Spring AI

(Also Spring with AI without Spring AI)

Contact Info

Ken Kousen

Kousen IT, Inc.

ken.kousen@kousenit.com
http://www.kousenit.com
https://kousenit.org (blog)
@kenkousen (twitter)
https://foojay.social/@kenkousen (mastodon)

Tales from the jar side (free newsletter)
https://kenkousen.substack.com
https://youtube.com/@talesfromthejarside

OpenAI API

- OpenAI
  - https://chat.openai.com/ → regular ChatGPT
  - https://platform.openai.com → developer API
  - Introduction and guides: https://platform.openai.com/docs/introduction
  - API Reference: https://platform.openai.com/docs/api-reference
- Need to register and generate a key
  - Requires credit card with a small deposit (about $10)
  - Each request costs a small amount (prices discussed later)
- Ollama → can run LLMs on your local machine
  - Also discussed later

OpenAI

- Get a key at https://platform.openai.com/signup
- Set as environment variable
  - SPRING_AI_OPENAI_API_KEY
- Spring reads configuration property
  - `spring.ai.openai.api-key`

Spring AI

- Reference page: https://docs.spring.io/spring-ai/reference/index.html
- Very early version: currently 0.8.0
- Based on LangChain and LlamaIndex, but not a direct port of either
- Aims to generalize to multiple LLMs

Dependencies

```xml
<repositories>
  <repository>
    <id>spring-milestones</id>
    <name>Spring Milestones</name>
    <url>https://repo.spring.io/milestone</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>spring-snapshots</id>
    <name>Spring Snapshots</name>
    <url>https://repo.spring.io/snapshot</url>
    <releases>
      <enabled>false</enabled>
    </releases>
  </repository>
</repositories>
```

```
repositories {
  mavenCentral()
  maven { url 'https://repo.spring.io/milestone' }
  maven { url 'https://repo.spring.io/snapshot' }
}
```

OpenAI Dependencies

```xml
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-openai-spring-boot-starter</artifactId>
    <version>0.8.0-SNAPSHOT</version>
</dependency>
```

```
dependencies {
    implementation 'org.springframework.ai:spring-ai-openai-spring-boot-starter:0.8.0-SNAPSHOT'
}
```

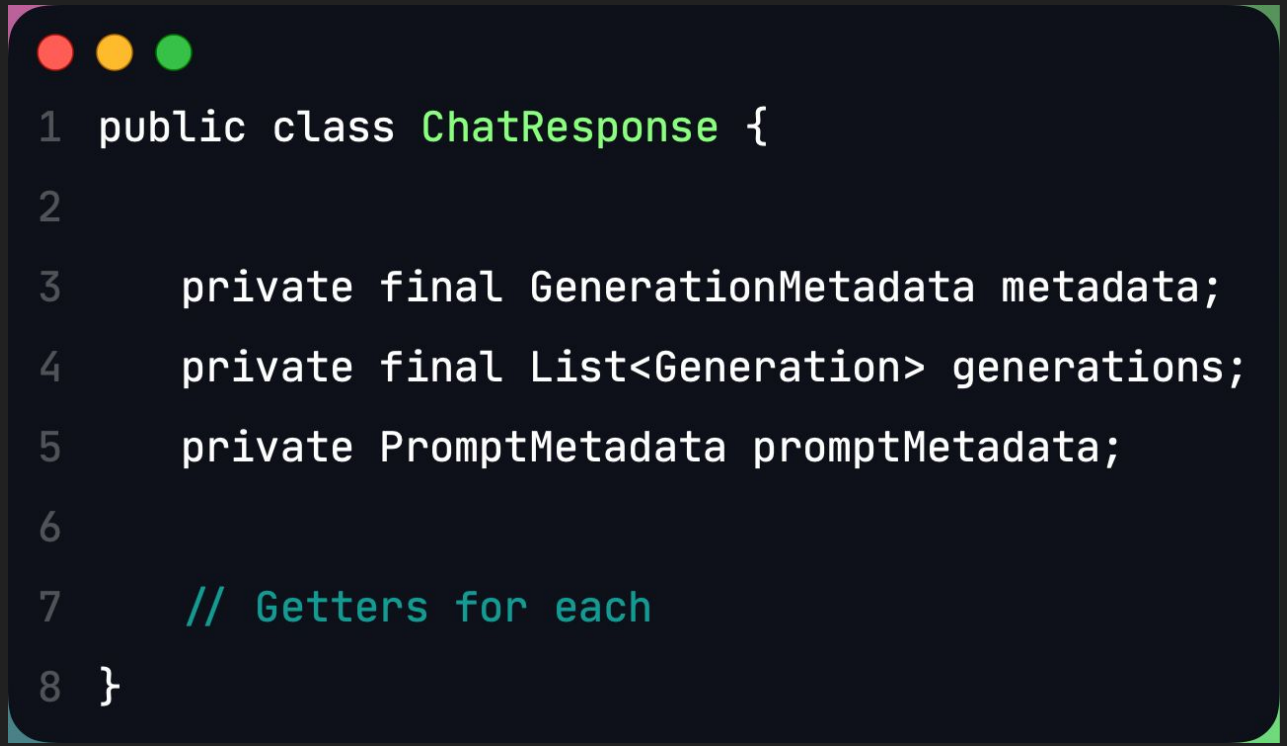
Spring AI

- Major features:
  - General interfaces
  - Prompt Templates
  - Output Parsing
  - Embeddings
  - Retrieval Augmented Generation (RAG)

```java
@FunctionalInterface
public interface ChatClient {

    default String generate(String message) {
        Prompt prompt = new Prompt(new UserMessage(message));
        return generate(prompt).getGeneration().getContent();
    }

    ChatResponse generate(Prompt prompt);
}
```

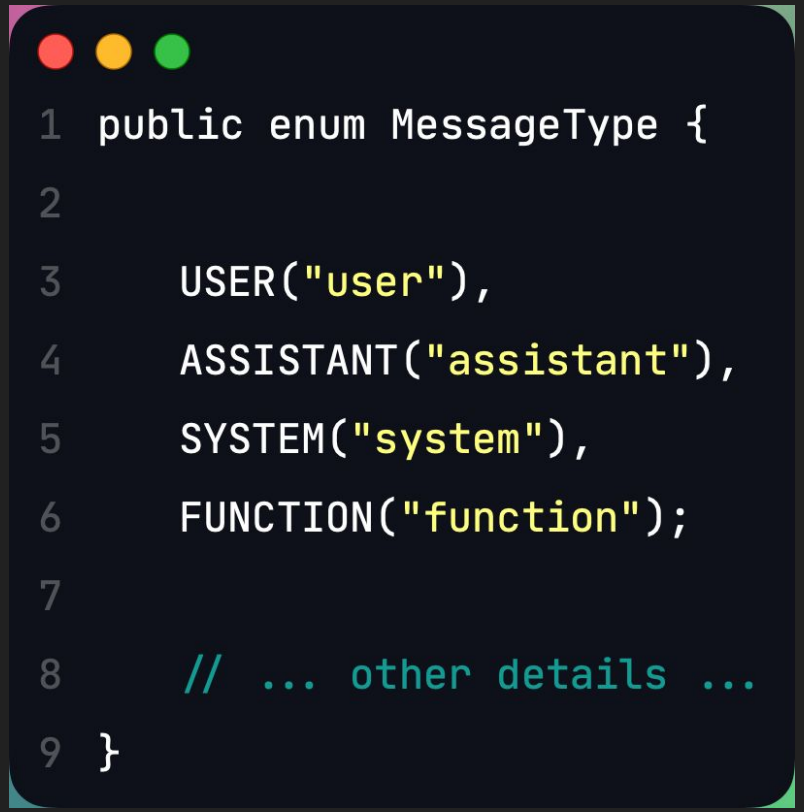

public class ChatResponse {
private final GenerationMetadata metadata;
private final List<Generation> generations;
private PromptMetadata promptMetadata;
// Getters for each
}

```java
public abstract class AbstractMessage implements Message {

    protected String content;
    protected Map<String, Object> properties = new HashMap<>();
    protected MessageType messageType;

    // Getters...
}
```

Message Types

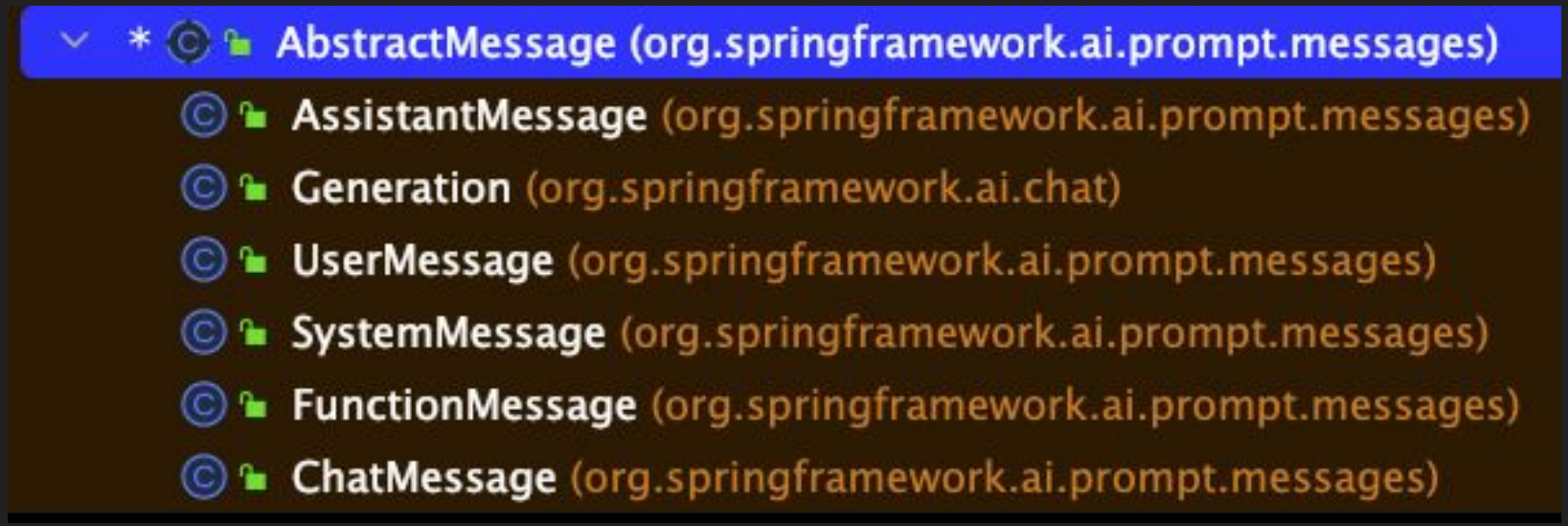


```
public enum MessageType {

    USER("user"),
    ASSISTANT("assistant"),
    SYSTEM("system"),
    FUNCTION("function");

    // ... other details ...
}
```

Message Hierarchy



Messages

- ChatMessage: String role, String content
- UserMessage: String message or Resource
- SystemMessage: String content or Resource

OpenAI Reference

- All based on (at least) OpenAI chat model
- Request:
  - model
  - list of messages, each with role and content
- Response:
  - Complicated, but includes:
  - model
  - list of choices, each with a message
  - finish reason
  - usage, with tokens

```
curl https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "gpt-3.5-turbo",
    "messages": [
      {
        "role": "system",
        "content": "You are a helpful assistant."
      },
      {
        "role": "user",
        "content": "Hello!"
      }
    ]
  }'
```

```
{
  "id": "chatcmpl-123",
  "object": "chat.completion",
  "created": 1677652288,
  "model": "gpt-3.5-turbo-0613",
  "system_fingerprint": "fp_44709d6fcb",
  "choices": [{
    "index": 0,
    "message": {
      "role": "assistant",
      "content": "\n\nHello there, how may I assist you today?",
    },
    "logprobs": null,
    "finish_reason": "stop"
  }],
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21
  }
}
```

Spring AI

- Headers (like Accept and Content-Type) supplied automatically
- Authorization key read from application.properties
  - Usually set as environment variable

Prompts

- Prompts done using messages
  - System message → context
  - User message → actual request

```java
public class Prompt {

    private final List<Message> messages;

    // constructors and utility methods omitted
}
```

Prompt Templates

- Boilerplate with template variables
- Based on StringTemplate open source library, https://www.stringtemplate.org/
- Uses { … } to specify variables

```
Tell me a {adjective} joke about {topic}
```

- Developer supplies a Map of keys and values

Prompt Templates

```
1 PromptTemplate promptTemplate = new PromptTemplate(jokeResource);
2 Prompt prompt = promptTemplate.create(
3         Map.of("adjective", adjective, "topic", topic));
4 chatClient.generate(prompt).getGeneration();
```

```
String userText = """
    Tell me about three famous pirates from the Golden Age of Piracy and why they did.
    Write at least a sentence for each pirate.
    """;

Message userMessage = new UserMessage(userText);

String systemText = """
  You are a helpful AI assistant that helps people find information.
  Your name is {name}
  You should reply to the user's request with your name and also in the style of a {voice}.
  """;

SystemPromptTemplate systemPromptTemplate = new SystemPromptTemplate(systemText);
Message systemMessage = systemPromptTemplate.createMessage(Map.of("name", name, "voice", voice));

Prompt prompt = new Prompt(List.of(userMessage, systemMessage));

List<Generation> response = aiClient.generate(prompt).getGenerations();
```

Spring Resources

- Spring supports the Resource abstraction
- Can put prompts in files and inject them with @Value

Tokens

- Cost is measured in tokens
- Prices are per 1000 tokens
- Rough estimate is 1000 tokens is about 750 words
- Each model has a cost (see https://openai.com/pricing)
  - GPT-4 Turbo: $0.01 / 1K input, $0.03 / 1K output
  - GPT-4 → multiply by 3 (seriously, turbo is cheaper)
  - GPT-3.5 Turbo: $0.001 / 1K input, $0.002 / 1K output (default)
  - Embedding (more about that later) is about $0.0001 / 1K tokens
  - DALL-E 3, standard resolution, 1024x1024: $0.040 / image
  - Whisper (transcribe audio): $0.006 / minute
  - Text-to-speech: $0.015 / 1K characters, or double that for HD

Output Parsing

- Output is a String
- Spring AI tries to format the string as requested

Output Parsers

- Similar to RowMapper in Spring JDBC
- `OutputParser` interface, extends Parser<T> and FormatProvider
- Available implementations:
  - `BeanOutputParser`
  - `MapOutputParser`
  - `ListOutputParser`

```java
public interface OutputParser<T> extends Parser<T>,
                                         FormatProvider {
}

@FunctionalInterface
public interface Parser<T> {
    T parse(String text);
}

public interface FormatProvider {
   String getFormat();
}
```

Adding your own data

- Read in data → Spring has document readers
  - `JsonReader`
  - `TextReader`
  - `PagePdfDocumentReader`
  - `ParagraphPdfDocumentReader`
  - `TokaDocumentReader`
- Divide documents to fit into AI model's context window
  - `TextSplitter`, several others
- Write out results for storage

Embedding Client

- Convert text to a series of vectors, known as embeddings
- Embedding is a numerical vector of similarity scores

Vector Databases

- Specialized database to handle similarity searches
- Do a similarity search to find similar documents
- Add the similar documents as the context for a query
- This is called Retrieval Augmented Generation (RAG)

Available Vector Stores

- Azure Vector Search
- Chroma
- Milvus
- Neo4j
- PgVector (PostgresQL)
- PineCone
- Redis
- SimpleVectorStore
- Weaviate

Ollama

- Ollama (https://ollama.ai/) lets you run LLMs on your local machine
  - Many models available
  - Default is llama2 (from Meta/Facebook)
- Installations available for macOS and Linux
  - Windows coming soon, but you can use WSL
- Download the model you want
- Ollama exposes it at localhost:11434

Ollama

- Download models (can be quite large)
  - Uncensored (!) models available
- Tends to be slow, but at least it's free
  - Also avoids sending data offsite
- Use additional Spring Boot Starter
  - spring-ai-ollama-spring-boot-starter
- Base URL and model are set as properties
  - spring.ai.ollama.base-url (defaults to localhost:11434)
  - spring.ai.ollama.chat.model (defaults to llama2)
- Same classes as OpenAI

Spring with AI without Spring AI

Spring with AI without Spring AI

- Always good to know one layer below your abstraction
  - Also useful for functionality not yet in Spring AI
- Spring includes REST clients
  - `RestTemplate` legacy
  - `WebClient` async in webflux module
  - `RestClient` sync version of WebClient, available in Spring 6.1
- Spring provides a JSON parser (Jackson)
- Spring Boot has a validation starter
  - Can use jakarta.validation.constraints annotations
- Spring Boot has a GraalVM native image compiler

Spring with AI without Spring AI

- What we can do:
  - List the models
  - Generate audio from text
  - Image generation using DALL-E 3
  - Work with LLMs Spring AI does not yet support (Claude, Gemini, …)

Process

- Map request and response JSON to Java records
- Define HTTP Exchange interface
  - Add abstract methods
  - `@GetExchange`, `@PostExchange`, etc
- Customize rest clients to submit headers
- Use boilerplate to tell Spring to generate implementation classes
- Inject interfaces into services to do pre- and post-processing
- Profit!

List the OpenAI Models

- All the models are available in a GET request to https://api.openai.com/v1/models

Map JSON to Records

JSON Model and Model List:

```
{
  "id": "davinci",
  "object": "model",
  "created": 1686935002,
  "owned_by": "openai"
}
```

```
{
  "object": "list",
  "data": [
    {
      "id": "model-id-0",
      "object": "model",
      "created": 1686935002,
      "owned_by": "organization-owner"
    },
    {
      "id": "model-id-1",
      "object": "model",
      "created": 1686935002,
      "owned_by": "organization-owner",
    }
  ]
}
```

Java Records

- Can nest records to match JSON structure
- Camel-case to snake-case conversion:
  - Either `@JsonProperty` specifying names, or
  - `@JsonNaming` annotation on class
  - Spring has a property for this, but it doesn't appear to work on records

Model and ModelList

```
1 public record ModelList(List<Model> data) {
2     @JsonNaming(PropertyNamingStrategies.SnakeCaseStrategy.class)
3     public record Model(String id, long created, String ownedBy) {
4 }
```

HTTP Exchange Interface

```java
@HttpExchange("/v1")
public interface OpenAIInterface {

    // Models
    @GetExchange(value = "/models", accept = "application/json")
    ModelList listModels();
}
```

Configure Beans

- Need a `RestClient` (or `WebClient` or `RestTemplate`)
  - Set Authorization header
  - Set base URL
- Need bean to tell Spring to implement the exchange interface
  - Boilerplate -- copy and paste
  - Just change interface class
- NOTE: Only has to be done once → add extra methods to interface later

```java
@Configuration
public class AppConfig {
    @Bean
    public RestClient openAIRestClient(@Value("${openai.baseurl}") String baseUrl,
                                       @Value("${OPENAI_API_KEY}") String apiKey) {
        return RestClient.builder()
                .defaultHeader("Authorization", "Bearer %s".formatted(apiKey))
                .baseUrl(baseUrl)
                .build();
    }

    @Bean
    public OpenAIInterface openAIInterface(RestClient client) {
        RestClientAdapter adapter = RestClientAdapter.create(client);
        HttpServiceProxyFactory factory = HttpServiceProxyFactory.builderFor(adapter).build();
        return factory.createClient(OpenAIInterface.class);
    }
}
```

Service classes

- This is actually enough, but:
  - Might want to pre- or post-process requests and responses
- Model list is a simple GET request with no inputs
- But output can be extracted from result and organized
- Therefore, autowire interface into a service

```java
@Service
public class OpenAIService {
    public static final String GPT35 = "gpt-3.5-turbo";
    public static final String GPT4 = "gpt-4-1106-preview";

    private final OpenAIInterface openAIInterface;

    @Autowired
    public OpenAIService(OpenAIInterface openAIInterface) {
        this.openAIInterface = openAIInterface;
    }

    public List<String> getModelNames() {
        return openAIInterface.listModels().data().stream()
                .map(ModelList.Model::id)
                .sorted()
                .toList();
    }
}
```

```java
@SpringBootTest
class OpenAIServiceTest {
    @Autowired
    private OpenAIService openAIService;

    @Test
    void getGPTModels() {
        List<String> modelNames = openAIService.getModelNames();
        assertThat(modelNames).anyMatch(name -> name.contains(OpenAIService.GPT35));
        assertThat(modelNames).anyMatch(name -> name.contains(OpenAIService.GPT4));
        modelNames.stream()
                .filter(name -> name.contains("gpt"))
                .forEach(System.out::println);
    }

    @Test
    void getAllModels() {
        openAIService.getModelNames()
                .forEach(System.out::println);
    }
}
```

Generate audio from text

- Now that we have the interface, the service, and the necessary beans:
  - Add methods to exchange interface
  - Add methods to service to pre-process requests and post-process responses
- Text to speech
  - Use speech endpoint
  - Send POST request with text
  - Choose built-in voice (six available)
  - Save returned byte[] to file
  - (Optional) Add an mp3 player
- https://platform.openai.com/docs/guides/text-to-speech

Sample curl request

Generate spoken audio from input text

curl ▾ Copy

```
curl https://api.openai.com/v1/audio/speech \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "tts-1",
    "input": "Today is a wonderful day to build something people love!",
    "voice": "alloy"
  }' \
  --output speech.mp3
```

Create speech

- Five total parameters; three required
  - model → `tts-1` or `tts-1-hd`
  - input → text for audio
  - voice → one of: *alloy*, *echo*, *fable*, *onyx*, *nova*, *shimmer*
  - response_format → one of: mp3, opus, aac, flac (mp3 is default)
  - speed → value between 0.25 and 4.0 (1.0 is default)

Map request JSON to record

```java
public record TTSRequest(
        String model,
        String input,
        Voice voice,
        ResponseFormat responseFormat,
        double speed
) {
    public TTSRequest(String model, String input, Voice voice) {
        this(model, input, voice, ResponseFormat.MP3, 1.0);
    }
}
```

Useful enums

```
public enum Voice {
    @JsonProperty("alloy") ALLOY,
    @JsonProperty("echo") ECHO,
    @JsonProperty("fable") FABLE,
    @JsonProperty("onyx") ONYX,
    @JsonProperty("nova") NOVA,
    @JsonProperty("shimmer") SHIMMER;
}
```

```
public enum ResponseFormat {
    @JsonProperty("mp3") MP3,
    @JsonProperty("opus") OPUS,
    @JsonProperty("aac") AAC,
    @JsonProperty("flac") FLAC
}
```

Add method to exchange interface

```java
@HttpExchange("/v1")
public interface OpenAIInterface {

    // Models
    @GetExchange(value = "/models", accept = "application/json")
    ModelList listModels();

   // Text-to-Speech
    @PostExchange(value = "/audio/speech",
            accept = "audio/mpeg", contentType = "application/json")
    byte[] getTextToSpeechResponse(@RequestBody TTSRequest ttsRequest);
}
```

Add methods to service

```java
public byte[] getAudioResponse(TTSRequest ttsRequest) {
    byte[] bytes = openAIInterface.getTextToSpeechResponse(ttsRequest);
    String fileName = FileUtils.writeSoundBytesToFile(bytes);
    logger.info("Saved {} to {}", fileName, FileUtils.AUDIO_DIRECTORY);
    return bytes;
}

public byte[] getAudioResponse(String prompt) {
    return getAudioResponse(new TTSRequest(DEFAULT_MODEL, prompt, DEFAULT_VOICE));
}

// .. add convenient overloads ...
```

Try it out

```java
@SpringBootTest
class OpenAIServiceTest {
    @Autowired
    private OpenAIService service;

    private final String audioPrompt = """
            The YouTube channel, "Tales from the jar side" is your best
            source for learning about Java, Spring, and other open source
            technologies, especially when combined with AI tools.
            The companion newsletter of the same name is also a lot of fun.
            """.replaceAll("\\s+", " ").trim();

    @Test
    void getAudioResponse() {
        byte[] audioResponse = service.getAudioResponse(audioPrompt);
        assertThat(audioResponse.length).isPositive();
    }
}
```

(Optional) Add MP3 Player

- JLayer: https://github.com/umjammer/jlayer
- Available through jitpack.io
- Instantiate `Player` class
  - Requires `InputStream`

JLayer dependency

```
repositories {
    mavenCentral()
    maven { setUrl("https://jitpack.io") }
}

dependencies {
   // JLayer for MP3 support
    implementation("com.github.umjammer:jlayer:1.0.2")
}
```

Add method to service

```
1  public void createAndPlay(String text, Voice voice) {
2      TTSRequest ttsRequest = new TTSRequest(TTS_1_HD, text, voice);
3      byte[] bytes = getAudioResponse(ttsRequest);
4      try {
5          Player player = new Player(buffer);
6          player.play();
7      } catch (JavaLayerException e) {
8          throw new RuntimeException(e);
9      }
10 }
```

(Optional) Validate Request

- Add spring-boot-starter-validation dependency
- Can now use JPA validation annotations

```java
import jakarta.validation.constraints.*;

public record TTSRequest(
        @Pattern(regexp = "tts-1(-hd)?(-1106)?") String model,
        @NotBlank @Size(max = 4096) String input,
        Voice voice,
        ResponseFormat responseFormat,
        @DecimalMin("0.25") @DecimalMax("4.0") double speed
) {
    public TTSRequest(String model, String input, Voice voice) {
        this(model, input, voice, ResponseFormat.MP3, 1.0);
    }
}
```

Validation

- Inject Validator instance into service
  - Careful! There are several Validator classes in classpath
  - Be sure to pick the one from jakarta.validation
- Validate request

```
1 private void validateRequest(TTSRequest ttsRequest) {
2     Set<ConstraintViolation<TTSRequest>> violations = validator.validate(ttsRequest);
3     if (!violations.isEmpty()) {
4         throw new IllegalArgumentException(violations.toString());
5     }
6 }
```

(Optional) Add a controller

```java
@RestController
public class TextToSpeechController {
    // ... autowire OpenAIService ...

    @PostMapping("/tts")
    public ResponseEntity<String> convertTextToSpeech(@RequestBody String text) {
        service.getAudioResponse(TextToSpeechService.TTS_1_HD, text, Voice.randomVoice());
        return ResponseEntity.ok("(%s...) converted to mp3".formatted(text.substring(0, 24)));
    }

    @PostMapping("/tts/play")
    public ResponseEntity<String> convertTextAndPlay(@RequestBody String text) {
        service.createAndPlay(text, Voice.randomVoice());
        return ResponseEntity.ok("(%s...) converted to mp3".formatted(text.substring(0, 24)));
    }
}
```

(Optional) GraalVM native image compiler

- Spring supports GraalVM native image compiler
- For stateless services, this can be convenient
- Use with Function As A Service cloud providers
- Rapid startup and shutdown of stateless services

Process

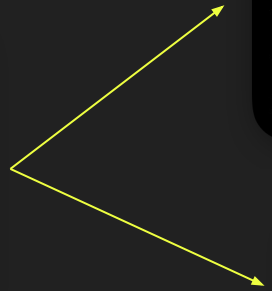
- This process can be used for all the services
  - Map inputs and outputs to records
  - Add methods to HTTP exchange interface
  - Autowire interface into service for preprocessing requests and postprocessing responses
- Optionally:
  - Add enums or other constants
  - Add helper functions to extract desired data from responses
  - Validate requests
  - Add controllers

DALL-E Image Generation

- Image generation is included in GitHub repository
  - Request has only a few parameters
  - Response can either be a URL for the image (default), or
  - Response can be the 64-bit binary encoded image itself
- Valid image sizes are all square:
  - 1024x1024 (default for DALL-E 3)
  - Other sizes cost more
  - High definition costs more
- Can request only 1 image at a time for DALL-E 3
- Endpoint:
  - POST to https://api.openai.com/v1/images/generations

Image Request

```
{
    "prompt": "A cute baby sea otter",
    "n": 2,
    "size": "1024x1024",
    "response_format": "b64_json"
}
```



```
public record ImageRequest(String prompt,
                           Integer n,
                           String size,
                           String responseFormat) {
}
```

```
public record ImageResponse(Long created,
                            List<Image> data) {
}
```

- response_format:
  - url (default)
  - b64_json → returns the binary encoded string directly

Saving an image to a file

- May return multiple images, so need unique file names
- My approach is to use a combination of a timestamp and a counter
- From FileUtils:
  - int counter is an attribute of this class

```
1 public static boolean writeImageToFile(String imageData) {
2     String timestamp = LocalDateTime.now()
3         .format(DateTimeFormatter.ofPattern("yyyyMMddHHmmss"));
4     String fileName = String.format("image_%s_%d.png", timestamp, counter++);
```

Saving an image to a file

- Java library has java.util.Base64 class
  - `Base64.getDecoder().decode(imageData)` → `byte[]`

```
try {
    Files.createDirectories(directory);
    byte[] bytes = Base64.getDecoder().decode(imageData);
    Files.write(filePath, bytes, StandardOpenOption.CREATE_NEW);
    System.out.printf("Saved %s to src/main/resources/images%n", fileName);
    return true;
} catch (IOException e) {
    throw new UncheckedIOException("Error writing image to file", e);
}
```

Test

```java
@Test
public void testDallE() {
    DallE dallE = new DallE();
    long num = dallE.getImages("""
            A realistic photo of a
            robot leaping into the air
            in joy after accomplishing a
            difficult task successfully
            """, 2);
    System.out.printf("Downloaded %d images%n", num);
    assertThat(num).isEqualTo(2);
}
```

Spring and AI without Spring AI

- See my AI playlist on the *Tales from the jar side* YouTube channel :)

Conclusions

- Spring AI works, but very early in development process
  - Expect frequent changes
- Best features:
  - Simple chat requests and responses
  - Prompt templates for input
  - Output parsers for output
  - RAG with embedding clients and vector databases
  - Works with multiple LLMs (though OpenAI most mature)

Conclusions

- Otherwise, Spring provides helpful features
  - HTTP Exchange interfaces
  - JavaConfig files to configure rest clients and implement exchange interfaces
  - Autowiring into services
  - Jackson JSON parsing
  - Validation using the Bean Validation specification
  - Rest controllers
  - GraalVM native image compilation
- Expect Spring AI project to improve