LangChain4j 1.7.1 Training Course

Building AI-Powered Java Applications

Press Space for next page →

Contact Info

Ken Kousen Kousen IT, Inc.

- ken.kousen@kousenit.com
- http://www.kousenit.com
- http://kousenit.org (blog)
- Social Media:
  - @kenkousen (twitter)
  - @kenkousen@foojay.social (mastodon)
  - @kousenit.com (bluesky)
- *Tales from the jar side* (free newsletter)
  - https://kenkousen.substack.com
  - https://youtube.com/@talesfromthejarside

Course Overview

5-Hour Hands-On Workshop

- LangChain4j fundamentals and architecture
- 10 progressive hands-on labs
- Real-world AI application patterns
- Production deployment strategies

Topics We'll Cover

- Chat models and streaming responses
- Structured data extraction with AI
- AI Services and type-safe interfaces
- Chat memory and conversation management
- Tool calling and function integration
- Model Context Protocol (MCP)
- Multimodal capabilities (images, audio)
- Retrieval-Augmented Generation (RAG)
- Vector stores and embeddings
- Production best practices

Prerequisites

- Java 17+ experience
- Basic understanding of AI/LLMs
- Gradle or Maven familiarity
- OpenAI API key (or alternative provider)
- IDE (IntelliJ IDEA recommended)

What is LangChain4j?

Java Library for Building LLM Applications

Key Features

Core capabilities

🤖 Multi-Model Support

OpenAI, Anthropic, Google AI, and more

💬 Chat & Memory

Stateful conversations with context

🛠️ Tool Calling

AI calls your Java methods

📚 RAG Support

Augment AI with your data

🖼️ Multimodal

Images, audio, and more

Why LangChain4j?

Java-native AI development

☕ Native Java

No Python required

🔒 Type Safety

Compile-time checks

🍃 Spring Boot

First-class integration

🚀 Production Ready

Battle-tested in production

📖 Documentation

What’s New in 1.7.1?

Core enhancements

🎯 Class-Based Agents

Create agents from classes

🗄️ ChromaDB API V2

Enhanced vector store

🐋 Docker MCP Transport

Containerized integration

What’s New in 1.7.1?

Additional improvements

🤖 Enhanced Models

OpenAI SDK v4.0, custom params

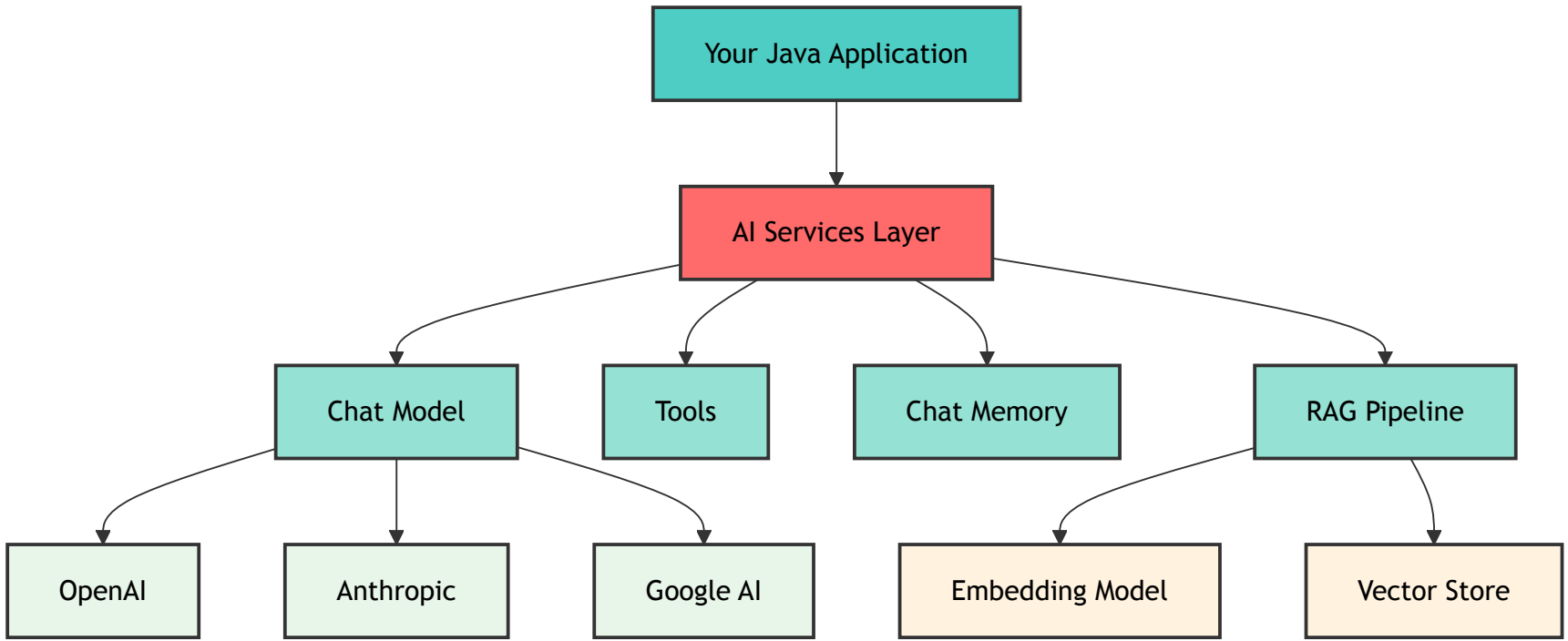
📄 New Parsers

YAML, Oracle, GPU support

⚠️ HuggingFace Deprecated

Use OpenAI/Anthropic/Google AI

LangChain4j Architecture



Installation and Setup

Add LangChain4j to your project

```
// build.gradle.kts
dependencies {
    // BOM for version management
    implementation(platform("dev.langchain4j:langchain4j-bom:1.7.1"))

    // Core library
    implementation("dev.langchain4j:langchain4j")

    // Model integrations
    implementation("dev.langchain4j:langchain4j-open-ai")
    implementation("dev.langchain4j:langchain4j-anthropic")
    implementation("dev.langchain4j:langchain4j-google-ai-gemini")

    // Embeddings and RAG
    implementation("dev.langchain4j:langchain4j-embeddings-all-minilm-l6-v2")
    implementation("dev.langchain4j:langchain4j-chroma")
}
```

Environment Setup

Configure API keys

```
# OpenAI
export OPENAI_API_KEY="your-openai-api-key"

# Anthropic (optional)
export ANTHROPIC_API_KEY="your-anthropic-api-key"

# Google AI (optional)
export GOOGLEAI_API_KEY="your-google-ai-key"
```

⚠️ **Security:** Never commit API keys to version control

Verify Installation

```
# Build the project
./gradlew build

# Run tests
./gradlew test
```

Lab 1: Basic Chat

Getting Started

Your first AI conversation

Lab 1.1: Simple Query

Basic chat interaction

```
@Test
void simpleQuery() {
    ChatModel model = OpenAiChatModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .modelName(GPT_5_NANO)
            .build();

    String response = model.chat("Why is the sky blue?");

    System.out.println(response);
    assertNotNull(response);
    assertFalse(response.isEmpty());
}
```

Lab 1.2: System Message

Control AI behavior

```
SystemMessage systemMessage = SystemMessage.from(
    "You are a helpful assistant that responds like a pirate.");

UserMessage userMessage = UserMessage.from(
    "Why is the sky blue?");

ChatResponse response = model.chat(systemMessage, userMessage);

System.out.println(response.aiMessage().text());
```

🏴‍☠️ System messages shape AI personality

Lab 1.3: Response Metadata

Access token usage and costs

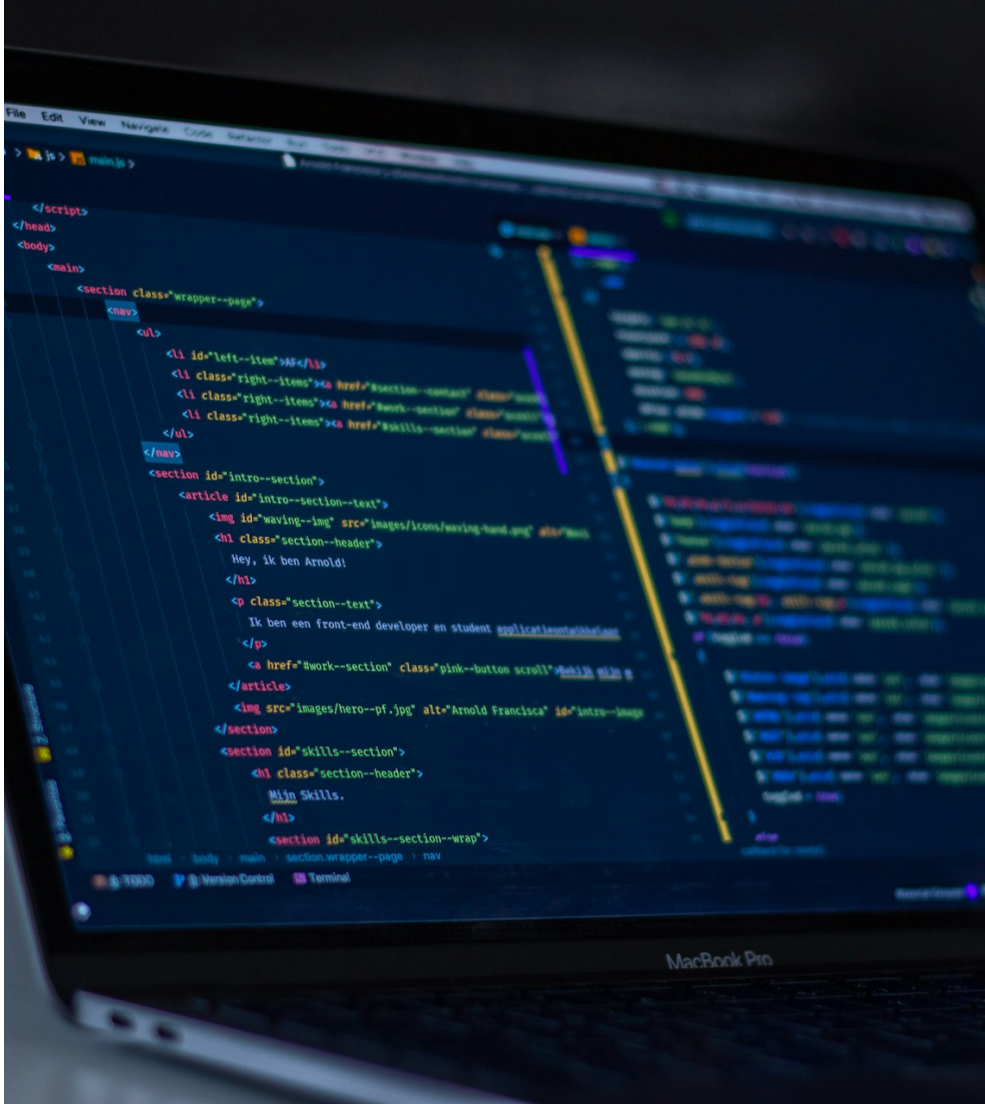
```
ChatResponse response = model.chat(userMessage);

System.out.println("Token Usage: " + response.tokenUsage());
System.out.println("Finish Reason: " + response.finishReason());
```

💡 **Track tokens:** Monitor costs and optimize prompts

Lab 2: Streaming

Real-Time Responses

Better user experience



Streaming Responses

Real-time AI responses for better user experience

```
@Test
void streamingChat() {
    StreamingChatModel model = OpenAiStreamingChatModel
        .builder()
        .apiKey(System.getenv("OPENAI_API_KEY"))
        .build();

    model.chat("Tell me a story",
        new StreamingChatResponseHandler() {
            public void onPartialResponse(String token) {
                System.out.print(token);
            }
            public void onCompleteResponse(ChatResponse r) {
                System.out.println("\nDone!");
            }
        });
}
```

🌊 Real-time tokens

⚡ Better UX

Streaming with Context

Add system messages

```
SystemMessage systemMessage = SystemMessage.from(
    "You are a helpful coding assistant.");

model.chat(Arrays.asList(systemMessage, userMessage),
    new StreamingChatResponseHandler() {
        // Handler implementation
    });

latch.await(30, TimeUnit.SECONDS);
```

Lab 3: Structured Data

Type-Safe Extraction

Parse AI responses

Structured Data Extraction

Type-safe data parsing

```
record Person(String name, int age, String occupation) {}

interface PersonExtractor {
    @UserMessage("Extract: {{text}}")
    Person extractPerson(@V("text") String text);
}

PersonExtractor ex = AiServices.create(
    PersonExtractor.class, model);

Person p = ex.extractPerson(
    "John Doe is a 35-year-old engineer");
```

💡 Returns: Person[name=John Doe, age=35, ...]

JSON Extraction

Manual parsing approach

```
ChatModel model = OpenAiChatModel.builder()
        .responseFormat("json_object")
        .build();

String prompt = """
    Generate JSON: {"actor": "Name", "movies": [...]}
    """;

String response = model.chat(prompt);
ObjectMapper mapper = new ObjectMapper();
ActorFilms data = mapper.readValue(response, ActorFilms.class);
```

Multiple Entities

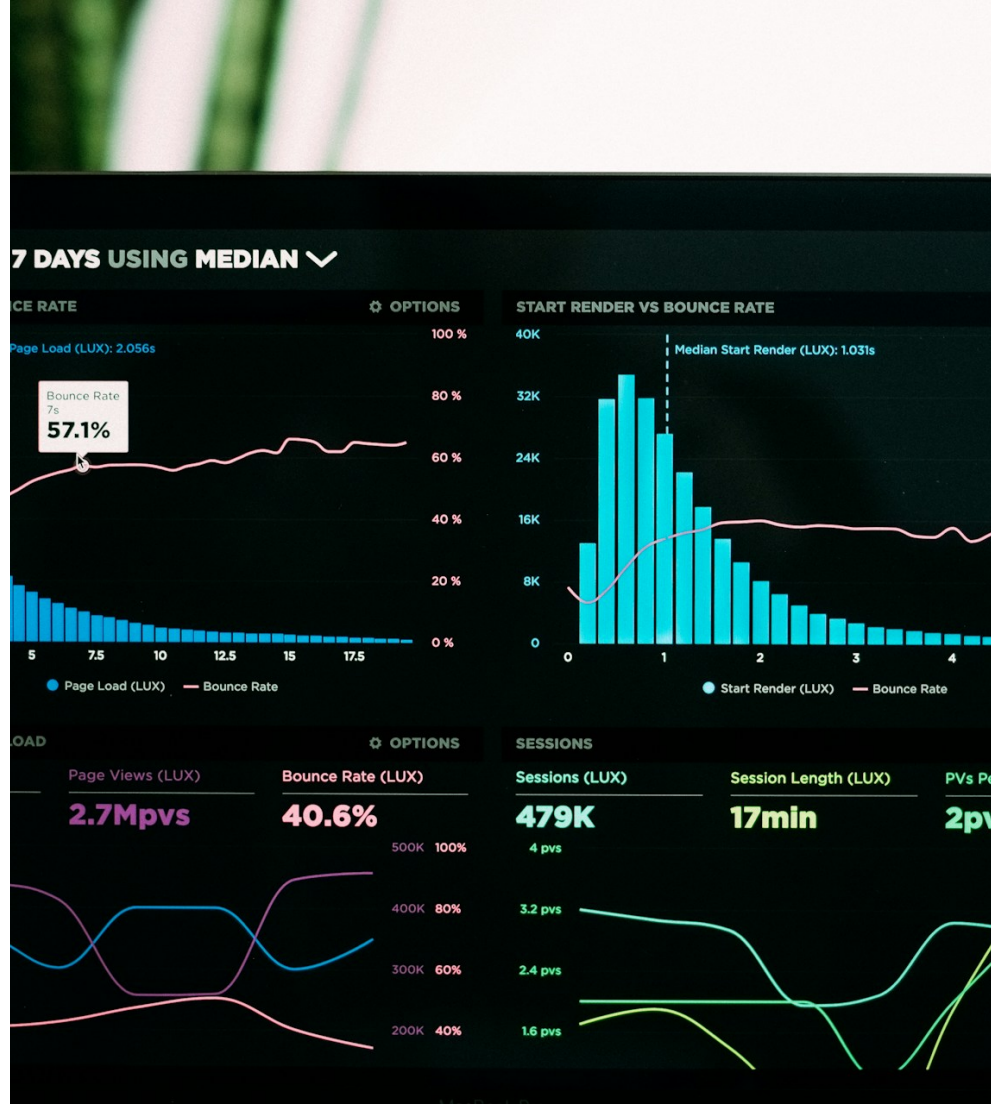
Extract collections

```
record ActorFilmographies(
    List<ActorFilms> filmographies) {}

interface ActorService {
    ActorFilmographies getMultipleActorFilmographies(
        @UserMessage String actors);
}

// Returns 3 actors with 4 movies each
ActorFilmographies result = service.getMultipleActorFilmographies(
    "Return 3 actors with 4 movies each");
```

Lab 4: AI Services

High-Level APIs

Type-safe AI integration



AI Services Interface

High-level type-safe AI interactions

```
interface FilmographyService {
    @SystemMessage("Movie expert")
    List<String> getMovies(
        @UserMessage String actor);
}

FilmographyService service =
    AiServices.builder(FilmographyService.class)
        .chatModel(model)
        .build();

List<String> movies = service.getMovies("Tom Hanks");
```

✨ Type-safe interfaces

✨ Annotation-driven

Service with Templates

Variable substitution

```
interface DocumentAnalyzer {
    @UserMessage("Analyze: {{content}}")
    String analyzeDocument(@V("content") String content);

    @UserMessage("Extract themes from: {{content}}")
    List<String> extractThemes(@V("content") String content);

    @UserMessage("Rate sentiment 1-10: {{content}}")
    int analyzeSentiment(@V("content") String content);
}
```

Service with Memory and Tools

Combine features

```
interface PersonalAssistant {
    String chat(String message);
}

PersonalAssistant assistant = AiServices.builder(PersonalAssistant.class)
    .chatModel(model)
    .chatMemory(MessageWindowChatMemory.withMaxMessages(10))
    .tools(new DateTimeTool())
    .build();

String response = assistant.chat("What's my name and the year in 3 years?");
```

Lab 5: Chat Memory

Stateful Conversations

Context management

Chat Memory

Single user conversation

```
interface Assistant {
    String chat(String message);
}

var assistant = AiServices.builder(Assistant.class)
    .chatModel(model)
    .chatMemory(MessageWindowChatMemory
        .withMaxMessages(10))
    .build();

assistant.chat("My name is Alice");
assistant.chat("What's my name?");
// Response: "Your name is Alice"
```

Chat Memory

Multi-user conversations

```
interface MultiUserAssistant {
    String chat(@MemoryId int userId,
                @UserMessage String msg);
}

var assistant = AiServices.builder(...)
    .chatMemoryProvider(memoryId ->
        MessageWindowChatMemory
            .withMaxMessages(10))
    .build();

assistant.chat(1, "I'm Alice");
assistant.chat(2, "I'm Bob");
```

Memory Types

Different memory strategies

```
// Message window - limit by count
ChatMemory messageMemory =
    MessageWindowChatMemory.withMaxMessages(10);

// Token window - limit by tokens
ChatMemory tokenMemory =
    TokenWindowChatMemory.withMaxTokens(
        1000,
        new OpenAiTokenizer(GPT_5_NANO)
    );
```

Stateless vs Stateful

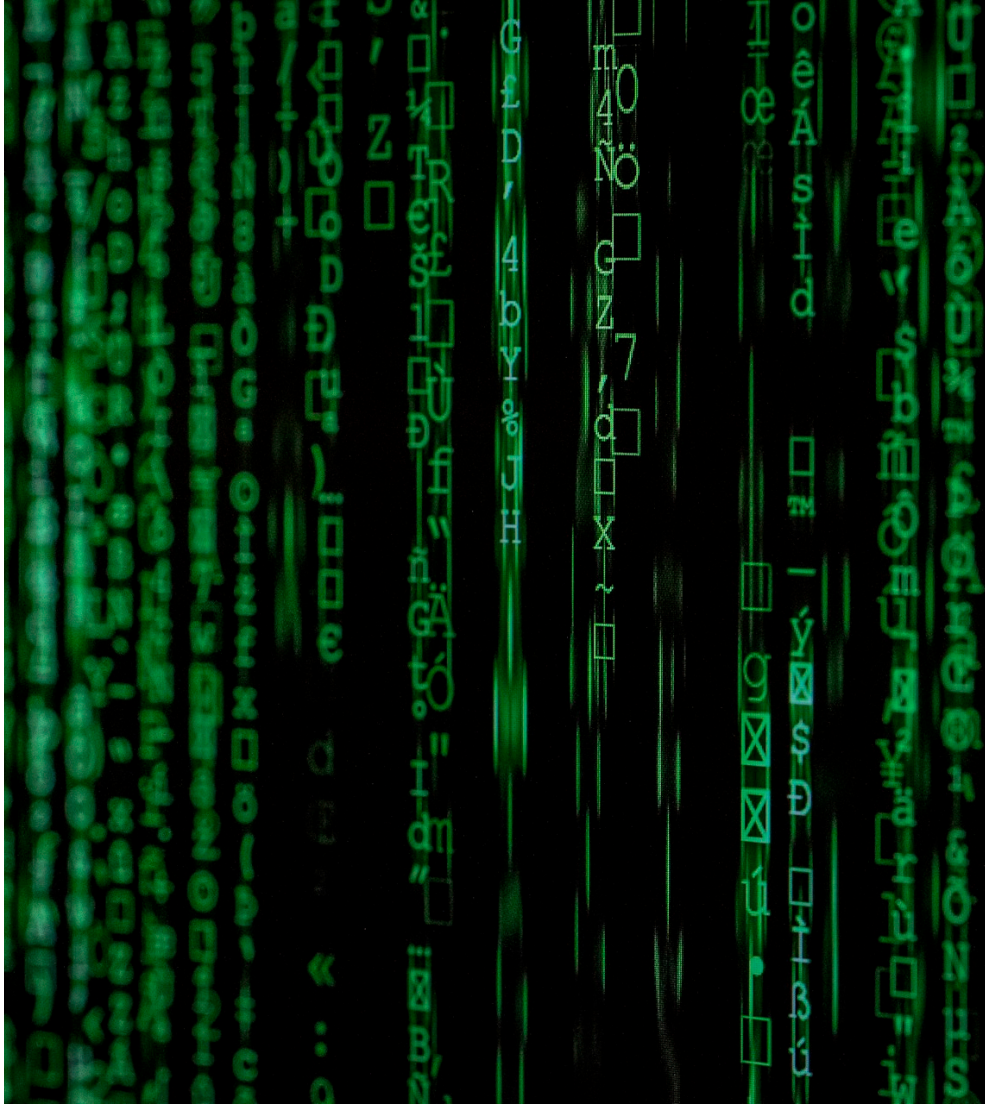
Understanding the difference

```
// Stateless - no memory
ChatModel model = OpenAiChatModel.builder().build();
model.chat("My name is Alice");
model.chat("What's my name?");
// AI doesn't remember

// Stateful - with memory
ChatMemory memory = MessageWindowChatMemory.withMaxMessages(10);
memory.add(UserMessage.from("My name is Alice"));
ChatResponse response = model.chat(memory.messages());
// AI remembers
```

Lab 6: AI Tools

Function Calling

AI calls your code



AI Tools (Function Calling)

Let AI call your Java methods

```
class WeatherTool {
    @Tool("Get current weather")
    String getWeather(String location) {
        return "Weather in " + location +
               ": 72°F, sunny";
    }
}

var assistant = AiServices.builder(Assistant.class)
    .chatModel(model)
    .tools(new WeatherTool())
    .build();

assistant.chat("What's the weather in NYC?");
```

Tool Examples

DateTimeTool implementation

```
class DateTimeTool {
    @Tool("Get the current date and time")
    public String getCurrentDateTime() {
        return LocalDateTime.now()
            .format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
    }

    @Tool("Get date N years from now")
    public String getDateYearsFromNow(int years) {
        return LocalDate.now().plusYears(years).toString();
    }

    @Tool("Set an alarm")
    public String setAlarm(String time) {
        return "Alarm set for " + time;
    }
}
```

Multiple Tools

Combine tool capabilities

```
Assistant assistant = AiServices.builder(Assistant.class)
    .chatModel(model)
    .tools(
        new DateTimeTool(),
        new WeatherTool(),
        new CalculatorTool()
    )
    .build();

String response = assistant.chat(
    "What's 15 * 8, and what year is it in 3 years?"
);
```

Tool Parameters

Complex parameters

```
class CalculatorTool {
    @Tool("Add two numbers")
    public double add(double a, double b) {
        return a + b;
    }

    @Tool("Divide first by second")
    public double divide(double a, double b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

Lab 6.5: MCP Integration

External Tools

Model Context Protocol

MCP Integration

External tool protocol

```
// MCP client (stdio)
McpTransport transport =
    new StdioMcpTransport.Builder()
        .command(List.of("npx", "-y",
            "@modelcontextprotocol/server-everything"))
        .build();

// Use with AI
McpToolProvider tools = McpToolProvider.builder()
    .mcpClients(new DefaultMcpClient.Builder()
        .transport(transport).build())
    .build();

Assistant ai = AiServices.builder(Assistant.class)
    .toolProvider(tools).build();
```

🐳 Docker transport in 1.7.1

MCP Architecture

MCP with Local Tools

Hybrid approach

```
// Combine local and external tools
Assistant assistant = AiServices.builder(Assistant.class)
    .chatModel(model)
    .tools(new DateTimeTool())  // Local tool
    .toolProvider(mcpToolProvider)  // External tools
    .build();

// AI can use both tool types
String response = assistant.chat(
    "What's the date in 5 years and fetch the weather data?"
);
```

Docker MCP Transport (NEW)

Containerized MCP servers

```
// NEW in 1.7.1 - Docker transport
McpTransport dockerTransport = new DockerMcpTransport.Builder()
    .image("mcp-server:latest")
    .build();

McpClient dockerClient = new DefaultMcpClient.Builder()
    .transport(dockerTransport)
    .build();

McpToolProvider toolProvider = McpToolProvider.builder()
    .mcpClients(dockerClient)
    .build();
```

Lab 7: Multimodal

Images & Audio

Beyond text

Multimodal: Images

Process images with AI

```
// Vision model
ChatModel model = OpenAiChatModel.builder()
    .modelName(GPT_5)
    .build();

// Analyze image
ImageContent img = ImageContent.from(imageString, "image/jpeg");
TextContent txt = TextContent.from("What do you see?");
UserMessage msg = UserMessage.from(txt, img);

String response = model.chat(msg).aiMessage().text();
```

 GPT-5 for vision

Image from URL

Remote image analysis

```
// Use public URL
String imageUrl = "https://example.com/image.jpg";

ImageContent imageContent = ImageContent.from(imageUrl);
TextContent textContent = TextContent.from(
    "Describe this landscape in detail"
);

UserMessage message = UserMessage.from(textContent, imageContent);
String analysis = model.chat(message).aiMessage().text();
```

Structured Image Analysis

Extract structured data from images

```
interface DetailedAnalyst {
    @UserMessage("Analyze image: {{image}}")
    ImageAnalysisResult analyzeComprehensively(
        @V("image") ImageContent image
    );
}

record ImageAnalysisResult(
    String description,
    List<String> objects,
    List<String> colors,
    String mood
) {}

DetailedAnalyst analyst = AiServices.builder(DetailedAnalyst.class)
    .chatModel(model)
    .build();
```

Multimodal: Audio

Speech-to-text with Gemini

```
// Gemini for audio
ChatModel model = GoogleAiGeminiChatModel.builder()
    .apiKey(System.getenv("GOOGLEAI_API_KEY"))
    .modelName("gemini-2.5-flash-preview-05-20")
    .build();

// Transcribe audio
AudioContent audio = AudioContent.from(
    readAudioData(), "audio/mp3");
UserMessage msg = UserMessage.from(
    TextContent.from("Transcribe:"), audio);

String text = model.chat(msg).aiMessage().text();
```

🎵 MP3, WAV support

Lab 8: Image Generation

Create Images

DALL-E integration

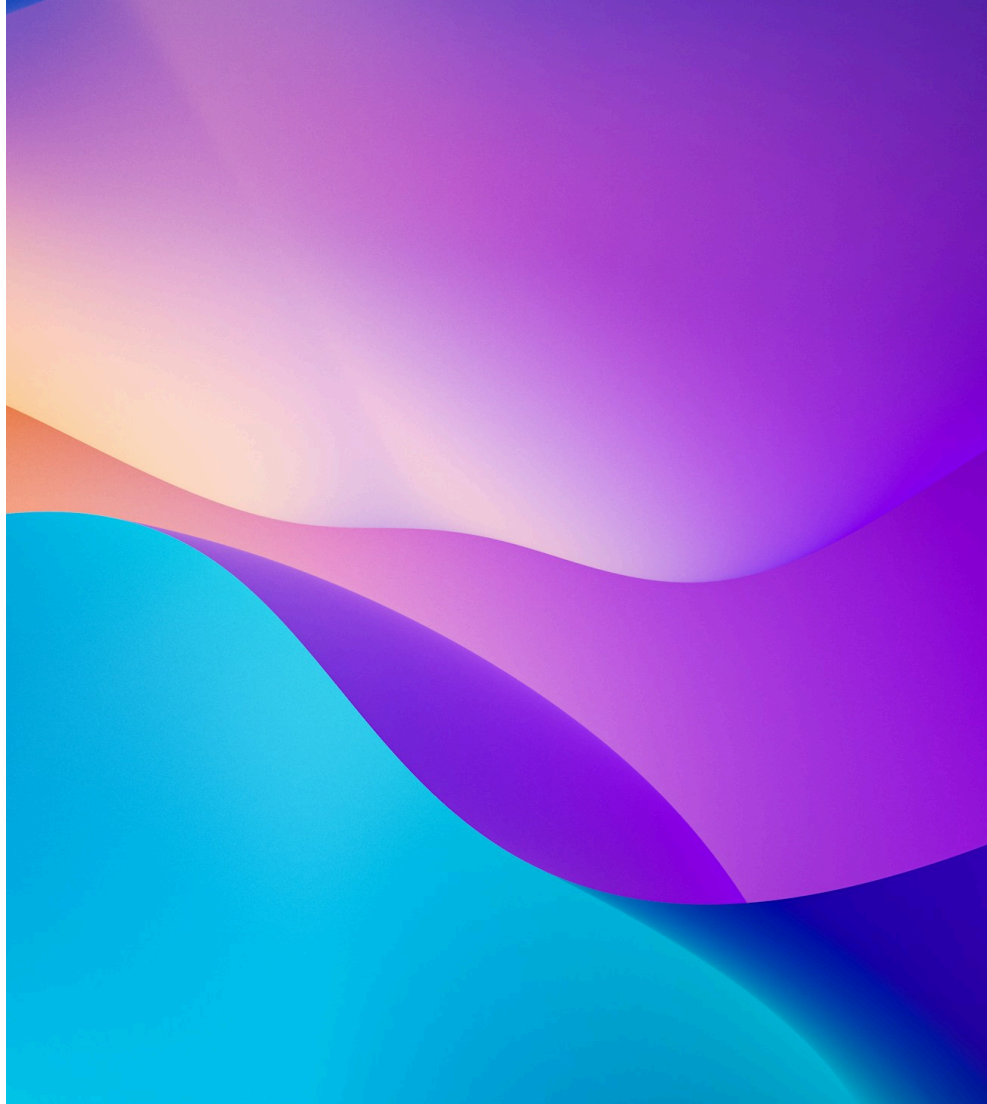


Image Generation

Create images with DALL-E

```
// DALL-E model
ImageModel imageModel = OpenAiImageModel.builder()
    .modelName(DALL_E_3)
    .quality("hd")
    .build();

// Generate from prompt
Response<Image> response = imageModel.generate(
    "A coffee cup on a desk, photorealistic");

Image img = response.content();
String url = img.url();
```

🎨 DALL-E 3 support

Image Generation Options

Configure quality and style

```
ImageModel model = OpenAiImageModel.builder()
    .apiKey(System.getenv("OPENAI_API_KEY"))
    .modelName(DALL_E_3)
    .size("1024x1024")
    .quality("hd")          // "standard" or "hd"
    .style("vivid")         // "vivid" or "natural"
    .build();

String prompt = "A futuristic cityscape at dawn with flying vehicles";
Response<Image> response = model.generate(prompt);
```

Base64 Image Generation

Using gpt-image-1 model

```
ImageModel model = OpenAiImageModel.builder()
    .modelName("gpt-image-1")
    .build();

Response<Image> response = model.generate("A warrior cat rides a dragon");
Image image = response.content();

// Get base64 data
byte[] imageBytes = Base64.getDecoder().decode(image.base64Data());

// Save to file
Path outputPath = Path.of("src/main/resources/generated_image.png");
Files.write(outputPath, imageBytes);
```

Creative Variations

Generate multiple images

```
String[] prompts = {
    "A steampunk robot playing chess",
    "A minimalist abstract representation of music",
    "A cozy library in a treehouse"
};

for (String prompt : prompts) {
    Response<Image> response = model.generate(prompt);
    Image image = response.content();

    System.out.println("Generated: " + image.url());
    System.out.println("Revised: " + image.revisedPrompt());
}
```

Lab 9: RAG Basics

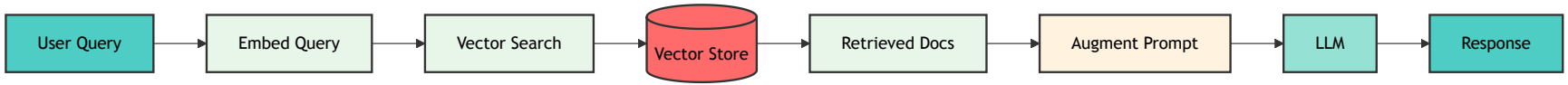
Knowledge Augmentation

AI + your data

RAG Implementation

```
// Setup embedding model and store
EmbeddingModel embeddingModel =
    new AllMiniLmL6V2QuantizedEmbeddingModel();
EmbeddingStore<TextSegment> store =
    new InMemoryEmbeddingStore<>();

// Load and split documents
Document doc = FileSystemDocumentLoader.loadDocument(
    Paths.get("knowledge-base.pdf"));
List<TextSegment> segments =
    DocumentSplitters.recursive(300, 50).split(doc);

// Create RAG assistant
Assistant assistant = AiServices.builder(Assistant.class)
    .chatModel(model)
    .contentRetriever(
        EmbeddingStoreContentRetriever.from(store))
    .build();
```

RAG Architecture



Document Loading

Load from various sources

```
// From file system
Document doc = FileSystemDocumentLoader.loadDocument(
    Paths.get("document.pdf")
);

// From URL
Document webDoc = UrlDocumentLoader.load(
    new URL("https://example.com/docs")
);

// Create from string
Document textDoc = Document.from("Sample content");
```

Document Splitting

Chunk documents for embeddings

```
// Recursive splitter
DocumentSplitter splitter = DocumentSplitters.recursive(
    300,  // Max chunk size
    50    // Overlap
);

// Character splitter
DocumentSplitter charSplitter = DocumentSplitters.character(
    500,
    100
);

// Sentence splitter
DocumentSplitter sentenceSplitter = DocumentSplitters.sentence(
    10  // Max sentences per chunk
);
```

Embedding and Storage

Store document embeddings

```
// Embed documents
List<Embedding> embeddings = embeddingModel.embedAll(segments).content();

// Store embeddings with segments
embeddingStore.addAll(embeddings, segments);

// Query with embedding
Embedding queryEmbedding = embeddingModel.embed("search query").content();

List<EmbeddingMatch<TextSegment>> matches = embeddingStore.search(
    EmbeddingSearchRequest.builder()
        .queryEmbedding(queryEmbedding)
        .maxResults(5)
        .minScore(0.7)
        .build()
).matches();
```

RAG with Metadata

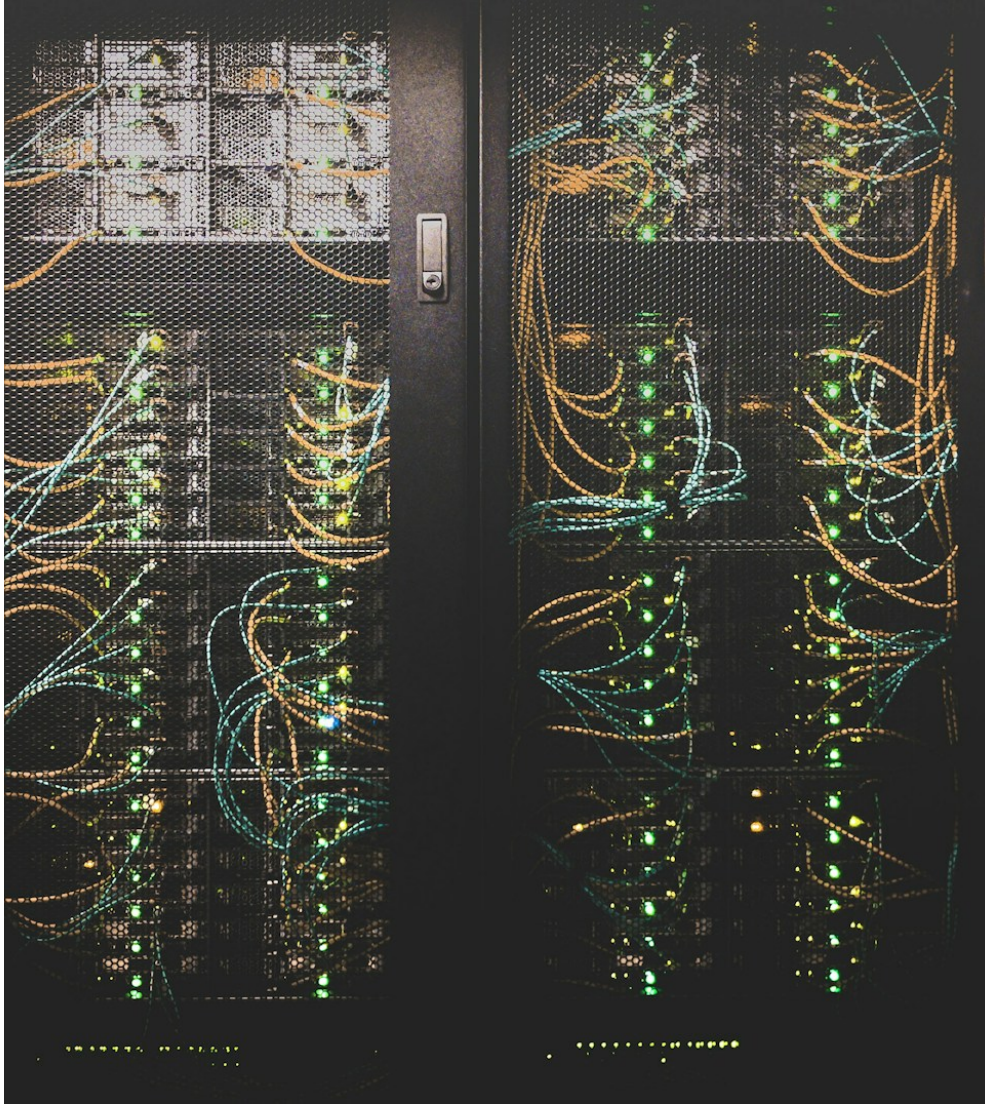
Add context to documents

```
Document doc = Document.from("Java was created by James Gosling")
    .toBuilder()
    .metadata("language", "java")
    .metadata("topic", "history")
    .metadata("created_at", LocalDateTime.now().toString())
    .build();

// Use metadata in retrieval
ContentRetriever retriever = EmbeddingStoreContentRetriever.builder()
    .embeddingStore(store)
    .embeddingModel(embeddingModel)
    .maxResults(3)
    .minScore(0.6)
    .build();
```

Lab 10: Vector Stores

Production RAG

ChromaDB integration



ChromaDB (API V2)

Production vector store

```
// ChromaDB with API V2 support (1.7.1)
EmbeddingStore<TextSegment> store =
    ChromaEmbeddingStore.builder()
        .baseUrl("http://localhost:8000")
        .collectionName(randomUUID())
        .build();

// Process and store
List<TextSegment> segments = splitDocuments(docs);
List<Embedding> embeddings =
    embeddingModel.embedAll(segments).content();
store.addAll(embeddings, segments);

// Semantic search
List<EmbeddingMatch<TextSegment>> matches =
    store.search(EmbeddingSearchRequest.builder()
        .queryEmbedding(queryEmbedding)
        .maxResults(5).build()).matches();
```

🚀 API V2: Better performance

ChromaDB Setup

Start ChromaDB server

```
# Docker (recommended)
docker run -p 8000:8000 chromadb/chroma:0.5.4

# Verify connection
curl http://localhost:8000/api/v1/heartbeat
```

💡 **API V2:** LangChain4j 1.7.1 supports ChromaDB API V2 for enhanced performance

Production RAG System

Complete implementation

```
// Configure with production settings
ChatModel chatModel = OpenAiChatModel.builder()
    .apiKey(System.getenv("OPENAI_API_KEY"))
    .modelName(GPT_5_NANO)
    .build();

EmbeddingStore<TextSegment> store = ChromaEmbeddingStore.builder()
    .baseUrl("http://localhost:8000")
    .collectionName(randomUUID())
    .logRequests(true)
    .build();

// Add metadata for production
segment.metadata().put("source", "docs");
segment.metadata().put("created_at", LocalDateTime.now().toString());

ContentRetriever retriever = EmbeddingStoreContentRetriever.builder()
    .embeddingStore(store)
    .maxResults(3)
    .minScore(0.6)
    .build();
```

RAG with Document Parsing

PDF processing with Apache Tika

```
// Load PDF document
Path documentPath = Paths.get("src/test/resources/document.pdf");
Document document = FileSystemDocumentLoader.loadDocument(documentPath);

// Split with metadata
DocumentSplitter splitter = DocumentSplitters.recursive(300, 50);
List<TextSegment> segments = splitter.split(document);

for (int i = 0; i < segments.size(); i++) {
    TextSegment segment = segments.get(i);
    segment.metadata().put("chunk_id", String.valueOf(i));
    segment.metadata().put("source_file", "document.pdf");
    segment.metadata().put("document_type", "pdf");
}
```

Vector Store Comparison

Different storage options

In-Memory

```
new InMemoryEmbeddingStore<>();
```

- ✅ Fast - ✅ Simple - ❌ No persistence - ❌ Limited scale

ChromaDB

```
ChromaEmbeddingStore.builder()
    .baseUrl("http://localhost:8000")
    .build();
```

- ✅ Persistent - ✅ Scalable - ✅ Production-ready - ✅ API V2 support

Best Practices

Production Patterns

Enterprise-ready code

Error Handling

Robust error management

```
try {
    String response = model.chat(prompt);
    return response;
} catch (Exception e) {
    log.error("AI call failed: {}", e.getMessage());
    // Fallback strategy
    return getDefaultResponse();
}

// With retries
Retry retry = Retry.ofDefaults("aiService");
String response = retry.executeSupplier(() ->
    model.chat(prompt)
);
```

Token Management

Monitor and control costs

```
ChatModel model = OpenAiChatModel.builder()
    .apiKey(System.getenv("OPENAI_API_KEY"))
    .maxTokens(500)  // Limit response size
    .build();

// Track usage
ChatResponse response = model.chat(userMessage);
TokenUsage usage = response.tokenUsage();

log.info("Input tokens: {}", usage.inputTokenCount());
log.info("Output tokens: {}", usage.outputTokenCount());
log.info("Total tokens: {}", usage.totalTokenCount());
```

Prompt Engineering

Effective prompts

```
@SystemMessage("""
    You are an expert Java developer assistant.
    Provide accurate, concise answers.
    Use code examples when appropriate.
    If uncertain, clearly state limitations.
    """)
interface JavaAssistant {
    @UserMessage("""
        Task: {{task}}
        Context: {{context}}
        Requirements: {{requirements}}

        Provide a detailed solution.
        """)
    String solveTask(
        @V("task") String task,
        @V("context") String context,
        @V("requirements") String requirements
    );
}
```

Testing AI Services

Unit and integration tests

```
@Test
void testAiServiceResponse() {
    // Use test model with fixed responses
    ChatModel mockModel = mock(ChatModel.class);
    when(mockModel.chat(any(String.class)))
        .thenReturn("Expected response");

    MyService service = AiServices.builder(MyService.class)
        .chatModel(mockModel)
        .build();

    String result = service.process("test input");

    assertThat(result).contains("Expected response");
    verify(mockModel, times(1)).chat(any(String.class));
}
```

Production Configuration

Environment-based settings

```
@Configuration
public class AiConfig {

    @Bean
    @Profile("production")
    public ChatModel productionModel() {
        return OpenAiChatModel.builder()
            .apiKey(System.getenv("OPENAI_API_KEY"))
            .modelName(GPT_5_NANO)
            .maxRetries(3)
            .timeout(Duration.ofSeconds(30))
            .build();
    }

    @Bean
    @Profile("development")
    public ChatModel devModel() {
        return OpenAiChatModel.builder()
            .modelName(GPT_5)
            .build();
    }
}
```

Caching Strategies

Reduce costs with caching

```
@Cacheable("ai-responses")
public String getCachedResponse(String prompt) {
    return model.chat(prompt);
}

// Redis cache configuration
@Bean
public CacheManager cacheManager(RedisConnectionFactory factory) {
    return RedisCacheManager.builder(factory)
        .cacheDefaults(
            RedisCacheConfiguration.defaultCacheConfig()
                .entryTtl(Duration.ofHours(24))
        )
        .build();
}
```

Security Best Practices

Protect sensitive data

- **Never log API keys** - Use environment variables
- **Sanitize inputs** - Validate and escape user input
- **Rate limiting** - Prevent abuse
- **API key rotation** - Regular updates
- **Audit logging** - Track AI usage
- **Content filtering** - Block inappropriate content

Monitoring and Observability

Track AI performance with AOP

```
@Aspect
@Component
public class AiMonitoringAspect {
    @Around("@annotation(Tool)")
    public Object monitorToolExecution(ProceedingJoinPoint pjp) {
        long start = System.currentTimeMillis();
        try {
            Object result = pjp.proceed();
            metrics.record(pjp.getSignature().getName(),
                System.currentTimeMillis() - start, "success");
            return result;
        } catch (Throwable e) {
            metrics.record(pjp.getSignature().getName(),
                System.currentTimeMillis() - start, "error");
            throw e;
        }
    }
}
```

📊 Track execution time and success rates

Production Considerations

🔒 Security

- • API key management
- • Input validation
- • Rate limiting

📊 Monitoring

- • Token usage tracking
- • Error rates
- • Response times

⚡ Performance

- • Caching strategies
- • Connection pooling
- • Async processing

🧪 Testing

- • Mock AI services
- • Integration tests
- • Load testing

Spring Boot Integration

Native framework support

```
@RestController
@RequestMapping("/api/chat")
public class ChatController {

    private final Assistant assistant;

    @Autowired
    public ChatController(ChatModel model, ChatMemory memory) {
        this.assistant = AiServices.builder(Assistant.class)
            .chatModel(model)
            .chatMemory(memory)
            .build();
    }

    @PostMapping
    public String chat(@RequestBody ChatRequest request) {
        return assistant.chat(request.getMessage());
    }
}
```

Spring Boot Configuration

Auto-configuration

```yaml
# application.yml
langchain4j:
  open-ai:
    chat-model:
      api-key: ${OPENAI_API_KEY}
      model-name: gpt-4o-mini
      temperature: 0.7
      max-tokens: 1000

  embedding-model:
    all-mini-lm-l6-v2:
      enabled: true
```

Lab Progression

10 hands-on labs

🚀 Foundation

Labs 1-3: Chat, Streaming, Extraction

🧠 Services & Memory

Labs 4-5: AI Services, Chat Memory

🛠️ Tools & Integration

Labs 6-8: Function Calling, MCP, Multimodal

📚 RAG Implementation

Labs 9-10: RAG, Vector Stores

Resources

Documentation and course materials

📚 Documentation

- • [LangChain4j Docs](https://docs.langchain4j.dev)
- • [GitHub Repository](https://github.com/langchain4j/langchain4j)
- • [Examples](https://github.com/langchain4j/langchain4j-examples)

🛠️ This Course (v1.7.1)

- • Main branch: Starter code
- • Solutions branch: Complete implementations
- • Labs.md: Step-by-step guide
- • UPGRADE_NOTES_1.7.1.md: What's new

Best Practices

Tips for production use

🎯 Best Practices

- • Use environment variables for API keys
- • Implement proper error handling
- • Monitor token usage
- • Cache embeddings when possible
- • Test with different models
- • Use ChromaDB API V2 for production

💡 Tips

- • Start simple, iterate
- • Read the JavaDocs
- • Check the examples repo
- • Join the community
- • Explore class-based agents (1.7.1)
- • Try Docker MCP transport

Community & Support

📚 Official Resources

- LangChain4j Documentation
- GitHub Repository
- Examples Repository

🆘 Getting Help

- GitHub Discussions
- Discord Community
- Stack Overflow: `langchain4j` tag

Course Repository

💻 Training Materials

`https://github.com/kousen/langchain4j-training`

📖 Labs Documentation

All labs with starter code and solutions

🎯 Hands-On Exercises

Progressive learning path from basics to production

Thank You!

Questions?

Kenneth Kousen*Author, Speaker, Java & AI Expert*

kousenit.com | @kenkousen