHcso2J/
tCx0O0u9O+LAtsq1oA+VzXz0Lres1Kmv3quSA/i8lIClfyUxL3bIGW/2Ha/
N2piWXXfMlQ3tkCE1SxKncuZt44t+aUnHkROEDh1i2nnKk1/7q8eaFLcerzQiamHjZCNLyE3zI/OnR
ZoLSwMR7Q5SUwHOHizouL92w320UJJHr8RwJWhVmcJGqWh6ttBNWPEjrGRItpPxY3zFaJCKM97I+e9
GTSkPvahSns2iPft/adls/veVyTq+S3IteaDZBv7klQNvdssAq1b1d64fYxRqOmVcAzKOXzYUUqnHP
nlZ1m0qEMAOmEIuOGmB6IZNTaRIxVXztbay2HWk91+Tzv7uzsnfu/i5QlWVxZE4M63dlWNX95vu+uy
yI0m4fL58cadc6nIfKdm257jsiXEdA975VTnTt15r18CmQhTSKbsS6RLUmeQid5y7+oI1K+GZNI2SB
ErJi+IeYPGx1mqCMmT6udWRy0MP+5SlE4HDBlKSuVD9/
Y+DjThQUm+bDNkXLpUVNE24zQhMCuLLtyv87T3YD18GA3AAdW4jocAq9xHOVhL8e82vC7Es9Ome/
mWE8Z93J8JGU8zPFIytjgeDRlPMbx0ZSxHyq/+bJkG+CMh71gX+iegrjHIgmyLbImqxFiq9kEMDHOv
hg3IEwqbhZ5NEPhXGAaeb6aaeS3synomEZhQ8EMiganoDgdcRRXFrCJQ51GyTS0hsLKQvc0SuNYVDe
NxbXTKGsoqixyz2DJ4Aycg87yyoIpLJ1BxeA0lk1h+RRWzKCS3FbGsSqp4w6Usr0aBQSvGI1YhGaUo
wVu7KNZ+7ETrYS6DU3w4Rq0EwYfze60bDpg640IouwLuBYj+CwNBHSCX3nkdpgv7iHkk+cgn92HSOX
BVTiJ5xCLJkp5Lp5HnJ5v7Rd4AYHlW9pGhtzy+RfYMoPVVHyNbwZr2a9rr6mNw8Xf+oGaWnccG+LY+
BA2nZszahmZAz0Ethdl6EMF+hkZA5bSLptlQmkH1uGFeBFFl2E5XoyXkMNLLT4voyp8zCc8PMZd0oG
bCT3xdG4+j0vyMI0txJ9T7rSJJLprLYQOUcz1xPkQUTjMUBri/GFsYogmI2OzhRModBK3JITuSoTVE
vd5VAkkm6SldoiNpDBaYjlBMrp1jlGzBS6zwk1O1ZKJm5W3xkkI686jPm8eQ38Kw8o5hjGcSjCM06I
89l6nJ46tcVwax7Y4tredRSmddPlgzRR2XHBeQd/sjOPKO7BKTt6LBueuggewezDf3TONq6aw5wJD+
OpzlpjFcOIyopyELcy5Sc5GCdsEVjN8qhg2exkks35czN8r8EqOyhl0r8KryWk1E86eew1/pcjvVnD
b4yhW8NqN1PoETts2iLsIioMkp5we5944GtuEz/0ISmfQNOhsnkLLhRl4GW372mWI7a91HoijtaPOe
XAabQN1DQXuh7Gm5mFUcNF3Fs6GwprKwjg6zkKRM5UF59prfbUXGooy6ZSaSiWFruhCm8TG2cXS8aw
LtHsP08LLRFOZmgOJ/hDVHGW9CXI8wNCRYzC2b2QiDrCVvY1cA1MOjGOVUbyI8buCMeBiJGym62qJy
3Yis4NI9bIfIN0hft9I3G7CbRaqboLiwhV4HZHO444a3E6afAI1idfjDaS4CSXc88YEvtdDfQIrFLx
JwR1Pkb7Q/
lTwZgVv4T8IjtT8p8gsz16yZlos8jIoiTn+mV2yiSG/30oz7pyLuQCnZaBsdHZPo6eNvip29k6jz/q
6F/1nUeLOfwAD07hGhrONh5P2gHoXUWOVEsrIdDa0i7CG7A8lI+VxlFHl4v0K3kaD355S3vMswJc7W
Ut7EjKc11oZfx/6ZbqLlOSpZ/sOK6Hu+i9QSwMEFAAACAgA50moVI30oDKjAQAAGQMAAD4AAABvcmc
vYXBhY2hlL21hdmVuL3dyYXBwZXIvUGF0aEFzc2VtYmxlciRMb2NhbERpc3RyaWJ1dGlvbi5jbGFzc
51S3UobQRg9s7vJNunaxPWvtloVRGJSuhfeiJaCRIRCaAUloHeTZEwmbHbD7CZe9rpvU8EgeOED9KG

K30xCqXUvSm++vznnfGdm9+ev+wcAe9goIoeVAhy81mHVxVsXawxuRybppRwyLDb6fMyDSMbBlQxFc
MrT3uEMcCwVQ/6jjGT6iaFWyYJmzXabDE497giGUkNG4sto0BLqnLdCmviNuM3DJldS97Ohk/ZkwjB
vzo5ptZKtUSrjiMH7HEVC1UOeJIIgB41YdQM+5O2eCAZ8LKLgWvHhUCiz+4hQA5JU28+k6FZ+V6R/
jswFlyu72W9QJDS90YnUDotn8Ui1xbTxn6z6oNkeClj38AKuhzxcF+8Y9v/
XKkPZOAp51A2+tvqinTLYFf2uO/+miS363Dn6Byz42g9VvjZH2QYjr0WKL6m7IIRNuVS9A6vWJrCq7
yewbwzVMzSHYkIxJdoIrzDGHE02pzTq5wFTaXlmKr3QotrHwmxNQFmf5aq3sH78Fs+b4Tcj6E0BM0H
6MzPJ9t/k7xlkC0smLqNMWV/VwRvyVXgEUEsDBBQAAAgIAOdJqFRxFgnYDgYAAOYMAAAsAAAAb3JnL
2FwYWNoZS9tYXZlbi93cmFwcGVyL1BhdGhBc3NlbWJsZXIuY2xhc3OdVmlXE2cUfoaETBgGkIihKLV
oXUICpNZWLahFNglCQMJStJROkgFGkpk4M0Hsvlu71652sXv9Ss9pidTT9nv/Sv9CT0/
vOzOGEEIP9kPe9b7P+9zn3vdO/vzn198BHMZNAc2YrMAuPCZgCucEGp1nzeOsmWZrT7DRDBs9WQkJc
TZKsCbJpjKPWQHVmGNW8zwUARewICCFtAAVmhcZAbW4yBqdLRvslMmaLI9FLy6xnSUvLnvxlICn8Qy
PZ3k8J2AvJr14nvUvePEi67MC9uAlHi9zqB3UElKqRzFMXYlnTUVTOYgRVZX17pRkGLLBwT90aqI3O
jMe6x2d6R8e6p2JjY1Goqc5+AYvSItSOCWpc+EYnVfnOjhUdWuqYUqqOSGlsjKH6pHR4YHe7rH8qaq
0tCir44as92tpMqizUVRFC88qKTk8IpnzhOM5rqiKeZKDK9A8wcHdrSXJuGZQUeVoNh2X9TEpnpIZC
ebAhKQrbO4sus15hZgHBjV9LixlpMS8HLauDV/SpUxG1q1bTpF/abLX6br6QCka7OaaOdlcr8/
Cf+BO2j2JMKvMZXWJneho3iKPfRuCwQRNFGJxCN/l5Ry8ccmQoxIT25skcHtYqWsaOabbM55t0MwZn
VMyHCYDGwN8156XDC6f1VP9kjFPI1LXHu29EwHZDI+PRu4cXZ9d7gwBUFKtN6Ug6XJaW5R7l0xZNSy
ZDpQgXxpStQRwJTWTAxchwWOmlFgYkjJWMtFLJK2IZVdeREuhcT3FQXDWLeECm99Y7L+bRYSOx7Ssn
pD7FCuR16VCGzsn4kEcFvEADvF4RUQ7XhUxgj4REQyI6ES3iNdwRcQoYiJex1Ueb4h4E2+JeJstv4O
rbBrjcOz/5h9j8K6IXvRx2FbsHY/3RLyPazw+EPEhPuLxsYhP8CnJ2CbiOj4T8Tm+EPElI3ODkfkKV
ygyWyNDQS0qPCK+xjckv1NRKJuzVEXakorO41sR32GAw841kqNZ1VTSlBMJOWM/
nfpiD7qySirJrvKwKLY3ifgeP1C4mxSjKasuqNolS4Ef1zk/HL8gJyhXWu/
qKRBEoLk4bbcX1ZYuKy3qAiUTtdh4xHoK/sAmOebbuEra6bKhpRbpkmDpglcai+U5lQSbnjOxAUMlY
Tarpds3rlOZdhEgh44Sz+f8ll+UWKgsZcY81RT7k0HfD1qoXQOKqKY8x4JeaWr98pKNy2FHIFJS9kr
6DpoRNSkvDc8yqxKUCL/CyMYNB8kfiJSGYsXOlo0VFPbsnapsandYeOSLWSllFF1kp1xH87n1r/CyY
cppG2xE1yjjzMuEwHJPTVJ+bqkCOk+gYxPXJuifwi76awO40IAWtIJDG83K0EjzcMF8N/2oUtHYS2t
UuKh9yPpvRPWA+vJgDtxPlunD1ArUAwfgxkEcoZFoG+EojlHPAB5xAPrIktkKDCAY+gVlayjVRAsIE
koIFUSPIfltaweJjRgtRqIdHQ7mX3SCp346GFqBKwf3ULAlh/JosDWUg6fdzZb5HLzUVaxCKEOD2+r
ay9lOpb0jFu60rKKK9j23UT3VUN7gyaFmOU+zDZXUHiGKR1FDtOrJuSaiEyRCh3CcRifQg5MYwKM4S
4WdudFvE8y7MU12J4h+BcbIspWQaxAl+1aSoJ5E6iQkN/1acIracsIR0UUjD1nSp8JxPE47biZSsMV
yfFs05P4NtVOu1tgt+JaLwtNLt/VZbIL2qTwbv3UzZ40YrzKyrCIfGJsy9r1w7jtOPUNiem2/
hbq1GzzW6TMFIfPm0b04Tf6zk/Slc5DmCZkF2x/y7ViB/0TjDYghV+MK6m/
CHVq2MnSN+Qipd9bCbrJPFTA/YzHnSb9BDJF11D71N+q4Th7DtDKS5z/gJJ+f+N/
jIqVIpVU0cFjFTg7F3kwUeFN441nHG/pQO7jXSE9m1eNrJOR7r8Mb/
Blly77d9myb775baHLuW76NPVO3sXcqh/t9+1awn2z2+w5Qs4KDOQT+yPvup+gC5ykW0/
Q0ZigvJOyDXMCqJ8+qh/KI6cC8Jy7VO2h/3PJngh4TU4T+Z9HbD6LiX1BLAwQUAAAICADnSahU6+BF
9VQGAADfDAAANgAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9TeXN0ZW1Qcm9wZXJ0aWVzSGFuZG
xlci5jbGFzc5VXWXdTVRT+bjPcNL2FDrQQ2kqRKR3DqJgUEEprKx1CU4sFFG/TQ3Lb9Cbe3NDWCQXn
eVZQwLmOS9C1ArUL8YkHX3j00bV88Se4FqJ1n5tmTln4kHPu2fvb5+yz93f2Ofn1359+BrAF39nhxJ
CIA3YU8f5+O4ZxUMQhO6x8fJiPH+CDB+04godEyHZIGBHhFzEqgtlRhiEbjvI+wJsgbxQ7lmKMm42X
IISJEqgI8yYiQivGBkRLUAOdL07QGI7ZMYkp3kzb8Qge5drHSvAwHrfhCQ44LuJJEU8JWOob9nkH+r
1HvLsHBzsG+gQ4esbkY7Irpishl8YCbMrllXWdaapHgLVNURV9pwCTs2FIgLk9PMpojh5FZX2xiRGm
DcojIZJU9IT9cmhI1hQ+XhCa9aASFbC5J6wFXHJE9geZa0I+xlTXpCZHIkxz+aajOpvwamEa6AqLds
nqaIhptHBlgOm5WgHrnAlfVSXsOqqEGPc06GnI2ECvHCFrm6L6dI3JEwKqEkoy6FYjMT0hJojAsnT9

HVN+FtGVMN+1aZxN8z0Z2pCsBlxkpqgBUokTsk770AqFrTeh4uszVdemfUwXYOngnwKkblVlWntIjk
b5TqqyfV5roMhySSS13U6FB3FZoR0LKFnATZMxT0huBOyRjLBVZ6jT4fRwRVbapiPJ1HkKudeWEZD+
kTHm1z35kp007bZs67b8QOZLuF2pT5f942RheCHihIiTdJREPE3Hxjgxzwgo9ikBVdZjGrm571b4cM
vL29r8oQW6233hmOZniRTULkLTVj6LhCY0CyhPL9klR4NGVlZk+0aHZrzfoJiIZyU8h+cpu4USk2fZ
H2FqwlLCC3hRwkt4mYIi4RW+dGV6M4NBLTzJYyfhVbyWVGXzW8DKtMFATNWVCZahXJ4bmj0xJTTKCb
+mQ9PCWv1kkKn1obA8Srr6NMvquac7JLyON3jzpoS38DYPzjsS2vCuiPckvI9TlEIJp/
GBhA9xhjKc9DGbaRIa0CjhLIeck+DGRyI+lvAJt/
sUn4n4XMIXmJHwJc5I+ApfCyiLppJ0uNXZ2tgg4Rt8K2Dj/y0+NFUuqYmZWU5SbcinXWa005UmCTVM
u6mqynpYS0JT9pxnlPZSJTrAArGQrCWIt7kgvQ/1LMYrT8NBqiAqm8xyoPOWZklzLHl+8kqmmWedyO
ksqOf3g8UfCkfJ8VJ5dNQXi0Q0RtUuwySbpIaJlSdDJUyLM/9UNuSJFtjoycEvFJ+b4W16OCGiU+fM
BxKisYAHi/pd7swsM1TqPVksMWq/TUklvDoLniQCv1CCcrSPTenGFUvpM6vGINvFhe1RtOha3Mcvp+
WL3NkCtmTuoj0oaz72cIypfpZVF3NvrLz5epP3nPmowtNjCWjhGDG/ytldMHhluTJyNsTUAD8YtLNu
igb5PiSHYsQPE9FGwPYCGbx5TlNxEP3hiYhxRm5CmwIvGqymt5OTXm5UBXmNoS8TfVMJp7aFRi7qBe
otjRchXKCPIrRSazWEa0kNer0ZAGzEJuqLsZmegUXcuGgSZthINj+HouGLMPU0mi7DfAmW07A0nZ+D
lYRiLxfaLqG4T3Cbm1visLdchddhvopqGpTMYK/b4jA7LHGUzmB7QrSJRBY3Ca+4rYRelYGudIuEFg
20yEUO65UZ1PfNYenwHMpoxfKKijgqG+NYFkdVy0VUX2mOY/ksVgjoa5mFQ8Ap42OlgF9Q4zb/
gFqHeRZ1AuK4LY5VfO046k9hvcNijmO12+qwxnH7WVQ3OawcucZAzmKtCQdm5r9vOk/huhMe7ASv46
3Yv9AfpNAwKPSelUh22BjX4ThOYAkFjod5AMuobaIgNqOSJFUU5DoKcwOFuBVbyWoTWW0lm+1ktQ0n
aZ1zdD1cpivmGnbgN+zC77gbf2IPrqMD/6DTSNkAzbieHuJbycZEVrW4gyxNhK2hme4if6/
Rqm7y2UJz7aDZdpFHlEiaazf1NvxFM7ZT4uvwB/bSzEU0M1BNfQJ/
j5F6juoiVDfpxrDkBqpE3DtPhLOI2EefInpE9Irog0BNSSc1XfSbRznEXEQKZZqnVUwpNbBHRH/
tDRSL8P4N73XS7jeIynfpSzF5tUFTImjFuktY/yNqLxi0T7N5A5kMGob3UaoAO0lr6C/DDmvxf1BLA
wQUAAAICADnSahUO0NXZuEDAADXCQAAMwAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9XcmFwcGV
yQ29uZmlndXJhdGlvbi5jbGFzc5WUbVcbRRTH/
wshCWHJY0uLWLBVa0gfVq2KWkQ3JJRISGIebXwRB9iGxbCbk93Awc/kC/XIsccX/
QB+KI93djeBLJMXfZGZuXfuvfOb/53Nv//98wbAM/
wUQQLfhvBdGGoEWWxHkEN+HjPYCeMFn3dDKITxPV/uzaOIfb4q8aHMhwoffgihKiGhFlvqy1qnUaqo
23udfKkpIVk8YWdM6TGjq9TsgW50n0tY3DYNy2aG3WS9oSYh5SXmyq1Ssazm3FSZ9c7ZhdUw+uzwFw
lSW0LUdeXMc6NnsiOKOdItqnowtHXToH33NEOzlUa1QCfFr+9nmaX5XBVmH0u45aXppvJK72kK91Jy
6Fe97+bwlRsa3NQN3d6SMJteJ8bAtnlE+7Gibmil4emBNqizg57G720esl6TDXRue86AfaxbEpSiOe
gqjK51rCmn7EwzlPMB6/
e1gdJyZ9Lnld4dDhhn5NfQLdV3czqfBElYmu3fCaTbHC06yhkJGBvHjj1dzc5NCBhPr/
sl5GmTQYn0ZAw/LeUr5ep2a1Rusv8pSxR8O30zVlTabcTSmNTXNX9xN/xOWhTNy0eofHvU6Ih13XB3
3ALejmfUzOHgUNvReVeXRU17yk+T8R7uS3jyVu2WsLKvNvOlTquqVir5amfiqwqhJqOORghNGS38KG
MJdyTcE6aMvicZd3nQw6kc/FaqZWmn9EwH1HK3WqNGpXbL+3kZK0iF8FJGG/
dkvMuHVaRkrPHVMt6hh3PVuvLBiXZoT7hqF5atndLXQ4pqxhmhiHoteiqJK1/
WNHsaI33kPhtY2tgUvhv6Nh5MvW6B//v0nKsu5/I7aqNY7+QKtXq1kG3UC+VSp6LWd3Gf/
tgS9Cc5TzdMIgV6zmTNYJbs29fsEP2oCc76rjeTJs5MujkzSebMq5695tgxqkEvhMYHZF2QX6J5I/M
3pExy9jUCrzF3iWAmGRqtw5lk5BILmT8hX2LRMaKuEfvDKfw+jauEBDoyQGXj+IBWaVplsI7HUPCUP
IDsHoUP8ZBmCR9RjIuh0Mz35jJ/
IfzbuGjQcX56LXlunLxOxd3kLYqe4dGZFYK9Qoo43g3K+dKpsORGeRX4iivHD34kBAn6QTaFII/
FIEE/iEo52SkgSx7IE1LqJkj8dx/IjhBEwcc3QB5dIu4H2aOc4hQQ/
ob4wZ+Q6jdBFvwgFSHIM3wmAlnwgzQopzkFZMV57BI+xxcCkEU/
SFsIskGdF4As+kE6lPPzFBD+FfGDvxIqEvWDHAlBvhYrEvWD6JRzMgVk1VPkuVCRmB/EEIJsihWJ+U

```
EsyrGngKx5inzj5Gz9D1BLAwQUAAAICADnSahU0LhLUrgJAAC9FQAALgAAAG9yZy9hcGFjaGUvbWF2
ZW4vd3JhcHBlci9XcmFwcGVyRXhlY3V0b3IuY2xhc3OVWAl4HGUZfifZzWy204NN0xJK7UJLu9kkXd
sCQovQK6WBNInZpJCWWqebaTJ0s7PuzjYNh7eieCAqQhUEQawHaos06WGpZ1FUFG/F+77vWx/
r+81M9soEypMn81/f93739//J4/879iiAVfiPileH8JoZuAWvVfG6MAK4NQwFr1fxhjDeiDepuK0eb
8btIbxFxVvDiOJtYdTjjjA03Kri7WHMwa315LhTPnfJZ78wvkNQ3imfu4XknhDeJeO9snOfzN4dwv0
yPiCy36PiwTDeiwOy874Q3i/jB+TzwRAekvFD8vmwio+EcFAUOCRAD4exAh8VgEdkdlhm4yomQjgix
0dl8w4Vx0Sh42Lbx8I4gUdVnAzjUnxcID8RxifxqTA+jc8IwX3CdyqExxQ0bexI9vV2rO/v6+ju2tn
f27mzp7e7p723b0BBpPN6fa+eSOuZoUTSzpmZoTUKZm6wMnlbz9hb9XTBUHBOBcD6dcn2MoT52zp6d
ib7unvbq08q2XrW9W32Z6s6CWdzVtbI2aaRVzDP1a9gm+lET3GfOs4qUW0y01RyrkuZMa3Ebm4kenR
7mHR1KSuz2xxSkOi0ckMJPaunho3EiL7XyCRGc3qWGIlr3HGDQ1nI6bZpZcjasNvKUeb1RsreaOb4t
XJjCtbG/AQ1PyN6+z4jVSAEgQMbrEFqPLvTzBhdhZFdRq5P3yU2RDqtlJ7equdMWXub4g9PBwVnUyc
PsKfKATOTtp7as0XPenx1l5kZ075cQZuvxv6Obd6qQCFzY1letO9LGVnPJwF72GRY4mduLutSxWdZk
kx5xjObM7J6ztho5pltuwoC258zFcyJNXtaGnaiv7dDQpe3CrkUlZlVecK6oFeKWuVVfI7W5wx90EG
1+nNp8qQtfbCnLJNCZoYJbugjRePojo5MtmC724R9nPWk4vOMzJBhl2tI9bhTkR8KVsSeOejVKaUaj
ltokxnbPrX0psfrkIJMp43c09CstyybOuvZLbrYqudskktAA3puiB5o8JFIlUwXWsGSMxHPQtvlI+d
p68tXMQX1DtVma4TumEH/etFijS2NTdW02U95zesCY126oFziw3emSIPGbr2Qnmx5wb0yspkzW5ixV
s7eYubzJC0p2einJL0dTjpp63WlqnJYLiwKYmdaPvRF5bmzavNWy0tNkDWm4Qv4IkX6FbaGrXhCPte
wh1R2A3ahPd1OJan4koYv40m27pJlvYWMbY4YxXJT8RUNX8XXFLR6ukZLWkQFMbrsgvyy6KDFZcayo
8Y+VtJyllAJs3uXNDQNX8c3RKdXsEc9q0rS0IktGjajQ8NVuFrDN/
GkhmFco+FF0DV8C9+mvMGyCl6v5w0N38FTGm7AzRq+K0IrKKQpavgevq/hB3iqQl03uCp+qOFH+LGG
n+CnTJgbzGySETJc6J8J9M8FtXjgIv5CEH8pHA0+LZVe3GAV0oOOp6RlRUd9nJqzRhynLhdv/
UrDkBj5a5H4G0H/rcx+B+aVVt4o2ccGK9psWoy/
cUoCdGeNjJsAGn6PP2j4I/7EZqjhz3iiQuu+4Zw1KpeLhr/grxoy0FX8TcPf8Q8V/9TwL/xbw8vFB+
u6rKhTQdFR0x6O7jHG3KTIZ42Uuds0BqNmxtfSyfRhvkSmXloKFpQbmBzL2Pq+Ml82+PT18t3uMlo1
Z+St9F7WaGz6blP9mqiNSYU3VJ5JpbO91jOE7ZLrnK/0vXK3d05XeWuat01mnFO4zjuAqgV6u7v7iq
+0siO5HvkUGNFtBVfHph5Ptai8+bv1598I6wzPhtn56kvwrFjlRez4ouqqlGpgD4r5Yjfk/YirIdxI
z4tNE4OQbbmAUwIhBFS8lngK1vhE1ecCnEZItaauTvP9337S9Clzm5l1DQrnyxfuiQvgnbiL+NPfVs
VqcwTUEybJ7ijXnMx7+IjK0MzG4n7Sra0UT7gftC3nXTUrY4xWVMOmM0rNUk+Y9FDVa4m5Ke2KzSTm
ey46B1NpS1wwUx8cTBayfPrl80YZi4+Z5z/
zE4ReTFGE7bwhFbTHnuUrbJqIT38D+T1faFzedhwd933I+aYJzuPfZBr/
XA2hCeuwnqsNXNVgJdcby9ZN/GkvWy/melPZeiHXV5atlyIilyHnEbkPnZE3JEeNf2F2oZuUPVytQ5
AzYP5xKAPxSM1R1NbgOAIDEwhOoO6gA/YCfuscstXodSAcFiQd4Hr0FeEyNETOVsVrT0A9gtB+LDiO
+oGHEY7MCJyANlAbTx7BzAnMOulIrBRV64iaxRG4AjOo3WJ6oSRylWdLv0OrLObA94srWxHZAe4ci0
9gNoHnEPisw4jEWw+jId5yGHNbXKUa9+OultYjmBd/hKcTmD+Os51ppEmGcZwzgQXjONfdW+juPeco
FingATGiR3DeOM53zxe750scngvcvaXu3rIpPLEDWNg16ZC465AWcUjbBFpOHqIlS3A37kWz54ouNP
C7mXZdjNkM51w6oIkuWEKPN9PqC0m1mQ5K8mc7x2HG4ia65xb65TaOd2MH0a7FfdiGo9jpuDIu/
wsgyrUYcBx9jGfbAWcm7lWcmSRPjePoeag5TZBaFdep2KHihVBU7GziKV9W/ErcbY41HJfSna2dLeN
oewwL6IS5R8F7m+uEk1hH8VwFB1tKoY44ERtkEhqQ59oSpB0d57lonmZBlskupKhZv1Mqyn+xRCWXw
VWAu7tJPVTU5RJnj2l0HCuY0isnsGoCF1Zncr4srWZ5ckp4YXk6unjKMo4qT3ZITl90BBdvUbpaWsf
xvNZTuLrtFBo5v+QA1q4OtDUFxrH6AC5yd5avDjQFupoCJ1cHSbqwRBpZXdfWVOeQqrLTFGTogzgb5
2IRLmXez8EKb7yUgjewsq/ibAXWOGvXdefQEGCMVDeS7iaejvH0JlyHmx3ToqSrZ1qYuN5x5g6vdmQ
mwXWdeRVCpyk04EZWxR6VEWB80+FN/FzJ39NEr6s4LpLUnpawe0lRct8If/
kQ9FrCSo7i5JBbE2uqA3FLWSBCXiAUWMh6/AmPPyj81cy3lzEHi8wvRs7LhTSdEJQ8a3WlX8YgtUiU
nn+oiOS2mzsZ87twFvY7iBe6XMXKiDBfbEdKBAXsdRwawSj2kTdMtcfo2kDRAc0857PaM2ANaYRejb
```

co7AXVJtxTlvBqUaDKSEo3F6CbPaBRAkm5NNKWBsnAy7tYZqF4ywSuaKuuqfup1wO07UF2kAPFuidz
UUSjJ0JmL+Gshhwz8FLOakt11qDiZTzhE95TIsl9UXXRZBdbG/
S62EBACn6yu1dZ+VCZlYuKKizyVFDwSof+Vf8HUEsDBBQAAAgIAOdJqFT2D9UU9wIAAFQIAAA/AAAA
b3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9BYnN0cmFjdENvbW1hbmRMaW5lQ29udmVydGVyLm
NsYXNznVVtUxJRFH6uLC9ulGaklL2I9gIrsJhZKcTkWM44Q+aMjDN967JuuA4szLJo/
ol+i33BmZrpB/Sjms69S4RACn7gnrNnzznPc5+95/Lr9/efAJaRV6EiHYSuYgxpFePICG9JeM/Esiy
W50GsBPEiiJcMgZxlW26ewRdP7DEoG7V9k2GiYNnmdrNaMp0iL1UoMlWoGbyyxx1LPLeDintgNRiyh
ZpT1nmdGwemXuVHpq0fO7xeNx3dqFj6eqnhOtxwN2rVKrf3ReuNmn1kOq7pZBmmz3c+qf/t/
ubqbXPFYjZPvYOGF2GIxwuH/IjrFW6X9S1KERjZRFfwQ+nQNFwqUrhTpl1FBhWQYHXuNEyHIXMxvS5
aO7KCagdxyHXFdl3HssuSufrui2HWXatmN4J4xTC+a5Vt7jYdEiY1dJ8E6cDwNn4xVclvv4vwYF0u2
XB/F4aV0ZEl47WBX6uf1ECeAZc+oOmKMzvgrU8CZIaWkNI9Ttds83jLbrjcNugbROIDsf1xL/
lTrrjW/z7fHxr6EPWdbXW31nQMc9MSsxK7aBrSAjWMMK4zpEc7tGGsYi2MLHJh3MZcGHEkhPeaYfXK
08kw2asDw9LIQlxGoatm3Sk3q6btdoaKBoquhs9WWQ7U8iXHtH+WxV3plzcBw+Z/rpaRp2Vv5GkZai
KUGN35KugE03+EnywdBFpv0NM6WSai2hnYN3J8mKA1IIOzmKQ17CXgJqbI3joXiZBldBqmqU60OyHr
IzvzA2Mfz+B7ryVbULTkYgv+FgKnHYAbMi2GEOaJy4IEmvNK20DCm0GU2oeo4g7u0t+YAJ8m62VFOl
mzlHWPfIXsfTwgbwwPJbG5NrEdsqJO1Ra1FoIthE57NpuQHLzuaoeDSt1j9P7fttU2ci/evMRbaON9
JauQjbaF2NZSQoiUECJ5Dt1TQqddZkiJJclC82o7LKJtJYT3CI8JLUR5QhOfZOYpF+1oEm1r4uU/
ofxetk8VRUwyRbxnjX6et4iktKk/UEsDBBQAAAgIAOdJqFR1zqfDwQQAABENAABJAAAAb3JnL2FwYW
NoZS9tYXZlbi93cmFwcGVyL2NsaS9BYnN0cmFjdFByb3BlcnRpZXNDb21tYW5kTGluZUNvbnZlcnRl
ci5jbGFzc71WWVMbRxD+Rlqx0rIGpFgQbOLY8SVARg7GxjZgBwO25YgjkQPBdoiX1SIWpF1ld8WROz
8jVX7Pa/Iih6SS8nP+Tx6SytWzOkBHGaGkIlVN7/T09Hz9TU/P/
PLXDz8DuIKvJJzAPRH3JfQizpsHEt5GQsIs5nh3nn8tiHhHQgD3JLyLpISHeK8di1hqx/
tY5s0jPx5L8OCJHx+0owcrvPMhb55yJ4qIVS5VESkRGkPbuG7ozi0Gb6R/kUGYMlOk7UzohjaXz65q
1kNlNUOaUMJUlcyiYum8X1IKzrpuM8wkTCsdU3KKuq7FssqWZsS2LSWX06yYmtFjk6u2Yymqs2CZpH
J0zZ4ys1nFSPFFpkxji5SaNcYQTGtlo935nKObBsPxSH9iQ9lSYhnFSMeSjqUbaTLtrTOd1hxFz2gp
hr4GY7Zq6SWPAdU01vR03qIArkReDv0A0AXFsgklJ6kt534zXD7qbJprlmA0P7cYA81tN7TtuGE7iq
ES9mCZmbyjZ2KzSo4sAkk9bSiOG9to7fh4PZH1mlvkRVSLm8KwcAg/blipA1jHaiDVQ/TmTEoZFqeU
2tR2F5VMXpvZyVmabbu0hBrttlgkzT6UtXo8DFKuknc8sWvxdFcn9m6unNxXW2TvWNJR1E2a4foRsU
ZnmlDM7KhaMQoRaYZn/5LZJuG0mgLVx25+dUNTHVIvtoj6oJeGjq3mSkijwjHeUog8SClp5i1Vu6vz
7R5srkwNcV8yTqKPoavWr4w3cFbGOZwVsS5jALqIDRmbyMg4z0eyoBQP7uO9r9jrhFmEya1JfiTDAm
WIIyOPLUoeGdvYkbGLO3RoJmR8jE8YIONTfCbjc3zBmy9lRHGJMq+KCBmX8SbD1H9QmxlutLw9dKT3
YcVJoTjmoQ4POJq00vmsZjiV88MwdLSqSxdU5HF9DvQfvQA3vXD59pLXFbscAFWf4UgLi3aQj6oLbD
ryPwRTd5IZHhx93TovldVHWplHFxzd7e6dQWyGqi64hG7zQtJRrWHw65Wc666aUM5Ffr8QxXPajuO+
gh7Ru8ZwO6JupLSd+TWGcKPQ4/wyy5Pd9Uh9SWuy7gXsPD875I/Di8cbPnbCkUZ64Qy9406A/
wJgvCJR+xr1YiSpRsA38BzsO/rw4BS1ba6yD6+D1w/
XAKdxBhAEXrZcea4kz3NJNhdwseQ0R9ZekhcGfT/Bs+wdKMCbXBZICMkCfLPRAtpmo9QVC/
Av7a8agkDtRXqJRmjtfvI84CI4XfRWROB+RWiUkV0nWQzSbIEXNZJ89WGSHLH/
RwSWn0P6tiaqoQNR+Us+h1wbTgbVQVqN+/
mVomgjuTLI8RfQXoC8h2MMc5f20MHwNWboo5PhBTw3hV4h1FVA8Jov7HuGYJS6oT284sXSNzhJPW/
YV8DxXiHsE54WEC4N/f19lKMTXHSnIFI7QlFfIzyj9My/gUlMYBm38QRvuahHCNMQRTxMoz6yHKH/
VZe1lQo7KzR/1I1uBdfJhwc3K/FyzRhpxuk7DOEPBEVM/ImYiFvsN/T8TkO3XUCMFiySMtk1TeTeqc
uXArprmW2QL12TxOhUaWdGiFm+k9LAYPQFegp49WUepFJE+4A8JUjTmHHl3X8AUEsDBBQAAAgIAOdJ
qFQlkB6FUwEAAGcCAAA/AAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGluZU
FyZ3VtZW50RXhjZXB0aW9uLmNsYXNznZHPTgIxEMan/

```
NlVRFEUTDzBQaNg7MGTwZAYoonJxoMQ7mVplprddtPdBV/LE4kHH8CHMk4LQaPEgz1MO19nfvM1ff9
4fQOASzgsQQH2TThwoeZCnYBzLaRIuwRqp94TmzIaMhnQfqqFDDpnQwKFnhpzAhVPSP6QRSOuB2wUo
lL1lM/CIdPC5EuxkE5EQqDjKR1QFjN/
wmnEplzSmWZxzDX1Q0F7KoqYHBvkjQ6yiMv09tnncSqU7BBwI54kLLAzfnki0Frj9JsymGg1M26s/a
LPsgRJtbUVBEp9lWmf3wnjvvmXsQsDKEMRHBN2CFz9+40Ejr7cPGYyFRFfXUIT8vhDZuWAmHkYXcy6
mOdwd1rtOZAXe7+BsWTVBlY2YRNP9UUV6luW4kAZtpFhWDtL1j3OyOPuttrnc8j9hB1j04mFNRZlK5
i7hJlTBXatxT3bXf0EUEsDBBQAAAgIAOdJqFTVJBE1SAEAALsDAAA3AAAAb3JnL2FwYWNoZS9tYXZl
bi93cmFwcGVyL2NsaS9Db21tYW5kTGluZUNvbnZlcnRlci5jbGFzc5WS30oCQRTGz5i6af/
sP72BXtQkEpQrQlhBIBSseD+up21kd3YZR+vZuugBeqhoHGFZcqJ1b2bmfJzfdzj7fX1/
fAJAC04dOHLgmIDjx2KOUhGo1/sTNmc0ZCKgjwolG4XoNjLFp9EEfeUSqN6/
+5goHoupAycEKh4PBFMziQTObZhOpuYpyUXgdt3GYKBZbavtqqt1kMvcZtpr6XdX78cyoCxh/
ivSiM1R0DfJkgQl9UNOn5mc4rgXRxET4z4Xf6zgan2MsR+u3ZdzFddrg9OVVHQGXnhgfl/
rH0wGYIjSbQx1djqD9upQXWt2vHgmfXzgoXY7y+B6yyCivFh0EWjmnSNtJFD7bUjgJi/mVgazCIVKs
10mQKAAi69UJLABRf0q6VdR18vg6FsBNo1SsShVo2xZlG2j7FiUXa3sQc3cl5V9ODDn4Q9QSwMEFAA
ACAgA50moVIe/u/5HBQAAAQwAADQAAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hb
mRMaW5lT3B0aW9uLmNsYXNznVZbcxNlGH62SbNpsj0QSygHoQiUNGkJykGllVMLcugBSG0tJ9kkH8l
Cshs3Gw4q6Ize+BO48U658UIdWwaZcRwvuPBPOOPPcBzx+TbbNEdHvOh+h33f53ue932+Zn/7+6efA
RzAZyFsxQUVF0PowoUebMGlHqSwIGfvhbGIpTDex7J8XFZxJYyruKbiunz9gYobIfRVs3Q5S8tZRs6
y8iGCuBlETm7mgzDk1i0Vt1UUFKhWyTEss6ygf+aWfkdPVhyjkEwJZ0JBT8rImbpTsYWCXY1vJ6vLg
m7mkinHNszcxFFmaLqdqxSF6SzcLzFpQ13UVEEvlxky2Lw3GZeZ4awoZ2zDJaMg0gqvIFSupDNWsai
bWUZkRckWGV3GL+m2yRBGGGamkuaeXCiXFQQmDdNwjirYFKtDPOsIW08XxMToogL/
lJUVUr1hirlKMS3sBflOcrAyemFRtw259jYDlsfQ7+QNFm3/jGXnknpJz+RFsqjfEWbyrq2XSsJOZg
pGcqpKV2LPu4lNBagxURBtPI71845sx7xD+XtTjp65PauX3FS3xUUVZJv4zyCyJKGccObXbLEhNtpi
jD3Nex34hPN6+YTnCAVvMOulq6XVQZBNL6ml6mwwuMak0SoDRb1UXrDqI6djrYH/g49aA+wjlel60w
ZsUa4UKHRzy0knK0YhK+wJ3lQmUlJDYoRIJwoF6269Ul9slAbeUns1S2ijVBB1IaG1OyBIZ6NLp82d
SFkVOyNOG9JL0RZJ+yRVDaOw2Or1np7Ry3n2VcM2vKqipOFD7NQwjJ10lAYbZdpKg4NzGiq4w4I3K1
ZxV8M93KfcddQZo0zIPdjNC9mhRBp2YbeGj/CxhhE5+wQP5PIhga6s6x0eH+YNv6ZhL2I0yZX1i8+9
T3Fcwb6Xa22DhPn0LZFhIyOtl2Zt09XjbjqW7XZrcU2qG79oGWyKf2H5wikFQaMWGG24OmsA0lh0xZ
y4x1P9pjs0WrtKSd7xhovHk/Ws7H6sNVb6JyAlS7eOt7V/J58ysSDMnJN3lZ1VsC029W/
RQceqbmEnf2S28hctgIj0DhRs56oLO/hH+9TWr/GPnXbne7xxxBvZU469jB1FnOsEV7/CBz/HY/
EnUOLP0LX8BL4V+OM/oHsFgcRTqApmx54iqOARtnLSo+AXhObiP8I//
hThLiw9fvHH94TwY4zP7VD5HONqB/pJLIpx7EcSE3wexetcgYR9XIWxj8R88nC+3w+4s22M4f80zjW
O1Z0DnB3kfCO6XlC7T8UhFYdVvAn8SbCq8rcY9DaOeLqSXMn0bsnzOzdgzC2f3Dzksqjid3tnV0Emu
DeJd/iUIIe5I2OC8UiIxYiv48ivCfCwADPWsYIelkKlbTC0dhjHiXGiLcYxvlNaxPQ2izndRoxCyJN
e8jFGy4PUeGIFvS3nn2fSjAsRrYbVeqFiiv2RJ0/XmPzugYln6KNT+mfIaOA5tIQcV7FhiWPkOWKJV
bzyFUKJgeFVDC4lIhvlOxkUcSeRqBe76RGGmmOH3JBVbJZUfS7VI/y0Ai6yOJdoqgUM8cNtBz/
ZRnCZ1K/SHNdY8OskeoOC0ozKYhk3ucrXSROetCCjT7FuVVNp6P4LQyreHYwOyoqeaV+5gZbKFVk5s
0PldnmVO4tzHtgpzwcDVB3g5foaqv8x/L5vayKr7SzXtXPAQzzotrar77iEPF+DnPYg+yVkRPumA+K
9OsT+RsReF3GmveJIi+IHzHnYQfGIp3hvzfmHPHpq3L+CTS1gnxPsizpqas27s22NH2k2/pdtjT/
nRs3/A1BLAwQUAAAICADnSahUx2yU6LIAAAABAQAANgAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlc
i9jbGkvQ29tbWFuZExpbmVQYXJzZXIkMS5jbGFzc5WMTYrCQBCFX/
kXJwpm5XoWs047eARRGFAUPEHZKWJCpzt0q3O3WXiAOZTY4tKVVdR7FHx8/7e/K4A5RgmSBB+EdO/
OXsuqMkKYLlzTsC3WlZUd+yA+r/nChMnSauNCZcuNnI6uSJASZs6XilvWR1ENX8SqX89tK15pU6kX1
dc3YfxjrfiF4RAkELKHXRm2pdoeatEnQv6eNPsEoYPndOMifj30Y/cwiDmMl0aGHkA2vANQSwMEFAA
```

ACAgA50moVGQwtcNfAwAAqAsAAEkAAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hb
mRMaW5lUGFyc2VyJEFmdGVyRmlyc3RTdWJDb21tYW5kLmNsYXNzzVZbTxNBFP6mVAa2FQoK3u8VoQW
21LtVpFQuSqEoSsLjsKxlsew220X8I/qgvniJzxg1XkiMbyb+JmM8s1uxpKAsQuLLXM79nDnzzXz78
ekzgNO4paABiXoaehQ0IqEgidMhnMFZjnMc5yXnogKOSxwpjisKQriqQEEfR5qjn6HWmTVK0QRDImv
ZeVUUhTarq/Pivm6qi7YoFnVb1QqGmrHm54U5kzVMfVzYJd1Oke5lwzScXgat3a/yn+VdoZkKrVTHJ
EMwY83oDI2SMLYwP63bt8V0gSjNWUsThUlhG3JfJgZlYgy703cd3R407JIzsTBdNskQvm6aup0piFJ
JJ6kBn/FH17JKFQlpvyX/WtLqLBl2WuaEI2wnV3QMy+S4xtDkrT3HxHTI8mx7dk7cF2pBmHl1wrENM
5+qpnT4zarKE0VUI+y8LHGVdTp/y5WnECvSzpVpG+8nT0MmTz61e6Oi6B6h28EDDA+3vbn8lqnHbcf
Ag4QceuSQZEj6t8KgTFgLtqYPGrJjW6tkumXRw2jGrjAiaOIYDGMIwxzXw7iBEYZufzVmGPIZ5B3zn
mktmmu04N71WGEcwEEZ9BjdK5/uRtZx1roeI7WpDvc6mFCgcitDzslhnCGzBXDgP3svmvSisPXV2a/
H6PvX+03X1G/X+mg6T4W6XGiaXipFLyToxqR9X+cOD3sWHKOgEjhIoFhFIIzK6w5DWyUo5qbndM35p
VtJ4rhJQL2qCG/WgtNtBpFKlN0cAE1t6BHYlG3Wu8Wg++ebuHHl8lvx/x/Opc17xFH6tTXS/45FIhL
2aVWDgHwFQP8Z2g3TvobmcCz+DoFYvPM9al7TPoAWGhskjz3GDvYEIfYUrURrJR7JYw/2A+6KMJpoD
IfIm2f1BeoQpDkde4vABwQ7P2JHAF9QO9b1Fc3L4FNx4tTF2HvULy1DmZJyywhNxTuJFO76xdy55AY
sQ2mWBtkzNLHnaGEvEWWv3HBinqOVcNI4huMUhlydoFWAkk4gipNkp42odWDfEeU4FZE2O8oB54gr7
XC3Aux3BWqlKfaowhVfccWJ0uS64oi7NZCrTnS5Ne52bajYR3M/uW2g//
Reohym+UhQQbs0EakjjSQuQH6kk7hM/BDNvZBf8CQywQCyNI8GlZ9QSwMEFAAACAgA50moVBD3cpay
AgAAMAgAAEEAAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hbmRMaW5lUGFyc2VyJE
FmdGVyT3B0aW9ucy5jbGFzc7VWbU8TQRB+9lp65VpoUUF8R0WFq3K8CCqoUVGJSUUStIl+W9qznLZ7
zfZ48S8ZP4BvJBr9op/8UcbZ61lOWjQnIZfszszOPM/M7eze/fj56SuACUwb0HHBQBJDnSQN68gZiO
OSgRFYOkZ1jDGkim61ykUp7wibYTTvyrLFa7y4bFtVvmoLa03yWs2WVrHiWAtc1u3S7HbEDEPiuiMc
7ybDxFDU4OECQ3zWLRFxRhnmV6pLtnzMlypkOZB3i7xS4NJRemCMe8tOnSF9+7lny0c1z3GFUh8IYc
vZCq/XbVJv/D2PUAZ+SnIwjEYVZav81ZK96HHpNYwMvUP5F3yVWxUuytaiJx1Rnhl+xhDjsqxSbVlk
6HJFCELHOENPQ26w0qJHFS23QW7DFbWkFia1U25QTcoV8674XVvqj3wW2lUalT3Mq2OCwY7cG1Epx/
xu0tZH1UB9PR4dgMFYdFdk0b7vqF7ra/
EZUW8mjRTSaricRicMHZMMcxG5noiXwl0Tbdqhf7clxTil40oaV3GNYWYPLc4w/
f/7yXBrr61I10zUvWGY/EcH7dYRI9GiGN61O5D73Lzh87IfZbZkSNcTL5XurXuSF3hlxd7lhitggD4
cSfqa0CWnup2kDmjqBJCli7Q50jSaUybbgmbmPiC2SaqGbhq7EaPIDXSwTRjsLTJk62u4I4uDgC8pW
EbPIfQGoFM0Ky8t9qaJlSAbiGMbQ2tiaGQ57GP041iA8ZAiiB2Zz4g/zZnvEVMZdmzsAPziAw40XJu
AmQBQScdJ0kg+gVMB9J2gZkOh5j4iYW6jGmqFfUOSfQ+lajSRjQZyVhV+OsC7Sz6KP2HmtsA2d2T4O
pRhoomTwBn/tSnpLAb9jM75kedppwCT9kmnsYe8jtB8NK7+A0ycpHlAAWST9Ctg4iKMX1BLAwQUAAA
ICADnSahUDfQvMiMEAACLDQAASgAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQ29tbWFuZ
ExpbmVQYXJzZXIkQmVmb3JlRmlyc3RTdWJDb21tYW5kLmNsYXNzzVfdUxtVFP/
dzcKlm6UNEWhRqbTSAkkgAfwoJMUChbY2JWjaaqhfm2Ubtk12M5ul1Af1RR3HBx+c8cH/
oo46VBk/3pzxv/AP0fHc3RSCCS1L2xlf9t577rnn/M6552Pvn//8/CuASawp6Mb0IfqkFfRgmuOsAh
kzCjJ4LYxzmOWYE4t5jvOCbVGBggscFztwieN1BZeRVdCJKxxLCqJY5niD402GdnfNrA2mGFJZ2ykl
taqmrxnJinbHsJIbjlatGk5SL5vJebtS0azVrGkZy5pTM5w0nc2YlunOMOjDQQ8/nN9jWm04lR65zi
DP26sGwxFBWFqvFA3nqlYsEyWatXWtfF1zTLGuE2VhGEPPnHHTdoxF06m5+fViXSaDesmyDGe+rNVq
BrEtBjRgsKVY8klY32F9pFOb7WTotK28qzluruqatsVxlaHLn/uaadMlyWvD2VvaHS1Z1qxSMu86pl
VKN1NGgprVpIkQhTSnJJzcJJ0iwPb4yZ3+xN+gZW7X8uzBYGyr6Wpwaq6ucf/x6p8QriWL9NtXtKoX
IpRDlDQc1xi+eerxG9QB417ES3dT4jMuPhMME8GlMCh5e93RKVJFUvQ28YyJW1VxDH0qenGUg9SmH+
O2hKi3VLyNAseKiht4h2Es2E2peBfvMVwIiOKadduyN6wWuXJsry0VJ3BSINYYpvarbtYprVcMy124
qxv1WDxRVzBQj9PRMvEN+CkxMHSqNjTGENlJn1zxlqG7HEUVOlaFfgK5ENDcy3sY29t6Q2i5yXD+SZ
S54GB9OLMbmmPsBrvXxrnHrVtUIIImS4BA9Y8wFPZVhQ+W/

```
Iqm60atNngmRUVgNnCFGvFxrLtmOUklT5S/XQSq6yXDZTjdaIIfmQ/
ONpJ28EwJPJPB8axwUCsI77qj71v57ymX1sbmdrCbifwXMzVCSpOKRu6camHRjYd7eLvN9bQ4S/
rYzBNuT4/ot0Eb6///wqYPrhED9DfdQz/hLBIRDZJmIUiiX4LhWVpdpHWIRjUW/
xFSLJ7YROg7Wkt4jr6HxR77CG3sY4TZJ3ieaL20R/
zoJ9nwZtSFiMbwIk7Xpf6FDrTRaGxBLsQTbBNtS7EfIN1He+IncAm/o2Na7pP/wLhPPvQtoltQCnFa
hmPEr97bQmchelj+BUcKodH8fUQ20fXbFqIFcWK0T37A+cw9zygB9zg4wf2UoH6GfvY5TrIvkGBfIs
2+wjL72oP/kg9sG76BIQwTbDEboZmEMFYQQ4Jk9pN7RjFGT5WkZ6j0N9IcKY7x7gjIiZN1c3PEK9PI
Pf+xHf+1C8HsQ09xzGfZVswJylFPMcfLngfF7BW86t3QGU/
GFF6gcY682U2vo+NEGaTxlCxeTxnE6ZE0IURFOiBeTwsQb6MMwZHozjPIyxLep/EDWfkXUEsDBBQAA
AgIAOdJqFR2lBnkYAIAAEMFAABUAAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5
kTGluZVBhcnNlciRDYXNlSW5zZW5zaXRpdmVTdHJpbmdDb21wYXJhdG9yLmNsYXNzrVNbaxNBFP5mc
9k2bmxML2q11ku1udRuWxGRlIIEhECshZS+T7bTdGoyG2Y31XdfFPEH+B/y4ksFH8Rnf5OIZ3aXSk1
VAj7snDNnvnO+c76d+fbj8xcAD7CRg42lHCbMksFds9yzsWzCJRsVG1UbKwzZTalkuMWQKpX3GNJ1f
18wTDWlEtuDXlvoXd7uUqTY9D3e3eNamn0STIeHMmBYrPNANFQgVCBDeSxaoZaqU/
d7fa556GsGp6GU0PUuDwJBCTtNX3dc3ufeoXB7/Fgo96Xm/
b7QrteVLmX2uNo3PexwHQi99A+CGoPtRTtqarnUPOLH3O1y1XFjZG00Um5Qjt8Ppa/
WzXQjgNPjDZpzXx4cMDDKybdC7r14xvuJBGfYnrePhBfWRiPlho37DA9L4w6+XjN/
xXq1xrAxfi7DZEt2FA8HRpjNP3U6CGXX/aXm5qgYW1Qq1/
IH2hNPpRl7boRu1SQ5yOGCg1W4DtbgMhR+L+VgGjM2SPPt/3sHznDF4zHMnDcfw9q4SjJMJ/dr1290
lK+F6YZh9py7Zm7W5CmaYXU8MtLNpldLNegdp2EZRWnn0M4ly8hmKp9gfSTHQp7WrAlar3GRfCcGYA
pFssyInSRrQqfJFqsrJ0htLXxAZmFo/PSQoqmoUtEgrDfIWm+Rs97Bsd5HVStxZlLVeLOYi5iKuEye
RT04uIKrVGc+7ug7HFZ4QvzXEv5HdJYim69UvyKzQt8JssO/DJGP6QogBRaSIo/JWokC7HwF5mLAaa
8Z3MAinadwM8Lfwu3I3sGlqH+L5C4TyzyuG3hh4idQSwMEFAAACAgA50moVN8JNMTCBwAAPxUAAEsA
AABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hbmRMaW5lUGFyc2VyJEtub3duT3B0aW
9uUGFyc2VyU3RhdGUuY2xhc3PtWFtYHHcV/82wMMswScgmkEA1GVtSYLksEJI2Dd2GEJrQcElLJNKq
zbA7hUmWmXVmFpJ4KdrW+601VqlVq9bWS6uJliUUe/EWbb37fT776pMv/T4f/
Pqp58zMLreNzYKPvvzn/G/nnP/5ndvu6/968WUAe/EnGe2YKKfBkNGBMxLOygghxSuTPJg8WDykeXg
fn7clOGG4EjIybsKUDAXTTJ0L43wFLuD9fOgDMj6ID/H0QQkflrEDD4XxsIxOPCLjo/
gY73xcxifwST74qTA+zSufqcBn8TkePi/hUd5+jLe/IGMGFyV8UUY9UmE8zt8v8fDlMGb5+4SMr+BJ
PvtVGbfjaxK+LuEpAYqVdg3LHHZtwxyX8E1aGVq2QtM+09TtnpTmOLoj4PZ+yx6PaWktMaHHJrUp3Y
xN21o6rduxRMqI9ViTk5qZ7DdM/YRmO7pdt5zbQQFlvjwBbdfNyOdAdysSS4tvycATn1zGhhiUOq7m
6hK+Rbx89YZ5QUBXsa9adpsfNaWlMmydyv4z2pQWy7hGKtZvOC7tlQ8b46bmZmwSU7dqu8ufpzRzPB
ZYKM7s3AnDqWsrxka+Pny3yzANNy4ISkOxlzeEbLFoFgveRgBqHBEQ6rGShMAWPjKYmRzT7ZPaWIpW
Iv1WQkuNaLbB82AxxAgIqD5uWtOmr/EKfzlarDqFGRFesmV22+OZSd10BZxoWOsQjRtzzbCW5x5Zy1
zAJjqYODugpYOXb2ZENdsd1M+5pJiAeMMGNdg8rrvHNGfplSUNjfd6D6eb6ZTO9iz1AoiSTdoDfijI
EfuL9JLAtyQ8LeHbEp4RhPj/oyCHRLF32724Ec+18dDOQwcPe3no5GGfgI7imRLyw1bGTuh3Guxv1W
vOtLKXKrgTRxUcQS9TzwrYtpQ5u21bO8/pk7e+o+AQuhUq1bcouBUHFNyGgwq6cIeCd4IecOB6Ncw5
aO+5hB74395guyVF+6pfudT6PU69mrR0RzUtV3W1s7qqmWouzFqpCCyF2dDYGT1Ben4X32Nlv6/gOT
xP5abbVPXJtHs+f02d1hw1bVtTRlJPqg9Ytpq4huhWCT9Q8ENcUnAKJyVcVvAjjCj4MUYkvCDgSJGI
DBiOQ67rOyTZIMhwVQXX+RFzFJeD1no0V5DFvIQrChbwooJFPE+RquAneEnBy4xVfCCTcg3KCXnujj
qt2/p1838Fr1LkK/gpW+Rn+PkKMPwYVfALxuGXuKrgV/i1gB2rTxzOGKmkblMVODmhqwpew+s8/EbA
zpw4w1GTetrWE2STpNpCh36LqxJ+p+D3zPwPzPyPGBLQ+z+pEwIOrSsrrWCxtcDawQ0kOwG3rT8bCe
hcT2qnIraUB/pc3dZci4BqKzYLCdhXbGEI8iEVLxrLyA8nNaplBwpU7Pv6VycAKuIFSm9VoWrPzFO6
Oe5OeLL6qH6u7BlpWUsmV93OiaG62lpcCSKLUn3uTqWs6XyJps5HMpxeTlBF8PPNJGC6aMuu7YvWh0
```

2FlkjojlPX3tlGRevudeqR77sKQPbfjbHGaan7J6xyhr5QAO9iG4a36MWu1RFRLnOMC6wPgT0S/
FiJNDSu/blSm/eGXC5e5hVhIx9y1Ssu50KRGEgTmsPNI4k0vc/2hsa1nspgJZNLPWEViT0SZFRS+JR
mm16KWXk5j0Me6X2M9NGim7yAqWHFThBH95TND+NfT3zJpOhqKdiJX6NSHFx1/
ppBv3Q+7Fq5LLqtgCpkxjTPUuQ2m8g0w5mxoOrR2y1z0ApqBFS0owNAGSLcAhEV4S7I+1Ij5H2pF/K
+1CdBwGGiRfRUVnJz5d0UudeinWM0exwltAI0R5vmIEaFeZREFxEanUfpHMqizXOQoi1zCEdrQnMoj
9bQqnzZ49hH4y5IRJZCFglRsQy1ooTdYhh7xHI0iTLuojOdKGX+OI4BwKNYb9GjWPMSj2LdQx7F2pe
SdoMYCnR8iqTw+cFoFhVP4IZFKKORTaGXsHm0JPoCpOEr2DKPyleastia246s3Sa6rGkB20ScIkbbL
xHHEu8VtWwBcQskcStpvx03i9WIizXoE2u9F1T70vMvGMQJ3E263UN0COLuXUQP42Sg7WlaE+i7h+S
Fs6iaRQ1LXkC1gFlsWsSOUTbzzksrldjMhhBVVIs3QhXrPMGKzygQfI93VlBpkRrdQFoHfflQOJD2X
B6bMl4Xo8sYhQNGAjeS/n3hIbIt2/0Ntu0s6pep6huypqAhy3l+BbW+1Bv6/WtvE0AYvN0jToeezpl
gVw6U3YVBWYAqYKB5Ae9gsVVE3CjgVdw02NSSRd2z//6bz2bPVXQQJV7BzYuoJw9tiDRmEWVuWTRFm
n2aDxKVBd1t9eexq4gQJQezLNou8XTJ9nFsIVO1kWt0kNvuJ/
Bvwf3iAUyLXbgoxjEr3oEnxUN4RjyM18Qe/Jm+fxF78VfxKP4u3uWZWCVeF8lx30WuwS79RmDsOOYw
insDd9lOLvgmjku4702US3j3P1F7685KUEC+NwDUJuAkhiza1NxSE6opnYdweSWowj88icf8c3m3VH
A/BbjgUaeDEFOgBSGmYCwIMQUJL8SYSkL3EsIDnoRx7PdcqoyCdgb7wH8cziAe4r8vZ9BPJ97DgirD
2EHzj4D/IZ3BN0LyfwBQSwMEFAAACAgA50moVFSijfy2AgAAxgcAAEoAAABvcmcvYXBhY2hlL21hdm
VuL3dyYXBwZXIvY2xpL0NvbW1hbmRMaW5lUGFyc2VyJE1pc3NpbmdPcHRpb25BcmdTdGF0ZS5jbGFz
c62V30/TUBTHv3cb3dYVGSqIP0FE3Q+kA0QU1Kj8MMaJJBhMfLuMm1Gz3S5tAf2ffJBEIfHBZ+OD+u
KTTz7pX2E8ty1zwF4qpEl7zr3nfs7Ptp//fPgIYBzTOlLI60ijkCapqNRhHRlcU2sjSjKTGE1ijEGz
G55lyyTGGbqf+vIid1zhLHncEwzGIymFM1Pjritchgdl26mavMEra8Ks8w0hzU2HNxrCMSs1y5yx63
UuV8uWFAFl6ABymnzetqTl3WWYyx0el19mSMzYqxRrlzJcWK+vCOcZX6nRyvGyXeG1Ze5YSg8XE96a
Ran0PLFc15LVAHnfqYYZz0eNqS2H0szW+esVQZrjBXvkM1d+yTe4WeOyai55Dh2bzr9giHOnqqI9sM
nQacs9iLU2iDbQo+hTxpYLttx1nNkzF4vtMonqdK+7rC1bTObkKtUll19O4jrD5hFMSlTCqD9bsVcl
dRtlGIsOYNCX7HWnIuYtNXm9B2xGVBENdCGrbhMGdBwzcAN9BiZx08BZnDNwC/1JTDHMHsVgMkz9f5
cY7h22DQylqGVkmIjc/qB5KcvdHV/dn65GTagQ7uYOOau6X9D1upAew0g0FAboe5ymLzW98arfJGmI
qRmglW7SHpIWo2emwHYQKxTfI75FaozMQeZxOvkFHewrdPYNJ2itNzDHSZLgSwrL6DqFvhA6FUK1Qn
EbiTdNnkb7YN9bOFqTo+E0zvgcmsOQ85j8UwToLLxDfBsdxeFtaG/34X74uIHAsInrDHFK0kmKkXwe
/SH4ThhgWoEpxuR+6M+WGNNNaLoZ4wAuhqhJXw9RFOPzf+XT1XH2Cyn228cZgWGAy4K6cCmEzJJlPC
jYDtjWvmA+tWTYWrAhv/BKuowrfkZX/ZM5yhkYRIKaU4KRUP/kEnqg/sglXCB9UAGyqb9QSwMEFAAA
CAgA50moVG8GVHm9AgAA3wcAAEsAAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hbm
RMaW5lUGFyc2VyJE9wdGlvbkF3YXJlUGFyc2VyU3RhdGUuY2xhc3O1VVtPE0EU/
qYt3Xa72IpQL3hBLQotsqWIF4oXQsAYK5LUNMG36XYtq+1uM7sF/SE+
+At88UETUSOJP8AfZTyzLFgChizBl505t+98c/acmV+/f/wEMI35JFIYVaHhmtxdV+kzpmIceRUFTC
i4oUKBrqCoIomSgmkFNxlShtNuc7tRsWyToVhxRFPnHW6smXqbr5u2viF4p2MK3WhZ+goXrtlY+BtR
Zoh7a5abKx4a2xPlwwgZO2fZlnefwRgLGxyW6HiNIbbgNOiQaalY7rbrpnjO6y3SDFQcg7dqXFhSDp
QxeTCG7LOOZzn2/AYXQe6qxz2ya49t2xQLLe66Jvk9CnmC3MG4VJVMm7+tmyQJb9uHYWis8oqvc73F
7aZe9YRlN8vjLxiiXDQl/X1G+rGOvezYOwCpPdRXDoILy38v6X7aGK+f8o5fPb+11KrTFYa5ZMlqZv
cBTEoKGtLIaDiJWxr6cULDbQwruKPhLmYVlDXM4R7DQkhq8y89UyxZwvWq3XpgZhg8SC1zUwOWj5Jh
u7j077VeUSI+YFg8lnZgmD36b1HwkGEm7GjlpvxhSVjuTvNMhps1agbeaCy+8QSv8VbX/
Ef71g4D3seMWoobhum6uZkiXTjToS8NOTLv/vtVc7R6m4fwOq48xbBRsRF6STR6YyIYlDMKZDJyakk
TRYyafQAMp0iqkEeU1ky+sAmWZ98QyU9sIvo5iAXFkp01obI1pJmFIdJlyUYxtJ4F/
J3MESHMcxgOkGdJll7xfOErYh938eJkB7N7cOK7OHGcxwWyM1zESICzgT7fq5T/gihB9dHKviP+Htk
tKKtSkhZinviA9BaSq4GofvJPK3OqEoF5SLBuT97Sbt5SkPcy7RVERp8ouEIuV33SOZyhNU80UvQon
45FqLAFXIrJp7mASfJI0jpFSWb8WiT+AFBLAwQUAAAICADnSahUxYNGhMoCAABgBwAARQAAAG9yZy9

```
hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQ29tbWFuZExpbmVQYXJzZXIkT3B0aW9uQ29tcGFyYXRvc
i5jbGFzc6VUW08TQRT+ZrtlS2mxXBW8oIK2hcJyRyhysdGkSVNMakj0bSiTsmS722y3yIv6F/
whvPgCRhLDsz/KeKa7wUqrstq0PffLnPPNfPv+5SuABeSiiGMyil7514MpDdNRqNCjmMWchnkNC1FE
pBzBkoZl6b6i4YmGVQ1rDF3rhmW4GwyhVHqXQc3Z+4LhRsGwRLFR3RPOK75nkqa/YJe5ucsdQ8q+Un
UPjDpDYqfmGraVs6s17nDXdhhiecsSTs7k9bogj+2C7VR0XuPlA6FX+ZGw9LcOr9WEo5dNQ6fIKrf2
ZdGX3KkLZ+JqxiyDVm5KVLacunY6L082qH86T/XsJj/HMBs0/DJ4nmZUNWSO/
sIhP+K6ya2KXnIdw6pkPRu5JFMtxp29Q1F2s+2adF4DxSxd//
D+LOeycrPK8SzDfPBYhu6SUbG425Cjf/O7ThuuYeo/97UedGQbVChashtOWbwwJLaG25qZkYViSKAv
hnU81UCoff5PuPIW0IrX4c4GWW1cw2YMW9gmoF/dIUMxYAM5Xhd5qy6suuEaR6K9k7G/
eMQwgmcMM8HmK6NGGTb/8xr+MgJv/
wyDnQBAtyYo1AgAFeF6JenN6EulW6BVEm42wKm9nAxDrb2ZJvXrJQ/
RzWPYShU62TtDOptux/6Vu+tf7HZNOk/
PcZweaXpa6dlWoUgYk9RPkk6UEQ1PnkH5RIyCAfrvkkrlGIPExzwHDOEWUSaX6Qd/pGRhostTpwidQ
33NzhD+jK4LaMVMm2pNPUdEyt3TI+opoieX1XoRIvYdBpT3SCofmlUXvcx+Vcndxp1mJ8u4S5yMHMc
9jFFskg52nzg1sU3dPfC7WyEL5UV8cuoCPRn6nSJ28ocjxr1iCfkd95OsElX8+bDO8xn2HC47DWMCj
8gewuOmfxKpJk3jJtGHNLM4MhQ1Sjx9EhHSZDBDcoToIkZ/AFBLAwQUAAAICADnSahUo+Ern7MBAAA
hBAAARgAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQ29tbWFuZExpbmVQYXJzZXIkT3B0a
W9uUGFyc2VyU3RhdGUuY2xhc3OlU11P1EAUPbdbtlBWWBDXTwSFh4UHihofdA1RNzGabIBkCQ+8zZZ
JGdJOm+ks8rd8MvHBH+Bf8T8Y73Q3IQRfKu3DuXN77jntmemvPz9+AniFzRANrIbwsRpgLcA6oflOa
WX3CI3u1jHB7+enkrA4UFruj7ORNEdilHJneZDHIj0WRrn1tOnbM1USlg4Kq3J9KEwpzdAKy49aX7S
Wpp+KspRM+TjITRKJQsRnMsrEhdTRVyOKQpooTlXUz7NM6FPnOlHZvCHZIyzkmktj9+Wl/WCSAM8I8
9dc97pbdY2uW4S5ZuVxJrUlHHYH5+JCRKnQSTS0Rumkd0v9hUTaz6K88uDYTypbnixSaWWA54TX3bo
2L3pu+7zLXcLL+rP8BsN8bGL5Sbl97dzg7LggWphBM8AG4f1td5PQvor2YHQuY87i7f9HS9it+9GEn
Xoj/jr/PT7c1YTnsgAh4FXESIwz29/
hfePCw2xF4ib9xhzXrQkBIe4APp8DLFW4jLsVruBehR3Xbzvmg6n4G0ZvKk7/
Fu9MCBPxqnqIR9XY44r/hO3BnQbfTzHvh1hkbDPed/T27F9QSwMEFAAACAgA50moVOzsrtq6AgAAGw
YAAEEAAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hbmRMaW5lUGFyc2VyJE9wdGlv
blN0cmluZy5jbGFzc6VU7U4TURA9tx9sWZZSi4AoCiJIu7RsCygqyIeoiRFBg2Lw32W7KYvtbrO7oD
6KT+BfSAQSTXwAH8o4925BoP1hQ5POnZk7c2bmzO7+/
vPjF4ApLLWjE7pKYlxFEjlh5hOYUGGgoKCoQkFOiEkhphKYFrH3FMwoeMAQ5V6ZIb2yw/
e4UeFO2VgPPNspzzK0ubXAdh1S5mzHDuYZxjKNcY2e7AZDbNktWQxdK7Zjre5WtyzvLd+qWKKSa/
LKBvdsYdedsWDb9hm0NVkwhCHzheNY3nKF+75Ft49XXK9s8Bo3ty2jyvcsx/
jk8VrN8gyzYhvLbrXKnZIo+Jp7vuWNnEWjcZJlK3hq+7UK/7LKq1T1aibbbOzO9YCbH1/xmmxOwUOG
ROCG1woeMWz+FwutdluUvEU+F4QoCjHJMNk6CkMHN03L90eK0wUCe5O5FG9NKVLX3V3PtJ7bYnm9DQ
gTIkXDFaQ0pNEtxKyGLqRoqnyeGNUwh8cMfRehn+zalZLlaZjHgoZBDDCwvIYbGFCwyDB7iUEYUv+q
rW3tWGZwznUSVWiVcHqFMmJzqh9wL/
Df28E2Q0+TRyT7gd4kAeWUGPLNAhpcdT6I8YnW2sIQveNJ+j5EiHsinrS02AedcenrBj3/
ZG2SFaWzSz9GVB8/REzPHYIdyNQemRYD2DxibAHtbBFJtoRe8g+FaejDdUBqogyTmigUIZ32Vi/
jSxuY078jlo4foe0rBn5C2TxGQpjt5Gd0HEH9hv6Ti45zF/uUH5U9tQks9kz2oYW49T5u0l9BZPSdg
lt0MXjagEGnCIzrR9D2T6cLkV6eQYqHSCkx+e168hrVjQlkfTx3jMjBhfQZma6HIaeEKBiuE6LgjiR
EaCMYRTSVwF2MNXRGk17sbOZcZ8MSOoKMlFn0y/1EaNX3cQ0q7YR+qcRfUEsDBBQAAAgIAOdJqFRLi
yvYqQIAANoFAABLAAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGluZVBhcnN
lciRPcHRpb25TdHJpbmdDb21wYXJhdG9yLmNsYXNzrVRNTxNBGH5mu+3CuoWKUBUtoCK0FNmWL2NKi
KRR06SCSQ2J3oYyaRfb3WZ3i/4NLyrEGG5eevECiSZGr/4mY3xnW1Esapp42Hk/5nm/npnZL9/
```

efwSwgGUd/ZjWoctFQ1LDjA4Vs3K5oWFObpM0NWQ0ZBkiK5Zt+asMoWRqk0HNO9uCYbBo2WK9Wd8S7
kO+VSPPUNEp89omdy1pd5yqX7U8hvhGw7ccu+S7ll3JO/
UGd7nvuAxGwbaFm69xzxOEu1d03IrJG7xcFWad7wrbfOryRkO4ZrlmmRRZ5/a2LP2Au55wJ0/Pm2PQ
yoFFLUwnizt8l5s1blfMNjLX7UkVKMYJsmXlLF2A4+15osSrOq5POPb4h0HeaMnn5Sf3eaMz+4nCG1
s7ouznuj2pggYKXkr2Ono2J49DeZZhmO89lqG/ZFVs7jclRyt/6rTpWzXzJ7Er3bysUiq95DTdsrhr
ybHjXeXmZJABA1GD7t+ihiWG9R47znNPFGxP2J7lW7ui+yaN/wMh608YGEGcIfb7FAx3/
svFO5G6TSTD8GlMMmR6PTO6ajVhV/xq8BTpvs71lgEZetk6GM7Qf0CFIg+ErAGyTJKMZHjmCMo7UhQ
M0hqRTuU5YqQbbQDOYpgkk0R2gj8jRH5gLX2IkHoATX0LNbQ6+6u1rCb2oI+o+wgrrcS+VPcQVlsfo
D5iRwinCR1pUZJQUHiMfkxQXmBAeYmE8gpTyj6SymtklTdYUA6ChhbbRTsNSe08LgRNruEiaQo1t4B
RXKKcCUziMmkqaaDoiHGbfYXB2qseW6Nxxjrj3CR8iGR0Jv0J2ix9h+hr/
YWTaLuFmLQnOklukVQ6hLLTCY23Acf9h3EFV2k/hGsBfhLXAzmFcyRHyd+PFIZIG5fwWB9Nk0Iao98
BUEsDBBQAAAgIAOdJqFQREjSyEQIAADIFAABAAAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2Nsa
S9Db21tYW5kTGluZVBhcnNlciRQYXJzZXJTdGF0ZS5jbGFzc6VTTW/
TQBB94zhxcFKaBFqgfJVSStLQuIA4QCqEVIGEFDWVAj1wQNokq8RVvI42boH/
xIULSBz4AfwoxKydlrTJJdSSPbOzb968mV3//vPzF4CnqLlIY81FBmtZ3HexjgcOHjooEzI7vvKjl4
RUuXJAsHfDriQsNnwl946CttTvRHvAkVIj7IjBgdC+WY+DdtT3R4TcvtAjqVuRiDiYf6uU1LsDMRpJ
3txphLrniaHo9KUXiGOpvE9aDIdSe52B7+2GQSBU19RLWNYnyOqEQiC+tCWvdNQcRn6oCEvlxqE4Ft
5AqJ7XirSvevXKB0LWH51AUkL3jOYpHGEhVBNsDiqEYuKf6aI/o8iMsvM2N1WJFeVCtReqE+n7s7q7
4AxDNQF5rboONgnPyvOyPq6bK2J93iY8mT+X4LbCI92Rb3xzd5anMDXTdx4OsnyAW7Wqg2oej1B0sE
V48f8T4AH8m2izfSg7EeHVRQ+OsD3vCM4ISY6W4AQiYgb+U2rz8dmr/
FNnYB4HlhkbCJd45bEltunNH7C+sWPB5W/GBKmJHPv5BMB2EbBZF4oxUYltQlLnLMtQV0up77C/
nqN5H9MsJ5CEJvau4CqICZdwPSa+gRWOE26eqqvGa37PK/
s4oYwSyoKhvD1OfM7WGrc1ldyc0JM+1ZPGHdyN01Zj/D1cZruCFO9sYMF2OWMD19jeMvBC9i9QSwME
FAAACAgA50moVCpxzOHkAgAAZwgAAE0AAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW
1hbmRMaW5lUGFyc2VyJFVua25vd25PcHRpb25QYXJzZXJTdGF0ZS5jbGFzc8VWW08TURD+zrbblu0C
pUDFGyqitkthuapcRAnxQkKQBCWKvhzaTVltzzbbLfDq//
FBIpfEB3+AP8o4Z7spFYqmYOLLzJw5M99cT7c/fn77DmACjzXEkNXQhuE2kkYkMSUZlRdjGsYxEcVU
FPcZ1IrHPSuKhwzxVe5WLHdNKhj0JSEsd7HIKxWrwjC37LgFk5d5bssyS3zbEuaOy8tlyzVzRdtcdE
olLvLLtrBqKIMNYLMMIe4WGJLLH/g2N4tcFMw1z7VFga7iuWNfhtE/x/FR8w3RCCAyZwvbm2f4lD6N
3yrcRcrMrDOEF508ldEpTVaqpU3LfcU3i5as3cnx4jp3bXkOlGFvy6bm9r0WH4WzI16WPdsRv41hqd
WEzoKiRnUULO8Fryy4hWrJEh5NJZ3ZILUjyML1Vqxdb0GOaT6dudi0NUccB1ltMpUL4sd4HZ1CkXW5
aMklnmb4+p93oFXfMX9rlN1RScYkGZdkgmG8dShqx5pTdXPWM1tuV+qUzYjsjY5u9Egyo6MTCR1dSO
rQ0KFjAP1RzOqYw6Mo6EU9/0fLx/CkRaQmEF1NdDPnHxX91rTaYIap9PkGPNLaCjK083z+6a7n8nVe
rNK5t9kr+ivwqXRwk74AbfSVCCEpR05SUi6Bz2kPiEegyAUBxaRTiU5h4r0GO4JiDB0gZGQPEDaGD6
Du0YWCFNEUVBLfQFXeQlM20Km8Q4/
yHpfozqgBoA9XAF+SAZkvyZCKL8lkQqS9imtB6CxxaaWEP9fjRKRG4T6uXrsNcBmuoz/wNANP1ThE5
MsJ50KDs1p3vkGdqTlPE1fOci75zqmaQb0iFbf8iph8QAHMPNnIEAljn7q1j9AhosTUY0RNoigVxBW
vIaVEDZUm0o3BAGuDOkj9RcwYyg4fge2dyGrF95+s2dSziuFO0OcY7gZ9juGe32cppZEhXIXmI9GG0
E58gDRxQtLD8q/CJC5TlrclWEKuzSQehLVfUEsDBBQAAAgIAOdJqFTFVwIqXBEAAB0qAAA0AAAAb3J
nL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGluZVBhcnNlci5jbGFzc61Ze2BU1Zn/f
cnM3MnNBZJAgEGQyKMkISHISwjhEQIiSgIaEeOr3iSXMDCZiTMTHtZ3aYtva32Bz9JiqlUrKgkx1e5
ure26drsPW7vutm53t93dPnartbUij/2dc2cmM5NBCO4fOffMOd/5Xuf7fuc7J28ce/
lVAHPlpQIE8YFq/qiaP6nmQwN/LsBHOGzCi49NjhwxsQFHDRwzYarvcRMWBwUinJY8Q/IN8RSKV3wm
isUwUSR+QwpMlIpZKIViFcoIGamaUYYUmbhRiv1SYmKCjDZkjIkzpdQvY9XvcaoZrygCJrplgl/

```
OUCMTVTNJjZzpl8nUUsr8cpYhU0xU4aiSPNWQaSZqcNQv0/mVz6hmhlpRrpoKxaBS8Z2pmipDqk0sk
lmG1Bgy20SdnG1iscwplLkyz5D5JpYrvnPFMmSBIeeYWKkoGmShIYtMrFaTtbLYL3XKA0vUzFK/
LFOaLFdNvSkrpEH1VhqyqhAXybl+WW3Ieaq7Rok+3y8XKB3X+qVR/W4yZJ2JKxTj9WKxJxfS7cqEK2
SGIRf5pVnZcLGJVtmgepf4ZaOSS09fakqLXGbI5QJrTTjsRBtCdizmxARj13XFg5FwczwaDHc0RDq7
7Kgdj0QNuUIwucGOOWvCMSccC8aD25xsIkGRuzp93ZWC8RvCW8OR7WF3cr0djTnR5rgddwy5ihIvOM
HcZwXFOYavFpQ2BmMxinZn66Mdeoq21G+KO1F3NGaILRijR84NRmPx5u5W6tVph9vJYIWzKRJ1ssYN
aU05oH67HXUy5LYJCjMGyMdK95ZgVMQVvWJnamTtFnubXdMdD4ZqGu2uxYKC5mBH2I53R6lvc+Zsnf
szZIc7alwGi9dGoh01dpfdttmp6bS3OeGa7VG7q8uJ1rSFgjUJvdcGw46ryOKllFBsh0KR7Y3BHU57
whVMvMsEo/V4xl5wpqTd6Yo6bbb2M2XSXfSPq0kwUqOHNkaDHCbr0dNUmET10pXBmN0acugFX10wHI
wvFeSXV1wi8DRE2h1lOtVq6u5sdaIXK0KKWhtps0OX2NGg+p0Y9MQ3B6nG7FO21N0DpUx5SsuEgkp6
UfagwNullghWlV8+1MMVnyxYS2tPE09+hW2DP6lGDqYCc9WONicZiKQ6tzyNag31UtafjnBLG5PYWc
GKYfpt2pCMIk+fE25nFjFM1jBAI8mYVjuWwzQJCgyX6GyujSQ0ybcVhzG5zGRMdmWbctItz2W8qdkk
sr1uuKZnGj02Mxp3diUjMtdO5UhNlWojyKxtK1NXL9XHWpshm3hmafi67JRZnUYgTB5Mc0JYIiRj9e
HBpJ9TfhK2ubLK7FIZvyFmd9AVgXQD6rku3O7GLRPNimRg35hItN2JJgPTHY1lAmCzE6eAUYSbTVrr
JE1RGs3aYEwR5W91djJxt9mh7pNHylAY5FInHI8qFqvcb2km1E7Tw0pQpDuuYiGnmUwGHhOddjwdFD
WLc5PDpBmZYTnt8YaoCb/5nfYOwdRMD5wgkqZl+eAEZItzmZHr3Mi5uj5bldM4Xeaf/MzKIZslhyEd
LKIM2WxIUNdIW1joEP+3RIJEj7ryNL4NkVDIadMicwF2Dkzy8hRRUZgOWOtat5CJgjc9GUttc7YMbn
PMSdUwvtbuTZvUfo8fImiFnlHiNqnKgfmRk2FdpTJ4HYs3wbLcdimSUzTtdE6tXBmRgOyYgT3Mcrut
zYnFps2fPVswt3y4MFHBciJvx+xBPgsVn/rh8xlSHiUZLlIMCxM/ztZqrj5d9kPKGH8d6d2SxWyOdE
fbWAkq5B87hMUsxcDCbbjdwq3YbeHz2GXhS9gtmJDkva473tUd58Y4dqcrwpCtFl6TkIU7pZMN7lPN
XTwGB809z45tpsmsH3LoaEjYkoh0WXgAjwhmDe9wGCKHuc6ks+QaiSqVYqph9K4c5vmZs2hWvLoN2W
bJdtnBs8+SnXKtJZ+T6wjs2XFryfVyA0OnutqSG+Um4tkwVUgv8JXkmxnX1dWXX7XkypmW3CI3WPJ5
2WXJF+SLlnxJuEtmYnbJrEpLbpXbFMHtltwhdzIM9JSawR1qRw3+rr5yViVTP+04SBa8q6LRSNSSu5
Ta44YiQzDUrkBDaNndco8lX5Z7LfmKcsV9stuS++UBSx6UOy15SAXC6BxHiYqRQxl7d3HUcbh3RJHT
KvIGr2HKVXtUs5f4a8nD8kjShsSBE97qtKci8pPDbQi2EKJOS73s66Mlj8pjGa6pj0btneo05GFhye
PK83lVZRZ+in+y5Am5VwBLvir7LPmaPGbI1y28IfsteTKLTeqgtKRHhaVnemx6OCM63aPCkm/
IdZY8JU9TznS9kd9kVMTKytQCS55ROzc296FgyLOK/
jnBlLSSj3DfYYdYWnd3sh5JXQlYvLkuKJsxPTajLBgrs0NEj/adZe0sjcJO+yylxbfUhj1vyQGVMrU
NdjgciZfZ7e1lkbS1NteGkyNtLk0sbkfjZduD8c1lM6pnzBrGhrr7Y8gLlrwo17LwGubG8j7QNMwlJ
31TWD1Mhid6bBh+mOZ+mBAs/7RXruFD7wkePBpOBz6HvIacXvpmv5AwRk//Pjb8kyDz5WXRqa7OkYs
lg0ih72g66sbkuhDwwpdRQGf81tA5MrOKzzwCd8biTueQxxW3cFg8eNnKKincV40sUGQx67NjrojpG
QViouqtGHqn8unbFFeOL684USEcOEEZrnQYl/sJgzP3D7syG+51d7hX/bO1XiVD9eWpH0xt8tgMRyQ
3X9XKm+1Yk7Mjrp+0WOt6wvrHmPKKXPeLok57Z6t6lYjGkydiaXmOmv0y9dJyTbcdimURJPeMBM5JX
Pn/5RqDNQdXxNTdp7s1lkij0vI1Oe8gI1SmpZm3OYd5n+aS8glvUyMTotV26DeqpcN+1ch6+DGC4XZ
nx7pN3Faau0aFwZrcZpsaazVeCNbn2tJPp0lhWyQct4Ph2AXqtcNna6ARVOeUdIKSU11n4pHUK0x5T
kNGdjhxlneDxphudRhyFPT6Qk64I75ZBzvdURgJN0XCya0u0pRJM1Yp/eYPN90TMZcFLskjXs/5WNj
Uh0KfgEBMDpNmpF5ZijOS131byicX19qVTqwtGkw+TXapR56FOVLuhMCZ9YKw1dmp8X7k4GSjrVzmV
bFDp/n1e5Om8VP8Je7DlY9dd2/dd6TBd470/TkJfKe/NU3MlaCDkeDdFOqObU6eXlkuFFSdYmAlXjk
WDUvXzIjLCYDcZ1MXp7GNQeW8UYMk+h9QTMV2JxaMOu2pW5dKle4YzkIQGwB4kaeu4BB8gb/y8EX+8
Tau+7fqb4m6rANF49Wdjr9HkpZ3b7Z389cSeNgDRlUOwNvyAnx9MPrgP6AZ3MPW5Be4gMvW4svsWS4
57sVX+FWs7kuwuoKsFO3Eyj4UkJ3Z0ofCg8hn12qZ2YcRByGDfEtIDaxjW09u63EGLtT8x7o8EvxV7
zbcTwk38+b/IOdcpfM0VUHlzEMY2YtRz6fY+rR6V6axKkixKsBD2MP5vewrqx/
m3wN4BPmKaf479GUh2XgGUNQygOKWypeQ348SQR9G92FM4wBKWyqrpA9jm2b2YxzLO0/A04/xgj3yp
uoFBH+BCbXe6oC3F2fskWcD3pKJvZi0B0UDOLNFLZ3c1COPcrisF2ftwQRSBrwkntKLqbW+gK8X00h
```

wHQmma4JZAW/Rkl58ZoHPpSwlxQxFawSMAH95rtZry7loNRdV6EVnaVqPN0FKtgGvL0U3+QVU7kUpq
WeSei8KB1DFjar+DpeQqNanzA74+jErj84aq3j5EhK1dngrwbvW0JRGgvLm6gHUkNHskrN7MUcRz+n
FXPWdWusP+Hsxbw9KAv6kuVQFt6nh+U3eBQWlBcplC/ZhIa3iL2XXjFozk6WZYGmmWM5v2lUgPcdf6
8Ey6pKXrm5ivgfTT6aXIjL1lp3T1HNsdnUvFlY9ryNExdOrqGG7hWG/FeMRwlR0MmAjOB/
X4GLEYKObM9uxEztwI9tbcC3z7DqG7PXYjxvwEkf7cBP+kjN/x0z9F+bqu8zQD5mLx3GrjMDtMg53y
CRebctwj1TgXjkH90kz7pfL8aBswcPShUfkejwmt+FxeRJP8Nr9iBzEk/JtPCWv42n5EZ6X3+CAvIe
D8gH65WMMyDG8kpdH7VUePINiaubDo3iM7f2YjMfxBHv7mXhqzKBmljtLXVw6g9Km4KvYB1NiWIavc
UUBtWrVs37KLUj0KvADfJ10BnVw1/pp2fvkvo+/6+VlPMmeV2VWMhPZewg9zL7x8lt8A08xT6fKL/
A0vslcfCYJMprqWfae40gdio5hpoFvGXjewIEjmG/
ghbJ5M8YdQTl7R7GGrYEXj1Cjw1h6GHmBj3BW3mH42E3LeBclX2L/
IHoTwDWfIwo1jErPQXgqB8HExb7vUvnX0rDPSGHfrblZ5A1h8ddk8UZOFn045LLIP0hvEX/khwNYpO
CyloCzmEBU10LcWNKHpU3VKUTqx7I8bBzAcgZ2fa2nuh8rCEfMeheO8KbqaThqqPW5bFYqNqvIRmMH
c+TcJJPVZHIec8FwuRQEChJcLlG9BKiZKu0WeJ7AlIA/V1b2Yw2Z9WDy4PTEodPHnwz4S84/
hAsohSqsVWxfR6HmvRdGSSMHPIGCgNmPpnxFPzt/gVdB6zpls9LOp6BJa3em6iW0M0q9KqMXHML6Bd
6e4z9XSy4cuqQrteQi5YV+NOvlWvgZVSUXe17BhhYNaTyYm3uxkQYtqkracylNKSVMtJRc5hrlzSBv
8SQ4uguP31HVi8sPpIDkWpzJ9i2mzY8RwE9QgbcxBz/lyfcOQeSfESY0bMfPeCC9y/
T8BVP3l5z5T478N36FX+N3+A0Osz2G34uJ9wgc70sR/iDz8EdZjj9JAz6UDnwk2/
Gx3I3j8j0R+YHkyZuSrwPvLp7gD2AlfcKUpsSn8DIGGIjbGYjfxitM2l9hGqXuo4a/wyR8R9OtJ9V+
+shHOZaGA4NSriKcfZdrGazJlGbvNXyPoVwgr+F1fJ9hH5CXCA0DTOk58jBTYBc81PgVJsITTAeV5v
P4dTX5m5QmbzKBXQk/5JjL429TPH5Ebyo42I2Rx1ELH6HLwN8b+AcD/2jgLSIDCD6rYA6dMPBj3f+J
JjqCJQSMwziD8HAUEzV4vJB/nE7yZi0FjqJIuOgwph3BCA0+XPv2YcxX0GKqx9dEXfIBvz5+zx/
AFQyZKxs9S2d+H76SxrWV/bgqvWDA9ETBUOudtBcFVTyLP7uxSh1CV6te/
tKe428zfuzBg2iWBgeD3vETkk2MFAvFMgpVUozZMhoLpBTLZSwjfhxWyXi958q/
VSyD3mF8eZVejLGf6VLofPyce8nSHCsYb/+q7VjOqNudDsCa/
t/4VR4fD89RFBv4dzkCQzuBTvuPj+gwF1J/yQptP2NIg5nUce+UJ/
bObKx6dWn+Ak+pZ9I+NFeVeubUenXd4E3UDYEBtLaUtCVSz9t8CO19cFiNKHjZlJzvyJ7f5eHB/+wA
GlpUCdiHzY1VGtKY8E3VSYSoTgGER8v0VCVx5V19xOeneXYqxsh0TJEZWMgjbYXMxKUcu0pqsEVmY7
fM4YE8Fw8xD5KeHcNs2q+Ptt0YoXNDVbR7U+XmXmavKje3YLbOoTztxynwHYPfwH8JHVmjY3Dyn+E9
ilIdoIyyqUV+5vyvEydLDb9qL5TXPM9k1bnL0s4ULwGCgrn4t0zgoYvzs4vk3Iv/J6fkvFOT/
L/4fY7FcgqSGYnvpS4l8/QMUFwS7MUWHgyeHnjyX0TlgSGMnnPvQNZy4cT7evoPLNMU3o5iYQat1kp
eljoxgRgbZOk2gcVEEF38XsNvlJEf4zfOld38bvOYRKIgyzmTiB1k4WSSVxCf43wpv9fxez2/
N3jyWNoFcRPpb+b3FtJdJF78la/g/wBQSwMEFAAACAgA50moVFGjBjBFBwAADBEAADQAAABvcmcvYX
BhY2hlL21hdmVuL3dyYXBwZXIvY2xpL1BhcnNlZENvbW1hbmRMaW5lLmNsYXNznVf5dxtXFf5Gi0eW
J3asJGqcpqmaxZZlxSqNKantOnXSuHFqOyUuNm4pMJYm8qSSRh2NnDjQltIWurG0rGnZN1MIkHJIQs
hh+Q0OfwO/8U9QTk7M996MZMsat8RH58y77767fu/eO6N/3vrTXwAcwuUo4iiqKEURQLEVvbBUlKMI
u5tnVNhRRNxNpQ0Oqm1YxDnxOK9iScWFKDpQjKIdn2vD51FV8WwbnsPzEXxBxQtRqn0xim14UcVLUX
ShGMHLYv1SBF8W6ytC4lWh+VobXkdVbN8QBr6i4qsRfC2K/fh6G97EW0LmGyq+qaDDKjumVaocXZp2
bLOUJ2firL6oZ6qOWchM6uUhBa3TZr6kO1XbUDDXeDrsbgt6KZ9xDQxNWHY+o5f17IKRKeqLRilzzt
bLZcPOZAtm5jHdrhi5Y1axqJdyE2bJOCX9D43QT3vZNipGyXFZlcZQpg2HIvsaOT7upSHjvGPro3a+
WqQ5Gtq6Rm3CrAhL+9exNjDVMmyWTGdEwR3JNQLjjmHr8wVjqHdGQeiYlTNEsMxmqlqcN+zHxZmC2I
SV1Qszum2KvcdsdQGnCyHQ5FOBVpYQuSAouH9zeDJyy7Nw7wdb8NMNOQtm5UM1m3xTU7Vql7fdDy8F
8UZQlso1YMZ85IdvN3RxZVumHT37NItTGmaVq/iWim+7HaZg3O8iN+FIXH3EsWptsz3Z63ubz1Qtxx
gt5U5aJu8i6V9FfpotFc+y6hIVcaVVp1x1FHQ1KRytmoWcYYsbMCtjpl2hlPIEHfom61PqnCkcNwqG
/28N36hbF/RKrXR3JJslehmTRpnR0lJNrCu5phWPWYWCkZUQC9GOgpWfMBaNQn0mxP2FFaR8DzZo64

P+Pv3zZBwn/VLZbGd25g3n+LoJFauVz9oZ1d3E3CCdLXouJy3O6IWqsQHyrNfwonveSvka/Bd8ZG+3
GzYNRXTaqtpZY8wUEyDeJNcvItMwgO8QtVUkTuiVBfa3hhT6mg74WtDQj4yCbasHo7atLwkANXwE93
EgaPguLnIqaHgb72j4Hr6voP/2slYwsJmsNfwAP1TxI76V1uOu4scafoKfKrjbG6ODiQOVdKLxdSZ4
Dcqn5s+yfDVM4ZSGn+HnfFltMB4UBNIJDb/AMqdDj4YJTGp4EPMafol3VfxKwxN4UsOvcUnB3jVjgP
2R1wu1CI6fzxoeAjvdnBI9Byo9iZLlJHLGGWaa6xeXxnx+o+G3eFfD7/DUh+HbBBebonkS1ZjyViXT
sZhWMCnHsVlnxBtapyYoxiOHzxQBlTrs7FBJbhrHt4uoqE+2an30dDbYdD9ItjQwGvasUDqR8/pwst
l4M8c3gpYzll3UaeMBnz598oONrL5LBMyl3Orca5xjG79MIlmr5OimyH6HTw4CwO1+g5SJ5wUc3X5K
fmm2N8456nNEicBzudFCAffwUzbOj+wgYqLpoSDNXQAHuWev1/f3cs8Gr+8PoZXrAD5K+n5AGaAFlZ
znUtegpG4gMHcNwSsIkQyTbLkClWSEZOsVRPuuo03BZPo6NAUX8SiJLQr+hvapG+igzNbB0MGr6LyO
mILBcFfYk0sISgpuG2xJ/QGhrpau0HVsD2J2eeVfyyu/f48RhPAxPvvRxmeKu3389D+AXejhsw9HmN
kJnprM6BwzusCcnmU2h2U2YfLb8QAG0UKp+zCEYVo4wvVBjDBD5sfdQ4CkRnGUeBwjrXF1OQ+TOk56
P0IrhCqoYkzFIyqt4Ra6VYyrOMndf9D9XyLnYvkodTgw+CSSOE3fwloitiP8Z8TngilmqI7DHdNzIU
FGBflH7LwsdUWu7TKyw9jJyHdzPVyPKOFFGxAjzL0pvM+1Rfi5gS5CvWsyNOJex2BIgCmBHhCUB3T4
rrcRTcfuvIrds+nYXXLpCq/ZBUeWV/6Rvoo9l9ehP8L/
ZEcQZQRJIpUmNocwRmwfIU4n+BvHSf4mmX8N/ST24jFKhCUONaRP4+MS6QilT2Oa8Uep/Tg+wbxX0R
dSMx76OxC8hZiKWYVY30RCxSff5/84F+45CnEae3CPkxPg2pnqu4q7ZwXWhCMRwKU6vlEpMEPUZmWk
cVehHl8nq2OQ5hR8Ck95KL/DNcS1l8buWVPqu2qlvm0qxRLfexHh0KXllX8HL9XBi8v051myOcJxhq
W0wOI1pesEU97FgIRDcem99SB68Wl8pqEcBeezHiAagjehEozdq2Wn84yvKA+HF9i+0qLoKwa9L8Ag
OybTf8fOG9g/FzsQksXYJ0rvGrr/mr4sp4YIOCYzdViHVdLnGOxSPdjGEF2c2nGn7KyADC0C5SZ62B
Y8yTJlF7+MxBMIi4pfLfUWr9FWSzzsmXdzMvg8g7yX08Pe3WrCCHPqCWD2vXXX+iJL9aU116rVw9WI
+6D0YeKsZ7HkXeueBpSmRNWk5cBK0sXB9a35CtdX2ZqvST8p10Ldzx45cBRJiTEToPRWCVCQ3Kelrc
L/AFBLAwQUAAAICADnSahUoS5jfeYCAABhBQAAOgAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9j
bGkvUGFyc2VkQ29tbWFuZExpbmVPcHRpb24uY2xhc3OdU9tOE1EUXdPb9DJAGS5eQCgXsRRlVKCKII
KIhqSKSQ0G3w7tSTs4nWlmpih8Cs8mvviAiUqQRH32o4z7zAxQiryYpueyZ5+111r7nN9/vv8AMIXH
SaSRkzGRRAi5BDpwM0nDLRmTcWhJxHA7hTu4G8eUmKdlzIg5L4Z7Mu7LmJUQ22ZGgzsS0oUtts20hq
sbWkF33DkJiaJeMZnbsLmE0ZbP8/7eYGZFK7q2blbmFuhIbF43dXdBQjg7vi4hsmyV6XBHQTf5i0Zt
k9uv2KZBEbVglZixzmxd7INgxK3qxCRfsOyKxuqsVOVajW1zU3tns3qd21rJ0LWXzHZ4edmq1ZhZFs
BrdVe3TCoer3B3XciR0J0dP09QQlvRZaW3z1k9qJg4PkFl1eMjzRaMnQteIDzFyuUlu9KocdOV0JM9
nyX8iLOTFPVf/OJV5gQKyMA3EpJFq2GX+FNdsO2/QPqkQFLQg14JXadsl2yb7QjKCjqhKhjDDQmDp1
VXDYNXmEGWuHzlfYl7WJThg2bKFncypuVmqtSCDDN3Mt5VmRSFRmQ8UDCHeQl9QToxz9QahqvXDe5n
OpT6EHQX0q1KFTzCooIlQWj6f5p9BnNtc4uXyNH2s42i++Tou76TqzRSq0VjVpsvhn+UjA9T+1raFn
wTXZB1Z6VWd3cwRO8rTY+P2ic8pbmLdiF0I0pragCNlyiygDCtgHTuAFLuCKGNA4S/IvLZy75Mo3iz
QD8iGMAVWil+Pq6iz8Pvx7UAazfAyue+IbqHtiPENlT5APGfuS+IHCJBQj8E4eRxOHyIVAi/
oOzTybBXsZc4AsOQMUrcx6huln65pur5oPoA/SMIdQo1g8gQU8FDo53Iigr8/RMhMS+oNcFEAxjfmi
GKDWMkEPOEIkK4IkAmDtEWwutWU6aRwoyH1+vnBnhiNYrrHg26ywHis4CW6rvRLmEPcuQjWfDpRLrP
cbaJo9okNYpQelGAZD0i438BUEsDBBQAAAgIAOdJqFR6n/DyhgEAADUDAABIAAAAb3JnL2FwYWNoZS
9tYXZlbi93cmFwcGVyL2NsaS9Qcm9qZWN0UHJvcGVydGllc0NvbW1hbmRMaW5lQ29udmVydGVyLmNs
YXNzpZLLTsJAFIbPcBdRBPGCcVE3BojQhUsviUJcESXBsB/
KsQxpO80wYHgrXZm48AF8KOMpVGOwURO7OH/n7/
+dOdPM69vzCwAcw14WUlDKwFYGtjOwk4bdNJQZpE6FJ/
Q5g3il2mOQaMoBMsi3hYfXE7eP6pb3HXKKbWlxp8eVCNahmdBDMWbQaktlm9zn1hBNl0/
RM+8V931UpuUIs6PkCC1NQo4WOG5K1+XeINijKb0pmahOGBRs/AjNbnwtpMegVKm2R3zKTYd7ttnVS
ng2Rcvfoi3UXDg4YLAf8W1sKRF2zHblRFl4JYID1P40WyOYIAdpyDBgHQY5f0HVSX0Gl13URmgFOt/
ZuJPK0EM0+hPhDIzFAEYFG3bDqHfcWZA7c2dT7kyw2qAm//+FDJo/Nrnoj7Xiv3WBA0jSTQmeBLDg0

FRXaGWSMtJk7QnYI73EIEs1NTfLsEo1twiQrs3xdciHcD2E48XYwxJqfEHjn+hGBBpfRg8j0UIEmlh
GjyJQuuXz1OY7UEsDBBQAAAgIAOdJqFSd6w1uiAEAACYDAABHAAAAb3JnL2FwYWNoZS9tYXZlbi93c
mFwcGVyL2NsaS9TeXN0ZW1Qcm9wZXJ0aWVzQ29tbWFuZExpbmVDb252ZXJ0ZXIuY2xhc3OlkstOwkA
Uhs9wrYAiiBeMi7oxYKRduHDhJSEQF6ZeEgz7oRzLmLbTTAcMb6UrExc+gA9lnEJjDDbExFmcM/PP/
505M5mPz7d3ADiG3QLkoKbBpgZbGmznYScPdQK5M+YzeUEg3Wj2CWQ6fIgEyhbz8WbsDVDc04GrlKr
Fber2qWDROhYzcsRCAh2LC8ekAbVHaHp0gr75JGgQoDBtl5m9aSjRuxNcCZJh2OGeR/1hdESH+xMlo
jglUHFQxqbpbSAZ9wnUGk3rkU6o6VLfMXtSMN9R1vovaxclZS4OCewl7IW2YHHFQo+PhY2XLOq/
+ZfWjKiBEuRBI0C6BIrhDGoFiiJw0kOpzxU9iI/V+YMuR6hf9a/1BhqOobe63jTaPfemE+qOsWkQaP
/70dTLL63RHoRSUFsurwL7kFVfIxoZINE9VVxRK1NlonL28BXIi5qkoKBibibWoahiaW5QeXWGr0E5
hlsxnK6mnhdQ/Qea/
kbXE9D0InqQiFYS0MwiepSAqm89c218AVBLAwQUAAAICADnSahUMomi6+oEAAC/DQAAPQAAAE1FVEE
tSU5GL21hdmVuL29yZy5hcGFjaGUubWF2ZW4ud3JhcHBlci9tYXZlbi13cmFwcGVyL3BvbS54bWy1V
99v2zYQftdfcTP20AKRlKQdUGSqCrdNMHepPcROuz7S0llmI5EaScU2hv3vO1KSI9mq2z2sLzWP391
991NM9GZb5PCISnMpXo8ugvMRoEhkykX2enS/uPFfjd7Enhf95PveLU9QaEzBSDBrhHHJEvpvLldmw
xTCjaxEygxZgmfj+c1zoCMqkAI9qaCQBEmkMIovK0OCvDYHLFOIBQqjA4A5orM9nS0m765hxXP0Uq5
rJXK94WZNAK5hI9UDrMgOS1NunbIcuCBB4Sh4CjOmbBzktNwpnq0NyI2gUNe8JE8LG8H8pqWha6vOI
cW3k1VDvxNpk4Az+FQnDC6Dc3hGAG/UXI2e/+pUC7YDIQ1UGp/
sAm4TLA2xJEpFmXMmEvSaiPbWidqXxoJcGkZg5iIAuerCgBnPA/
q3Nqa8CsPNZhMwRzOQKgvbqMJbSuN0fu0TVc+7FzlqDQr/qriibC53wEoikrAl0cvZBmw+bT1ckcn3
RlFuRXbm6bbK3Wo8ZahlRaF2AZQjJrzReA6T+QjejueT+Rl8nix+m90v4PP47m48XUyu5zC7g3ez6f
vJYjKb0ukGxtMv3u+T6fszQMoPOcFtqSx3Isht7jDtdEvr3baDLYcuMeErnlBIIqtYhpBJanJhu6FE
VXBty6eJW0r1L7hxLaO9o3gCz/dt+5dKfsXEAE2L0K9HTc4L9oiim/U/Zh/Dl8G5nSKHvNpqvkfbCm
1eONzl+flF+OfH2zkpFsznQhvbC6Sl+ZV2wluZOFI/4My1gB6CbHVaC30HDOg8iqlrokKmmDddHLu7
KOzJbG9FJdVbmNi1WZQpWZWTNCa7rYva30ZRD6GKwhZR45kyfMUSQ4KaQoPza6tR2AHUGs0ail8EF8
FFFLZHyyR8ouKdsN03apGCFRh/tBj4XGPgAyOck1tAijpRvLSpPsZBShsjlyw9A1ejPG96hlWCEqBb
qR0XpjI0UJvYj4BdEkwDsTd2fvsOasc0jpUIorBLxKuZlUj9KBKOusnQXrSrBZ2yfK0ENwc16GeqQR
ymvZP4l8HFi+Cyl3l3r2n/YGxQk3r9u6YTHvI5RdD2zZoViXJmTvBsQX5CX4yTfC//
V7KFTB64kSe5NpjvU30Z/HLQ0/+FbOfsmsGKlhXP28kp8yqjVtybrc/tkQT0+Qa9llVOOx9p/
xe0DlNgK2PbkL5Aydru+6/skcErsCtvr/nNsW98HqVnaDzRfpcTO/
tOaShTpIZbTNzI6CdhV9yVkpyncWPXX+4M0qMF/Sa/UcjTA3QmWa77skbaWqFI7KmvFg7oRfSKWfGs
UuyYFd2qKscjTzaO2s3bhuunfiv0oAXbfkgfWsTPf9cZt48GekaooN41/9DK7uGGLPFMUGumc9tbA6
SOME0v9kRDdsPvGI7CHws3CgfTFa0Yz2OjKqqK+3lQlW9WgPwet0tH+DQjYXdI2pOup62ZLfcNVNK+
3doN3Bz3VqjP6qnwFebINHZbrzujtXZ/TjuygwCPJohpjcUy352cIKc6PEX9q4F6EWstKztMTSB7l8
fD5BRODoFD6Adejhsj8YrGiJLTkw0p0cdwJvLd2Fy3ZO+kNK36N26HDDGVrPkjvqVQ3tNjN6E/
OXZxEIQ2dQM3Ax18esyHGu3gol/o8Ggrh4d7O+wu9bDTa/sDYd1v+xqNvX8BUEsDBBQAAAgIAOdJqF
SgxyUlQAAAAEgAAABEAAAATUVUQS1JTkYvbWF2ZW4vb3JnLmFwYWNoZS5tYXZlbi53cmFwcGVyL21h
dmVuLXdyYXBwZXIvcG9tLnByb3BlcnRpZXNLLCrJTEtMLvFMsc1NLEvN0y0vSiwoSC3iSi/KLy0Aiu
YXpeslFiQmZ6TqgRXowRSUpRYVZ+bn2RrrGeoZcgEAUEsBAhQDCgAACAAA50moVAAAAAAAAAAAAAAA
AAkAAAAAAAAAAAAQAO1BAAAAAE1FVEEtSU5GL1BLAQIUAxQAAAgIAOdJqFT5+/LtqgAAAEsBAAAUAA
AAAAAAAAAAAACkgScAAABNRVRBLUlORi9NQU5JRkVTVC5NRlBLAQIUAwoAAAgAAOdJqFQAAAAAAAAA
AAAAAAAEAAAAAAAAAAAAAEADtQQMBAABvcmcvUEsBAhQDCgAACAAA50moVAAAAAAAAAAAAAAAAAsAAA
AAAAAAAAAQAO1BJQEAAG9yZy9hcGFjaGUvUEsBAhQDCgAACAAA50moVAAAAAAAAAAAAAAAABEAAAAA
AAAAAAAQAO1BTgEAAG9yZy9hcGFjaGUvbWF2ZW4vUEsBAhQDCgAACAAA50moVAAAAAAAAAAAAAAAAB
kAAAAAAAAAAAAQAO1BfQEAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9QSwECFAMKAAAIAADnSahU

AAAAAAAAAAAAAAAAHQAAAAAAAAAABAA7UG0AQAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS
9QSwECFAMKAAAIAADnSahUAAAAAAAAAAAAAAAADwAAAAAAAAAABAA7UHvAQAATUVUQS1JTkYvbWF2
ZW4vUEsBAhQDCgAACAAA50moVAAAAAAAAAAAAAAACgAAAAAAAAAAAQAO1BHAIAAE1FVEEtSU5GL2
1hdmVuL29yZy5hcGFjaGUubWF2ZW4ud3JhcHBlci9QSwECFAMKAAAIAADnSahUAAAAAAAAAAAAAAAA
NgAAAAAAAAAABAA7UFiAgAATUVUQS1JTkYvbWF2ZW4vb3JnLmFwYWNoZS5tYXZlbi53cmFwcGVyL2
1hdmVuLXdyYXBwZXIvUEsBAhQDFAAACAgA50moVJtHxQFxAAAACwEAABUAAAAAAAAAAAAAAKSBtgIA
AE1FVEEtSU5GL0RFUEVOREVOQ0lFU1BLAQIUAxQAAAgIAOdJqFS0tOKGbQ8AAF4sAAAQAAAAAAAAA
AAAACkgVoDAABNRVRBLUlORi9MSUNFTlNFUEsBAhQDFAAACAgA50moVEux2AyAAAAArgAAAA8AAAAA
AAAAAAAAAKSB9RIAAE1FVEEtSU5GL05PVElDRVBLAQIUAxQAAAgIAOdJqFRYTKbfIwcAAO8OAAAzAA
AAAAAAAAAAACkgaITAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvQm9vdHN0cmFwTWFpblN0YXJ0
ZXIuY2xhc3NQSwECFAMUAAAICADnSahUQN2xuSkCAAAhBAAAMgAAAAAAAAAAAAApIEWGwAAb3JnL2
FwYWNoZS9tYXZlbi93cmFwcGVyL0RlZmF1bHREb3dubG9hZGVyJDEuY2xhc3NQSwECFAMUAAAICADn
SahUm1BSwC0CAADbBAAAUwAAAAAAAAAAAApIGPHQAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL0
RlZmF1bHREb3dubG9hZGVyJFN5c3RlbVByb3BlcnRpZXNQcm94eUF1dGhlbnRpY2F0b3IuY2xhc3NQ
SwECFAMUAAAICADnSahUy/ZceaEMAADuGQAAMAAAAAAAAAAAAAApIEtIAAAb3JnL2FwYWNoZS9tYX
Zlbi93cmFwcGVyL0RlZmF1bHREb3dubG9hZGVyLmNsYXNzUEsBAhQDFAAACAgA50moVHTMO4C3AAAA
6QAAACkAAAAAAAAAAAAAKSBHC0AAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9Eb3dubG9hZGVyLm
NsYXNzUEsBAhQDFAAACAgA50moVHB+ACriAgAA7gYAACoAAAAAAAAAAAAAKSBGi4AAG9yZy9hcGFj
aGUvbWF2ZW4vd3JhcHBlci9JbnN0YWxsZXIkMS5jbGFzc1BLAQIUAxQAAAgIAOdJqFQAjFIInxEAAJ
0lAAAoAAAAAAAAAAAAACkgUQxAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvSW5zdGFsbGVyLmNs
YXNzUEsBAhQDFAAACAgA50moVMQ8wLtOAgAAJQQAACUAAAAAAAAAAAAAKSBKUMAAG9yZy9hcGFjaG
UvbWF2ZW4vd3JhcHBlci9Mb2dnZXIuY2xhc3NQSwECFAMUAAAICADnSahUxOfLWzAMAAA3GwAALwAA
AAAAAAAAAAApIG6RQAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL01hdmVuV3JhcHBlck1haW4uY2
xhc3NQSwECFAMUAAAICADnSahUjfSgMqMBAAAZAwAAPgAAAAAAAAAAAAApIE3UgAAb3JnL2FwYWNo
ZS9tYXZlbi93cmFwcGVyL1BhdGhBc3NlbWJsZXIkTG9jYWxEaXN0cmlidXRpb24uY2xhc3NQSwECFA
MUAAAICADnSahUcRYJ2A4GAADmDAAALAAAAAAAAAAAAAApIE2VAAAb3JnL2FwYWNoZS9tYXZlbi93
cmFwcGVyL1BhdGhBc3NlbWJsZXIuY2xhc3NQSwECFAMUAAAICADnSahU6+BF9VQGAADfDAAANgAAAA
AAAAAAAAApIGOWgAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL1N5c3RlbVByb3BlcnRpZXNIYW5k
bGVyLmNsYXNzUEsBAhQDFAAACAgA50moVDtDV2bhAwAA1wkAADMAAAAAAAAAAAAAKSBNmEAAG9yZy
9hcGFjaGUvbWF2ZW4vd3JhcHBlci9XcmFwcGVyQ29uZmlndXJhdGlvbi5jbGFzc1BLAQIUAxQAAAgI
AOdJqFTQuEtSuAkAAL0VAAAuAAAAAAAAAAAAACkgWhlAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZX
IvV3JhcHBlckV4ZWN1dG9yLmNsYXNzUEsBAhQDFAAACAgA50moVPYP1RT3AgAAVAgAAD8AAAAAAAAA
AAAAAKSBbG8AAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQWJzdHJhY3RDb21tYW5kTGluZU
NvbnZlcnRlci5jbGFzc1BLAQIUAxQAAAgIAOdJqFR1zqfDwQQAABENAABJAAAAAAAAAAAAAACkgcBy
AABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0Fic3RyYWN0UHJvcGVydGllc0NvbW1hbmRMaW
5lQ29udmVydGVyLmNsYXNzUEsBAhQDFAAACAgA50moVCWQHoVTAQAAZwIAAD8AAAAAAAAAAAAAAKSB
6HcAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQ29tbWFuZExpbmVBcmd1bWVudEV4Y2VwdG
lvbi5jbGFzc1BLAQIUAxQAAAgIAOdJqFTVJBE1SAEAALsDAAA3AAAAAAAAAAAAACkgZh5AABvcmcv
YXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hbmRMaW5lQ29udmVydGVyLmNsYXNzUEsBAhQDFA
AACAgA50moVIe/u/5HBQAAAQwAADQAAAAAAAAAAAAAKSBNXsAAG9yZy9hcGFjaGUvbWF2ZW4vd3Jh
cHBlci9jbGkvQ29tbWFuZExpbmVPcHRpb24uY2xhc3NQSwECFAMUAAAICADnSahUx2yU6LIAAAABAQ
AANgAAAAAAAAAAAAApIHOgAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGlu
ZVBhcnNlciQxLmNsYXNzUEsBAhQDFAAACAgA50moVGQwtcNfAwAAqAsAAEkAAAAAAAAAAAAAKSB1I
EAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQ29tbWFuZExpbmVQYXJzZXIkQWZ0ZXJGaXJz

dFN1YkNvbW1hbmQuY2xhc3NQSwECFAMUAAAICADnSahUEPdylrICAAAwCAAAQQAAAAAAAAAAAAApI
GahQAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGluZVBhcnNlciRBZnRlck9w
dGlvbnMuY2xhc3NQSwECFAMUAAAICADnSahUDfQvMiMEAACLDQAASgAAAAAAAAAAAAApIGriAAAb3
JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGluZVBhcnNlciRCZWZvcmVGaXJzdFN1
YkNvbW1hbmQuY2xhc3NQSwECFAMUAAAICADnSahUdpQZ5GACAABDBQAAVAAAAAAAAAAAAAAApIE2jQ
AAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGluZVBhcnNlciRDYXNlSW5zZW5z
aXRpdmVTdHJpbmdDb21wYXJhdG9yLmNsYXNzUEsBAhQDFAAACAgA50moVN8JNMTCBwAAPxUAAEsAAA
AAAAAAAAAAAKSBCJAAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQ29tbWFuZExpbmVQYXJz
ZXIkS25vd25PcHRpb25QYXJzZXJTdGF0ZS5jbGFzc1BLAQIUAxQAAAgIAOdJqFRUoo38tgIAAMYHAA
BKAAAAAAAAAAAAAACkgTOYAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hbmRMaW5l
UGFyc2VyJE1pc3NpbmdPcHRpb25BcmdTdGF0ZS5jbGFzc1BLAQIUAxQAAAgIAOdJqFRvBlR5vQIAAN
8HAABLAAAAAAAAAAAAAACkgVGbAABvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL0NvbW1hbmRM
aW5lUGFyc2VyJE9wdGlvbkF3YXJlUGFyc2VyU3RhdGUuY2xhc3NQSwECFAMUAAAICADnSahUxYNGhM
oCAABgBwAARQAAAAAAAAAAAAApIF3ngAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21t
YW5kTGluZVBhcnNlciRPcHRpb25Db21wYXJhdG9yLmNsYXNzUEsBAhQDFAAACAgA50moVKPhK5+zAQ
AAIQQAAEYAAAAAAAAAAAAAAKSBpKEAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQ29tbWFu
ZExpbmVQYXJzZXIkT3B0aW9uUGFyc2VyU3RhdGUuY2xhc3NQSwECFAMUAAAICADnSahU7Oyu2roCAA
AbBgAAQQAAAAAAAAAAAAApIG7owAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5k
TGluZVBhcnNlciRPcHRpb25TdHJpbmcuY2xhc3NQSwECFAMUAAAICADnSahUS4sr2KkCAADaBQAASw
AAAAAAAAAAAAApIHUpgAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGluZVBh
cnNlciRPcHRpb25TdHJpbmdDb21wYXJhdG9yLmNsYXNzUEsBAhQDFAAACAgA50moVBESNLIRAgAAMg
UAAEAAAAAAAAAAAAAAAKSB5qkAAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvQ29tbWFuZExp
bmVQYXJzZXIkUGFyc2VyU3RhdGUuY2xhc3NQSwECFAMUAAAICADnSahUKnHM4eQCAABnCAAATQAAAA
AAAAAAAAApIFVrAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGluZVBhcnNl
ciRVbmtub3duT3B0aW9uUGFyc2VyU3RhdGUuY2xhc3NQSwECFAMUAAAICADnSahUxVcCKlwRAAAdKg
AANAAAAAAAAAAAAAAApIGkrwAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Db21tYW5kTGlu
ZVBhcnNlci5jbGFzc1BLAQIUAxQAAAgIAOdJqFRRowYwRQcAAAwRAAA0AAAAAAAAAAAAACkgVLBAA
BvcmcvYXBhY2hlL21hdmVuL3dyYXBwZXIvY2xpL1BhcnNlZENvbW1hbmRMaW5lLmNsYXNzUEsBAhQD
FAAACAgA50moVKEuY33mAgAAYQUAADoAAAAAAAAAAAAAAKSB6cgAAG9yZy9hcGFjaGUvbWF2ZW4vd3
JhcHBlci9jbGkvUGFyc2VkQ29tbWFuZExpbmVPcHRpb24uY2xhc3NQSwECFAMUAAAICADnSahUep/w
8oYBAAA1AwAASAAAAAAAAAAAAAAApIEnzAAAb3JnL2FwYWNoZS9tYXZlbi93cmFwcGVyL2NsaS9Qcm
9qZWN0UHJvcGVydGllc0NvbW1hbmRMaW5lQ29udmVydGVyLmNsYXNzUEsBAhQDFAAACAgA50moVJ3r
DW6IAQAAJgMAAEcAAAAAAAAAAAAAAKSBE84AAG9yZy9hcGFjaGUvbWF2ZW4vd3JhcHBlci9jbGkvU3
lzdGVtUHJvcGVydGllc0NvbW1hbmRMaW5lQ29udmVydGVyLmNsYXNzUEsBAhQDFAAACAgA50moVDKJ
ouvqBAAAvw0AAD0AAAAAAAAAAAAAAKSBANAAAE1FVEEtSU5GL21hdmVuL29yZy5hcGFjaGUubWF2ZW
4ud3JhcHBlci9tYXZlbi13cmFwcGVyL3BvbS54bWxQSwECFAMUAAAICADnSahUoMclJUAAAABIAAAA
RAAAAAAAAAAAAAApIFF1QAATUVUQS1JTkYvbWF2ZW4vb3JnLmFwYWNoZS5tYXZlbi53cmFwcGVyL2
1hdmVuLXdyYXBwZXIvcG9tLnByb3BlcnRpZXNQSwUGAAAAADUANQAYFAAA59UAAAAA

spring-boot-example\.mvn\wrapper\maven-wrapper.properties

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#   https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
distributionUrl=https://repo.maven.apache.org/maven2/org/apache/maven/apache-
maven/3.8.7/apache-maven-3.8.7-bin.zip
wrapperUrl=https://repo.maven.apache.org/maven2/org/apache/maven/wrapper/
maven-wrapper/3.1.1/maven-wrapper-3.1.1.jar
```

spring-boot-example\mvnw

```
#!/bin/sh
# ----------------------------------------------------------------------------
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#    https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
# ----------------------------------------------------------------------------
# ----------------------------------------------------------------------------
# Maven Start Up Batch script
#
# Required ENV vars:
# ------------------
#   JAVA_HOME - location of a JDK home dir
#
# Optional ENV vars
# -----------------
#   M2_HOME - location of maven2's installed home dir
#   MAVEN_OPTS - parameters passed to the Java VM when running Maven
#     e.g. to debug Maven itself, use
#       set MAVEN_OPTS=-Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000
#   MAVEN_SKIP_RC - flag to disable loading of mavenrc files
# ----------------------------------------------------------------------------
if [ -z "$MAVEN_SKIP_RC" ] ; then
  if [ -f /usr/local/etc/mavenrc ] ; then
    . /usr/local/etc/mavenrc
  fi
  if [ -f /etc/mavenrc ] ; then
    . /etc/mavenrc
  fi
  if [ -f "$HOME/.mavenrc" ] ; then
    . "$HOME/.mavenrc"
  fi
fi
# OS specific support.  $var _must_ be set to either true or false.
cygwin=false;
darwin=false;
mingw=false
case "`uname`" in
  CYGWIN*) cygwin=true ;;
  MINGW*) mingw=true;;
  Darwin*) darwin=true
    # Use /usr/libexec/java_home if available, otherwise fall back to /
Library/Java/Home
    # See https://developer.apple.com/library/mac/qa/qa1170/_index.html
    if [ -z "$JAVA_HOME" ]; then
```

```
      if [ -x "/usr/libexec/java_home" ]; then
        export JAVA_HOME="`/usr/libexec/java_home`"
      else
        export JAVA_HOME="/Library/Java/Home"
      fi
    fi
    ;;
esac
if [ -z "$JAVA_HOME" ] ; then
  if [ -r /etc/gentoo-release ] ; then
    JAVA_HOME=`java-config --jre-home`
  fi
fi
if [ -z "$M2_HOME" ] ; then
  ## resolve links - $0 may be a link to maven's home
  PRG="$0"
  # need this for relative symlinks
  while [ -h "$PRG" ] ; do
    ls=`ls -ld "$PRG"`
    link=`expr "$ls" : '.*-> \(.*\)$'`
    if expr "$link" : '/.*' > /dev/null; then
      PRG="$link"
    else
      PRG="`dirname "$PRG"`/$link"
    fi
  done
  saveddir=`pwd`
  M2_HOME=`dirname "$PRG"`/..
  # make it fully qualified
  M2_HOME=`cd "$M2_HOME" && pwd`
  cd "$saveddir"
  # echo Using m2 at $M2_HOME
fi
# For Cygwin, ensure paths are in UNIX format before anything is touched
if $cygwin ; then
  [ -n "$M2_HOME" ] &&
    M2_HOME=`cygpath --unix "$M2_HOME"`
  [ -n "$JAVA_HOME" ] &&
    JAVA_HOME=`cygpath --unix "$JAVA_HOME"`
  [ -n "$CLASSPATH" ] &&
    CLASSPATH=`cygpath --path --unix "$CLASSPATH"`
fi
# For Mingw, ensure paths are in UNIX format before anything is touched
if $mingw ; then
  [ -n "$M2_HOME" ] &&
    M2_HOME="`(cd "$M2_HOME"; pwd)`"
  [ -n "$JAVA_HOME" ] &&
    JAVA_HOME="`(cd "$JAVA_HOME"; pwd)`"
fi
if [ -z "$JAVA_HOME" ]; then
  javaExecutable="`which javac`"
  if [ -n "$javaExecutable" ] && ! [ "`expr \"$javaExecutable\" :
'\([^ ]*\)'`" = "no" ]; then
    # readlink(1) is not available as standard on Solaris 10.
    readLink=`which readlink`
    if [ ! `expr "$readLink" : '\([^ ]*\)'` = "no" ]; then
      if $darwin ; then
        javaHome="`dirname \"$javaExecutable\"`"
        javaExecutable="`cd \"$javaHome\" && pwd -P`/javac"
      else
        javaExecutable="`readlink -f \"$javaExecutable\"`"
```

```
      fi
      javaHome="`dirname \"$javaExecutable\"`"
      javaHome=`expr "$javaHome" : '\(.*\)/bin'`
      JAVA_HOME="$javaHome"
      export JAVA_HOME
    fi
  fi
fi
if [ -z "$JAVACMD" ] ; then
  if [ -n "$JAVA_HOME"  ] ; then
    if [ -x "$JAVA_HOME/jre/sh/java" ] ; then
      # IBM's JDK on AIX uses strange locations for the executables
      JAVACMD="$JAVA_HOME/jre/sh/java"
    else
      JAVACMD="$JAVA_HOME/bin/java"
    fi
  else
    JAVACMD="`\\unset -f command; \\command -v java`"
  fi
fi
if [ ! -x "$JAVACMD" ] ; then
  echo "Error: JAVA_HOME is not defined correctly." >&2
  echo "  We cannot execute $JAVACMD" >&2
  exit 1
fi
if [ -z "$JAVA_HOME" ] ; then
  echo "Warning: JAVA_HOME environment variable is not set."
fi
CLASSWORLDS_LAUNCHER=org.codehaus.plexus.classworlds.launcher.Launcher
# traverses directory structure from process work directory to filesystem root
# first directory with .mvn subdirectory is considered project base directory
find_maven_basedir() {
  if [ -z "$1" ]
  then
    echo "Path not specified to find_maven_basedir"
    return 1
  fi
  basedir="$1"
  wdir="$1"
  while [ "$wdir" != '/' ] ; do
    if [ -d "$wdir"/.mvn ] ; then
      basedir=$wdir
      break
    fi
    # workaround for JBEAP-8937 (on Solaris 10/Sparc)
    if [ -d "${wdir}" ]; then
      wdir=`cd "$wdir/.."; pwd`
    fi
    # end of workaround
  done
  echo "${basedir}"
}
# concatenates all lines of a file
concat_lines() {
  if [ -f "$1" ]; then
    echo "$(tr -s '\n' ' ' < "$1")"
  fi
}
BASE_DIR=`find_maven_basedir "$(pwd)"`
if [ -z "$BASE_DIR" ]; then
  exit 1;
```

```
fi
##############################################################################
############
# Extension to allow automatically downloading the maven-wrapper.jar from
Maven-central
# This allows using the maven wrapper in projects that prohibit checking in
binary data.
##############################################################################
############
if [ -r "$BASE_DIR/.mvn/wrapper/maven-wrapper.jar" ]; then
    if [ "$MVNW_VERBOSE" = true ]; then
      echo "Found .mvn/wrapper/maven-wrapper.jar"
    fi
else
    if [ "$MVNW_VERBOSE" = true ]; then
      echo "Couldn't find .mvn/wrapper/maven-wrapper.jar, downloading it ..."
    fi
    if [ -n "$MVNW_REPOURL" ]; then
      jarUrl="$MVNW_REPOURL/org/apache/maven/wrapper/maven-wrapper/3.1.0/
maven-wrapper-3.1.0.jar"
    else
      jarUrl="https://repo.maven.apache.org/maven2/org/apache/maven/wrapper/
maven-wrapper/3.1.0/maven-wrapper-3.1.0.jar"
    fi
    while IFS="=" read key value; do
      case "$key" in (wrapperUrl) jarUrl="$value"; break ;;
      esac
    done < "$BASE_DIR/.mvn/wrapper/maven-wrapper.properties"
    if [ "$MVNW_VERBOSE" = true ]; then
      echo "Downloading from: $jarUrl"
    fi
    wrapperJarPath="$BASE_DIR/.mvn/wrapper/maven-wrapper.jar"
    if $cygwin; then
      wrapperJarPath=`cygpath --path --windows "$wrapperJarPath"`
    fi
    if command -v wget > /dev/null; then
        if [ "$MVNW_VERBOSE" = true ]; then
          echo "Found wget ... using wget"
        fi
        if [ -z "$MVNW_USERNAME" ] || [ -z "$MVNW_PASSWORD" ]; then
            wget "$jarUrl" -O "$wrapperJarPath" || rm -f "$wrapperJarPath"
        else
            wget --http-user=$MVNW_USERNAME --http-password=$MVNW_PASSWORD
"$jarUrl" -O "$wrapperJarPath" || rm -f "$wrapperJarPath"
        fi
    elif command -v curl > /dev/null; then
        if [ "$MVNW_VERBOSE" = true ]; then
          echo "Found curl ... using curl"
        fi
        if [ -z "$MVNW_USERNAME" ] || [ -z "$MVNW_PASSWORD" ]; then
            curl -o "$wrapperJarPath" "$jarUrl" -f
        else
            curl --user $MVNW_USERNAME:$MVNW_PASSWORD -o "$wrapperJarPath"
"$jarUrl" -f
        fi
    else
        if [ "$MVNW_VERBOSE" = true ]; then
          echo "Falling back to using Java to download"
        fi
        javaClass="$BASE_DIR/.mvn/wrapper/MavenWrapperDownloader.java"
        # For Cygwin, switch paths to Windows format before running javac
```

```
        if $cygwin; then
          javaClass=`cygpath --path --windows "$javaClass"`
        fi
        if [ -e "$javaClass" ]; then
            if [ ! -e "$BASE_DIR/.mvn/wrapper/
MavenWrapperDownloader.class" ]; then
                if [ "$MVNW_VERBOSE" = true ]; then
                  echo " - Compiling MavenWrapperDownloader.java ..."
                fi
                # Compiling the Java class
                ("$JAVA_HOME/bin/javac" "$javaClass")
            fi
            if [ -e "$BASE_DIR/.mvn/wrapper/MavenWrapperDownloader.class" ];
then
                # Running the downloader
                if [ "$MVNW_VERBOSE" = true ]; then
                  echo " - Running MavenWrapperDownloader.java ..."
                fi
                ("$JAVA_HOME/bin/java" -cp .mvn/wrapper
MavenWrapperDownloader "$MAVEN_PROJECTBASEDIR")
            fi
        fi
    fi
fi
##########################################################################################
#############
# End of extension
##########################################################################################
#############
export MAVEN_PROJECTBASEDIR=${MAVEN_BASEDIR:-"$BASE_DIR"}
if [ "$MVNW_VERBOSE" = true ]; then
  echo $MAVEN_PROJECTBASEDIR
fi
MAVEN_OPTS="$(concat_lines "$MAVEN_PROJECTBASEDIR/.mvn/jvm.config")
$MAVEN_OPTS"
# For Cygwin, switch paths to Windows format before running java
if $cygwin; then
  [ -n "$M2_HOME" ] &&
    M2_HOME=`cygpath --path --windows "$M2_HOME"`
  [ -n "$JAVA_HOME" ] &&
    JAVA_HOME=`cygpath --path --windows "$JAVA_HOME"`
  [ -n "$CLASSPATH" ] &&
    CLASSPATH=`cygpath --path --windows "$CLASSPATH"`
  [ -n "$MAVEN_PROJECTBASEDIR" ] &&
    MAVEN_PROJECTBASEDIR=`cygpath --path --windows "$MAVEN_PROJECTBASEDIR"`
fi
# Provide a "standardized" way to retrieve the CLI args that will
# work with both Windows and non-Windows executions.
MAVEN_CMD_LINE_ARGS="$MAVEN_CONFIG $@"
export MAVEN_CMD_LINE_ARGS
WRAPPER_LAUNCHER=org.apache.maven.wrapper.MavenWrapperMain
exec "$JAVACMD" \
  $MAVEN_OPTS \
  $MAVEN_DEBUG_OPTS \
  -classpath "$MAVEN_PROJECTBASEDIR/.mvn/wrapper/maven-wrapper.jar" \
  "-Dmaven.home=${M2_HOME}" \
  "-Dmaven.multiModuleProjectDirectory=${MAVEN_PROJECTBASEDIR}" \
  ${WRAPPER_LAUNCHER} $MAVEN_CONFIG "$@"
```

spring-boot-example\mvnw.cmd

```
@REM
----------------------------------------------------------------------------
@REM Licensed to the Apache Software Foundation (ASF) under one
@REM or more contributor license agreements.  See the NOTICE file
@REM distributed with this work for additional information
@REM regarding copyright ownership.  The ASF licenses this file
@REM to you under the Apache License, Version 2.0 (the
@REM "License"); you may not use this file except in compliance
@REM with the License.  You may obtain a copy of the License at
@REM
@REM    https://www.apache.org/licenses/LICENSE-2.0
@REM
@REM Unless required by applicable law or agreed to in writing,
@REM software distributed under the License is distributed on an
@REM "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
@REM KIND, either express or implied.  See the License for the
@REM specific language governing permissions and limitations
@REM under the License.
@REM
----------------------------------------------------------------------------
@REM
----------------------------------------------------------------------------
@REM Maven Start Up Batch script
@REM
@REM Required ENV vars:
@REM JAVA_HOME - location of a JDK home dir
@REM
@REM Optional ENV vars
@REM M2_HOME - location of maven2's installed home dir
@REM MAVEN_BATCH_ECHO - set to 'on' to enable the echoing of the batch
commands
@REM MAVEN_BATCH_PAUSE - set to 'on' to wait for a keystroke before ending
@REM MAVEN_OPTS - parameters passed to the Java VM when running Maven
@REM     e.g. to debug Maven itself, use
@REM set MAVEN_OPTS=-Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000
@REM MAVEN_SKIP_RC - flag to disable loading of mavenrc files
@REM
----------------------------------------------------------------------------
@REM Begin all REM lines with '@' in case MAVEN_BATCH_ECHO is 'on'
@echo off
@REM set title of command window
title %0
@REM enable echoing by setting MAVEN_BATCH_ECHO to 'on'
@if "%MAVEN_BATCH_ECHO%" == "on"  echo %MAVEN_BATCH_ECHO%
@REM set %HOME% to equivalent of $HOME
if "%HOME%" == "" (set "HOME=%HOMEDRIVE%%HOMEPATH%")
@REM Execute a user defined script before this one
if not "%MAVEN_SKIP_RC%" == "" goto skipRcPre
@REM check for pre script, once with legacy .bat ending and once with .cmd
ending
if exist "%USERPROFILE%\mavenrc_pre.bat" call "%USERPROFILE%\mavenrc_pre.bat"
%*
if exist "%USERPROFILE%\mavenrc_pre.cmd" call "%USERPROFILE%\mavenrc_pre.cmd"
%*
:skipRcPre
@setlocal
set ERROR_CODE=0
```

```
@REM To isolate internal variables from possible post scripts, we use another
setlocal
@setlocal
@REM ==== START VALIDATION ====
if not "%JAVA_HOME%" == "" goto OkJHome
echo.
echo Error: JAVA_HOME not found in your environment. >&2
echo Please set the JAVA_HOME variable in your environment to match the >&2
echo location of your Java installation. >&2
echo.
goto error
:OkJHome
if exist "%JAVA_HOME%\bin\java.exe" goto init
echo.
echo Error: JAVA_HOME is set to an invalid directory. >&2
echo JAVA_HOME = "%JAVA_HOME%" >&2
echo Please set the JAVA_HOME variable in your environment to match the >&2
echo location of your Java installation. >&2
echo.
goto error
@REM ==== END VALIDATION ====
:init
@REM Find the project base dir, i.e. the directory that contains the folder
".mvn".
@REM Fallback to current working directory if not found.
set MAVEN_PROJECTBASEDIR=%MAVEN_BASEDIR%
IF NOT "%MAVEN_PROJECTBASEDIR%"=="" goto endDetectBaseDir
set EXEC_DIR=%CD%
set WDIR=%EXEC_DIR%
:findBaseDir
IF EXIST "%WDIR%"\.mvn goto baseDirFound
cd ..
IF "%WDIR%"=="%CD%" goto baseDirNotFound
set WDIR=%CD%
goto findBaseDir
:baseDirFound
set MAVEN_PROJECTBASEDIR=%WDIR%
cd "%EXEC_DIR%"
goto endDetectBaseDir
:baseDirNotFound
set MAVEN_PROJECTBASEDIR=%EXEC_DIR%
cd "%EXEC_DIR%"
:endDetectBaseDir
IF NOT EXIST "%MAVEN_PROJECTBASEDIR%\.mvn\jvm.config" goto
endReadAdditionalConfig
@setlocal EnableExtensions EnableDelayedExpansion
for /F "usebackq delims=" %%a in ("%MAVEN_PROJECTBASEDIR%\.mvn\jvm.config")
do set JVM_CONFIG_MAVEN_PROPS=!JVM_CONFIG_MAVEN_PROPS! %%a
@endlocal & set JVM_CONFIG_MAVEN_PROPS=%JVM_CONFIG_MAVEN_PROPS%
:endReadAdditionalConfig
SET MAVEN_JAVA_EXE="%JAVA_HOME%\bin\java.exe"
set WRAPPER_JAR="%MAVEN_PROJECTBASEDIR%\.mvn\wrapper\maven-wrapper.jar"
set WRAPPER_LAUNCHER=org.apache.maven.wrapper.MavenWrapperMain
set DOWNLOAD_URL="https://repo.maven.apache.org/maven2/org/apache/maven/
wrapper/maven-wrapper/3.1.0/maven-wrapper-3.1.0.jar"
FOR /F "usebackq tokens=1,2 delims==" %%A IN ("%MAVEN_PROJECTBASEDIR%
\.mvn\wrapper\maven-wrapper.properties") DO (
    IF "%%A"=="wrapperUrl" SET DOWNLOAD_URL=%%B
)
@REM Extension to allow automatically downloading the maven-wrapper.jar from
Maven-central
```

```
@REM This allows using the maven wrapper in projects that prohibit checking
in binary data.
if exist %WRAPPER_JAR% (
    if "%MVNW_VERBOSE%" == "true" (
        echo Found %WRAPPER_JAR%
    )
) else (
    if not "%MVNW_REPOURL%" == "" (
        SET DOWNLOAD_URL="%MVNW_REPOURL%/org/apache/maven/wrapper/maven-
wrapper/3.1.0/maven-wrapper-3.1.0.jar"
    )
    if "%MVNW_VERBOSE%" == "true" (
        echo Couldn't find %WRAPPER_JAR%, downloading it ...
        echo Downloading from: %DOWNLOAD_URL%
    )
    powershell -Command "&{"^
™"$webclient = new-object System.Net.WebClient;"^
™"if (-not ([string]::IsNullOrEmpty('%MVNW_USERNAME%') -and
[string]::IsNullOrEmpty('%MVNW_PASSWORD%'))) {"^
™"$webclient.Credentials = new-object
System.Net.NetworkCredential('%MVNW_USERNAME%', '%MVNW_PASSWORD%');"^
™"}"^
™"[Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::Tls12; $webclient.DownloadFile('%DOWNLOAD_URL%',
'%WRAPPER_JAR%')"^
™"}"
    if "%MVNW_VERBOSE%" == "true" (
        echo Finished downloading %WRAPPER_JAR%
    )
)
@REM End of extension
@REM Provide a "standardized" way to retrieve the CLI args that will
@REM work with both Windows and non-Windows executions.
set MAVEN_CMD_LINE_ARGS=%*
%MAVEN_JAVA_EXE% ^
  %JVM_CONFIG_MAVEN_PROPS% ^
  %MAVEN_OPTS% ^
  %MAVEN_DEBUG_OPTS% ^
  -classpath %WRAPPER_JAR% ^
  "-Dmaven.multiModuleProjectDirectory=%MAVEN_PROJECTBASEDIR%" ^
  %WRAPPER_LAUNCHER% %MAVEN_CONFIG% %*
if ERRORLEVEL 1 goto error
goto end
:error
set ERROR_CODE=1
:end
@endlocal & set ERROR_CODE=%ERROR_CODE%
if not "%MAVEN_SKIP_RC%"=="" goto skipRcPost
@REM check for post script, once with legacy .bat ending and once with .cmd
ending
if exist "%USERPROFILE%\mavenrc_post.bat" call "%USERPROFILE%
\mavenrc_post.bat"
if exist "%USERPROFILE%\mavenrc_post.cmd" call "%USERPROFILE%
\mavenrc_post.cmd"
:skipRcPost
@REM pause the script if MAVEN_BATCH_PAUSE is set to 'on'
if "%MAVEN_BATCH_PAUSE%"=="on" pause
if "%MAVEN_TERMINATE_CMD%"=="on" exit %ERROR_CODE%
cmd /C exit /B %ERROR_CODE%
```

spring-boot-example\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.14</version>
        <relativePath/>
    </parent>
    <groupId>dev.langchain4j</groupId>
    <artifactId>spring-boot-example</artifactId>
    <version>0.23.0</version>
    <properties>
        <java.version>1.8</java.version>
        <!-- due to vulnerabilities in version 1.X (transitive dependency of
spring boot) -->
        <snakeyaml.version>2.1</snakeyaml.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-spring-boot-starter</artifactId>
            <version>0.23.0</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>0.23.0</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

spring-boot-example\src\main\java\dev\example\Booking.java

```
package dev.example;
import java.time.LocalDate;
public class Booking {
    private String bookingNumber;
    private LocalDate bookingFrom;
    private LocalDate bookingTo;
    private Customer customer;
    public Booking(String bookingNumber, LocalDate bookingFrom, LocalDate
bookingTo, Customer customer) {
        this.bookingNumber = bookingNumber;
        this.bookingFrom = bookingFrom;
        this.bookingTo = bookingTo;
        this.customer = customer;
    }
}
```

spring-boot-example\src\main\java\dev\example\BookingCannotBeCancelledException.java

```
package dev.example;
public class BookingCannotBeCancelledException extends RuntimeException {
    public BookingCannotBeCancelledException(String bookingNumber) {
        super("Booking " + bookingNumber + " cannot be canceled");
    }
}
```

spring-boot-example\src\main\java\dev\example\BookingNotFoundException.java

```
package dev.example;
public class BookingNotFoundException extends RuntimeException {
    public BookingNotFoundException(String bookingNumber) {
        super("Booking " + bookingNumber + " not found");
    }
}
```

spring-boot-example\src\main\java\dev\example\BookingService.java

```
package dev.example;
import org.springframework.stereotype.Component;
import java.time.LocalDate;
@Component
public class BookingService {
    public Booking getBookingDetails(String bookingNumber, String
customerName, String customerSurname) {
        ensureExists(bookingNumber, customerName, customerSurname);
        // Imitating retrieval from DB
        LocalDate bookingFrom = LocalDate.now().plusDays(1);
        LocalDate bookingTo = LocalDate.now().plusDays(3);
        Customer customer = new Customer(customerName, customerSurname);
        return new Booking(bookingNumber, bookingFrom, bookingTo, customer);
    }
    public void cancelBooking(String bookingNumber, String customerName,
String customerSurname) {
        ensureExists(bookingNumber, customerName, customerSurname);
        // Imitating cancellation
        throw new BookingCannotBeCancelledException(bookingNumber);
    }
    private void ensureExists(String bookingNumber, String customerName,
String customerSurname) {
        // Imitating check
        if (!(bookingNumber.equals("123-456")
                && customerName.equals("Klaus")
                && customerSurname.equals("Heisler"))) {
            throw new BookingNotFoundException(bookingNumber);
        }
    }
}
```

spring-boot-example\src\main\java\dev\example\BookingTools.java

```
package dev.example;
import dev.langchain4j.agent.tool.Tool;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class BookingTools {
    @Autowired
    private BookingService bookingService;
    @Tool
    public Booking getBookingDetails(String bookingNumber, String
customerName, String customerSurname) {
        System.out.println("==========================================
======================================");
        System.out.printf("[Tool]: Getting details for booking %s for %s %s...
%n", bookingNumber, customerName, customerSurname);
        System.out.println("==========================================
======================================");
        return bookingService.getBookingDetails(bookingNumber, customerName,
customerSurname);
    }
    @Tool
    public void cancelBooking(String bookingNumber, String customerName,
String customerSurname) {
        System.out.println("==========================================
======================================");
        System.out.printf("[Tool]: Cancelling booking %s for %s %s...%n",
bookingNumber, customerName, customerSurname);
        System.out.println("==========================================
======================================");
        bookingService.cancelBooking(bookingNumber, customerName,
customerSurname);
    }
}
```

spring-boot-example\src\main\java\dev\example\Customer.java

```
package dev.example;
public class Customer {
    private String name;
    private String surname;
    public Customer(String name, String surname) {
        this.name = name;
        this.surname = surname;
    }
}
```

spring-boot-example\src\main\java\dev\example\CustomerSupportAgent.java

```
package dev.example;
import dev.langchain4j.service.SystemMessage;
interface CustomerSupportAgent {
    @SystemMessage({
            "You are a customer support agent of a car rental company named
'Miles of Smiles'.",
            "Before providing information about booking or cancelling
booking, you MUST always check:",
            "booking number, customer name and surname.",
            "Today is {{current_date}}."
    })
    String chat(String userMessage);
}
```

spring-boot-example\src\main\java\dev\example\CustomerSupportApplication.java

```
package dev.example;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.DocumentSplitter;
import dev.langchain4j.data.document.splitter.DocumentSplitters;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.memory.chat.MessageWindowChatMemory;
import dev.langchain4j.model.chat.ChatLanguageModel;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.openai.OpenAiTokenizer;
import dev.langchain4j.retriever.EmbeddingStoreRetriever;
import dev.langchain4j.retriever.Retriever;
import dev.langchain4j.service.AiServices;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.EmbeddingStoreIngestor;
import dev.langchain4j.store.embedding.inmemory.InMemoryEmbeddingStore;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.Resource;
import org.springframework.core.io.ResourceLoader;
import java.io.IOException;
import static
dev.langchain4j.data.document.FileSystemDocumentLoader.loadDocument;
import static dev.langchain4j.model.openai.OpenAiModelName.GPT_3_5_TURBO;
@SpringBootApplication
public class CustomerSupportApplication {
    @Bean
    CustomerSupportAgent customerSupportAgent(ChatLanguageModel
chatLanguageModel,
                                              BookingTools bookingTools,
                                              Retriever<TextSegment>
retriever) {
        return AiServices.builder(CustomerSupportAgent.class)
                .chatLanguageModel(chatLanguageModel)
                .chatMemory(MessageWindowChatMemory.withMaxMessages(20))
                .tools(bookingTools)
                .retriever(retriever)
                .build();
    }
    @Bean
    Retriever<TextSegment> retriever(EmbeddingStore<TextSegment>
embeddingStore, EmbeddingModel embeddingModel) {
        // You will need to adjust these parameters to find the optimal
setting, which will depend on two main factors:
        // - The nature of your data
        // - The embedding model you are using
        int maxResultsRetrieved = 1;
        double minScore = 0.6;
        return EmbeddingStoreRetriever.from(embeddingStore, embeddingModel,
maxResultsRetrieved, minScore);
    }
    @Bean
    EmbeddingModel embeddingModel() {
        return new AllMiniLmL6V2EmbeddingModel();
    }
    @Bean
    EmbeddingStore<TextSegment> embeddingStore(EmbeddingModel embeddingModel,
```

```
ResourceLoader resourceLoader) throws IOException {
        // Normally, you would already have your embedding store filled with
your data.
        // However, for the purpose of this demonstration, we will:
        // 1. Create an in-memory embedding store
        EmbeddingStore<TextSegment> embeddingStore = new
InMemoryEmbeddingStore<>();
        // 2. Load an example document ("Miles of Smiles" terms of use)
        Resource resource = resourceLoader.getResource("classpath:miles-of-
smiles-terms-of-use.txt");
        Document document = loadDocument(resource.getFile().toPath());
        // 3. Split the document into segments 100 tokens each
        // 4. Convert segments into embeddings
        // 5. Store embeddings into embedding store
        // All this can be done manually, but we will use
EmbeddingStoreIngestor to automate this:
        DocumentSplitter documentSplitter = DocumentSplitters.recursive(100,
0, new OpenAiTokenizer(GPT_3_5_TURBO));
        EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
                .documentSplitter(documentSplitter)
                .embeddingModel(embeddingModel)
                .embeddingStore(embeddingStore)
                .build();
        ingestor.ingest(document);
        return embeddingStore;
    }
    public static void main(String[] args) {
        SpringApplication.run(CustomerSupportApplication.class, args);
    }
}
```

spring-boot-example\src\main\resources\application.properties

```
langchain4j.chat-model.provider=openai
langchain4j.chat-model.openai.api-key=demo
langchain4j.chat-model.openai.model-name=gpt-3.5-turbo
langchain4j.chat-model.openai.temperature=0.0
#langchain4j.chat-model.openai.top-p=1.0
#langchain4j.chat-model.openai.max-tokens=100
#langchain4j.chat-model.openai.presence-penalty=0.0
#langchain4j.chat-model.openai.frequency-penalty=0.0
langchain4j.chat-model.openai.timeout=PT60S
#langchain4j.chat-model.openai.max-retries=3
#langchain4j.chat-model.openai.log-requests=true
#langchain4j.chat-model.openai.log-responses=true
logging.level.dev.langchain4j=DEBUG
logging.level.dev.ai4j.openai4j=DEBUG
```

spring-boot-example\src\main\resources\miles-of-smiles-terms-of-use.txt

Miles of Smiles Car Rental Services Terms of Use

1. Introduction

These Terms of Service (“Terms”) govern the access or use by you, an individual, from within any country in the world, of applications, websites, content, products, and services (“Services”) made available by Miles of Smiles Car Rental Services, a company registered in the United States of America.

2. The Services

Miles of Smiles rents out vehicles to the end user. We reserve the right to temporarily or permanently discontinue the Services at any time and are not liable for any modification, suspension or discontinuation of the Services.

3. Bookings

3.1 Users may make a booking through our website or mobile application.

3.2 You must provide accurate, current and complete information during the reservation process. You are responsible for all charges incurred under your account.

3.3 All bookings are subject to vehicle availability.

4. Cancellation Policy

4.1 Reservations can be cancelled up to 7 days prior to the start of the booking period.

4.2 If the booking period is less than 3 days, cancellations are not permitted.

5. Use of Vehicle

5.1 All cars rented from Miles of Smiles must not be used:

for any illegal purpose or in connection with any criminal offense.

for teaching someone to drive.

in any race, rally or contest.

while under the influence of alcohol or drugs.

6. Liability

6.1 Users will be held liable for any damage, loss, or theft that occurs during the rental period.

6.2 We do not accept liability for any indirect or consequential loss, damage, or expense including but not limited to loss of profits.

7. Governing Law

These terms will be governed by and construed in accordance with the laws of the United States of America, and any disputes relating to these terms will be subject to the exclusive jurisdiction of the courts of United States.

8. Changes to These Terms

We may revise these terms of use at any time by amending this page. You are expected to check this page from time to time to take notice of any changes we made.

9. Acceptance of These Terms

By using the Services, you acknowledge that you have read and understand these Terms and agree to be bound by them.

If you do not agree to these Terms, please do not use or access our Services.

spring-boot-example\src\test\java\dev\example\CustomerSupportApplicationTest.java

```
package dev.example;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest
class CustomerSupportApplicationTest {
    @Autowired
    CustomerSupportAgent agent;
    @Test
    void
should_provide_booking_details_and_explain_why_cancellation_is_not_possible()
{
        // Please define API keys in application.properties before running
this test.
        // Tip: Use gpt-4 for this example, as gpt-3.5-turbo tends to
hallucinate often and invent name and surname.
        interact(agent, "Hi, I forgot when my booking is.");
        interact(agent, "123-457");
        interact(agent, "I'm sorry I'm so inattentive today. Klaus Heisler.");
        interact(agent, "My bad, it's 123-456");
        // Here, information about the cancellation policy is automatically
retrieved and injected into the prompt.
        // Although the LLM sometimes attempts to cancel the booking, it
fails to do so and will explain
        // the reason why the booking cannot be cancelled, based on the
injected cancellation policy.
        interact(agent, "My plans have changed, can I cancel my booking?");
    }
    private static void interact(CustomerSupportAgent agent, String
userMessage) {
        System.out.println("================================================
========================================");
        System.out.println("[User]: " + userMessage);
        System.out.println("================================================
========================================");
        String agentAnswer = agent.chat(userMessage);
        System.out.println("================================================
========================================");
        System.out.println("[Agent]: " + agentAnswer);
        System.out.println("================================================
========================================");
    }
}
```

vespa-example\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>vespa-example</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-vespa</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</project>
```

vespa-example\src\main\java\README.md

````
# Server side setup with Maven to run example on Vespa Cloud
1. Do steps 1 - 4 from https://cloud.vespa.ai/en/getting-started-java.html
2. Create new Maven Java application and go to it
3. Add following dependency to `pom.xml`, so the latest Vespa release is used
on each build:
```xml
<parent>
    <groupId>com.yahoo.vespa</groupId>
    <artifactId>cloud-tenant-base</artifactId>
    <version>[8,9)</version>
    <relativePath/>
</parent>
```
4. Create `src/main/application/schemas/langchain4j.sd` with following
content:
```sd
schema langchain4j {
  document langchain4j {
    field text_segment type string {
      indexing: summary | index
    }
    field vector type tensor<float>(x[384]) {
      indexing: summary | attribute
      attribute {
        distance-metric: prenormalized-angular
      }
    }
  }
  rank-profile cosine_similarity {
    inputs {
      query(q) tensor<float>(x[384])
    }
    function cosine() {
      expression: 1.0 - distance(field, vector)
    }
    first-phase {
      expression: if (cosine() < query(threshold), -1, cosine())
      rank-score-drop-limit: 0.0
    }
  }
}
```
5. Create `src/main/application/services.xml` with following content:
```xml
<?xml version="1.0" encoding="utf-8" ?>
<services version="1.0">
    <container version="1.0" id="default">
        <document-api/>
        <search/>
    </container>
    <content id="langchain4j" version="1.0">
        <redundancy>1</redundancy>
        <documents>
            <document type="langchain4j" mode="index"/>
        </documents>
    </content>
</services>
```
````

6. Do steps 6 from https://cloud.vespa.ai/en/getting-started-java.html. This will generate SSL private key and
   public certificate and store them locally. Their path will be used in client example. Certificate will be added
   to the project under `src/main/application/security/clients.pem`.
7. Now you can build (you need Java > 17 for this) and deploy your application: do steps 7 & 8 from
   https://cloud.vespa.ai/en/getting-started.html.
   Alternatively you can deploy your build application by using [Cloud UI](https://console.vespa-cloud.com) and
   uploading `target/application.zip` there.

vespa-example\src\main\java\VespaEmbeddingStoreExample.java

```
import static java.util.Arrays.asList;
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.vespa.VespaEmbeddingStore;
import java.util.List;
/**
 * Example of integration with Vespa. You need to configure Vespa server side first, instructions are
 * inside of README.md file.
 */
public class VespaEmbeddingStoreExample {
  public static void main(String[] args) {
    EmbeddingStore<TextSegment> embeddingStore = VespaEmbeddingStore
      .builder()
      // server url, e.g. https://alexey-heezer.langchain4j.mytenant346.aws-us-east-1c.dev.z.vespa-app.cloud
      .url("url")
      // local path to the SSL private key file,
      // e.g. /Users/user/.vespa/mytenant346.langchain4j.alexey-heezer/data-plane-private-key.pem
      .keyPath("keyPath")
      // local path to the SSL certificate file,
      // e.g. /Users/user/.vespa/mytenant346.langchain4j.alexey-heezer/data-plane-public-cert.pem
      .certPath("certPath")
      .build();
    EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
    TextSegment segment1 = TextSegment.from("I like football.");
    Embedding embedding1 = embeddingModel.embed(segment1).content();
    embeddingStore.add(embedding1, segment1);
    TextSegment segment2 = TextSegment.from("I've never been to New York.");
    Embedding embedding2 = embeddingModel.embed(segment2).content();
    embeddingStore.add(embedding2, segment2);
    TextSegment segment3 = TextSegment.from(
      "But actually we tried our new swimming pool yesterday and it was awesome!"
    );
    Embedding embedding3 = embeddingModel.embed(segment3).content();
    embeddingStore.add(embedding3, segment3);
    List<String> ids = embeddingStore.addAll(
      asList(embedding1, embedding2, embedding3),
      asList(segment1, segment2, segment3)
    );
    System.out.println("added/updated records count: " + ids.size()); // 3
    TextSegment segment4 = TextSegment.from(
      "John Lennon was a very cool person."
    );
    Embedding embedding4 = embeddingModel.embed(segment4).content();
    String s4id = embeddingStore.add(embedding4, segment4);
    System.out.println("segment 4 id: " + s4id);
    Embedding queryEmbedding = embeddingModel.embed(
      "What is your favorite sport?"
    ).content();
```

```
    List<EmbeddingMatch<TextSegment>> relevant = embeddingStore.findRelevant(
      queryEmbedding,
      2
    );
    System.out.println(
      "relevant results count for sport question: " + relevant.size()
    ); // 2
    System.out.println(relevant.get(0).score()); // 0.639...
    System.out.println(relevant.get(0).embedded().text()); // football
    System.out.println(relevant.get(1).score()); // 0.232...
    System.out.println(relevant.get(1).embedded().text()); // swimming pool
    queryEmbedding = embeddingModel.embed("And what about
musicians?").content();
    relevant = embeddingStore.findRelevant(queryEmbedding, 5, 0.3);
    System.out.println(
      "relevant results count for music question: " + relevant.size()
    ); // 1
    System.out.println(relevant.get(0).score()); // 0.359...
    System.out.println(relevant.get(0).embedded().text()); // John Lennon
  }
}
```

weaviate-example\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.langchain4j</groupId>
    <artifactId>weaviate-example</artifactId>
    <version>0.23.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-weaviate</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>dev.langchain4j</groupId>
            <artifactId>langchain4j-embeddings-all-minilm-l6-v2</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</project>
```

weaviate-example\src\main\java\WeaviateEmbeddingStoreExample.java

```
import dev.langchain4j.data.embedding.Embedding;
import dev.langchain4j.data.segment.TextSegment;
import dev.langchain4j.model.embedding.AllMiniLmL6V2EmbeddingModel;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingMatch;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.weaviate.WeaviateEmbeddingStore;
import java.util.List;
public class WeaviateEmbeddingStoreExample {
    public static void main(String[] args) {
        EmbeddingStore<TextSegment> embeddingStore =
WeaviateEmbeddingStore.builder()
                // Find it under "Show API keys" of your Weaviate cluster.
                .apiKey(System.getenv("WEAVIATE_API_KEY"))
                // The scheme, e.g. "https" of cluster URL. Find in under
Details of your Weaviate cluster.
                .scheme("https")
                // The host, e.g. "test-o1gvgnp4.weaviate.network" of cluster
URL.
                // Find in under Details of your Weaviate cluster.
                .host("test3-bwsieg9y.weaviate.network")
                // "Default" class is used if not specified. Must start from
an uppercase letter!
                .objectClass("Test")
                // If true (default), then WeaviateEmbeddingStore will
generate a hashed ID based on provided
                // text segment, which avoids duplicated entries in DB. If
false, then random ID will be generated.
                .avoidDups(true)
                // Consistency level: ONE, QUORUM (default) or ALL.
                .consistencyLevel("ALL")
                .build();
        EmbeddingModel embeddingModel = new AllMiniLmL6V2EmbeddingModel();
        TextSegment segment1 = TextSegment.from("I like football.");
        Embedding embedding1 = embeddingModel.embed(segment1).content();
        embeddingStore.add(embedding1, segment1);
        TextSegment segment2 = TextSegment.from("The weather is good today.");
        Embedding embedding2 = embeddingModel.embed(segment2).content();
        embeddingStore.add(embedding2, segment2);
        Embedding queryEmbedding = embeddingModel.embed("What is your
favourite sport?").content();
        List<EmbeddingMatch<TextSegment>> relevant =
embeddingStore.findRelevant(queryEmbedding, 1);
        EmbeddingMatch<TextSegment> embeddingMatch = relevant.get(0);
        System.out.println(embeddingMatch.score()); // 0.8144288063049316
        System.out.println(embeddingMatch.embedded().text()); // I like
football.
    }
}
```