AI Java Labs: Building Intelligent Java Applications

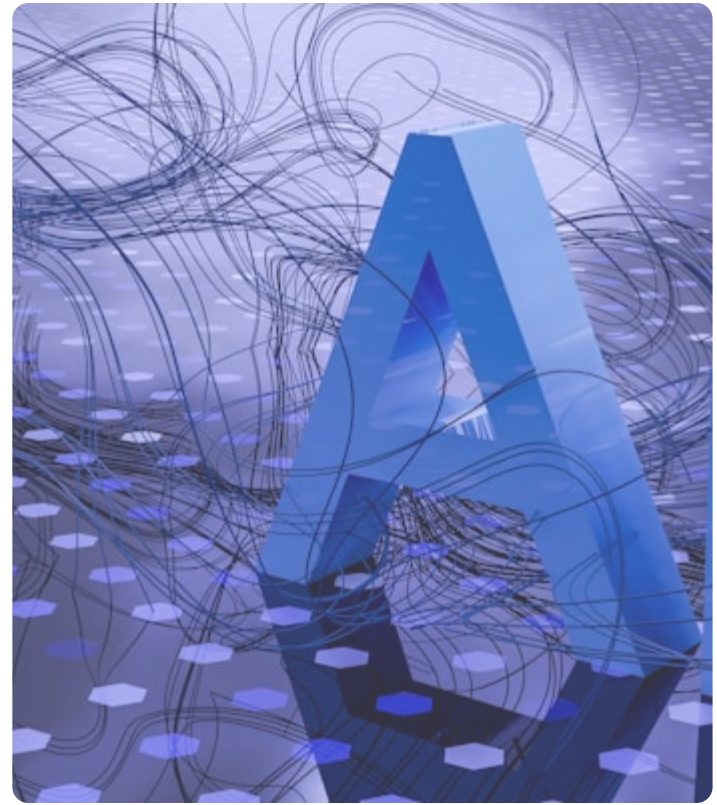
From HTTP fundamentals to advanced RAG systems →

Contact Info

Ken Kousen
Kousen IT, Inc.

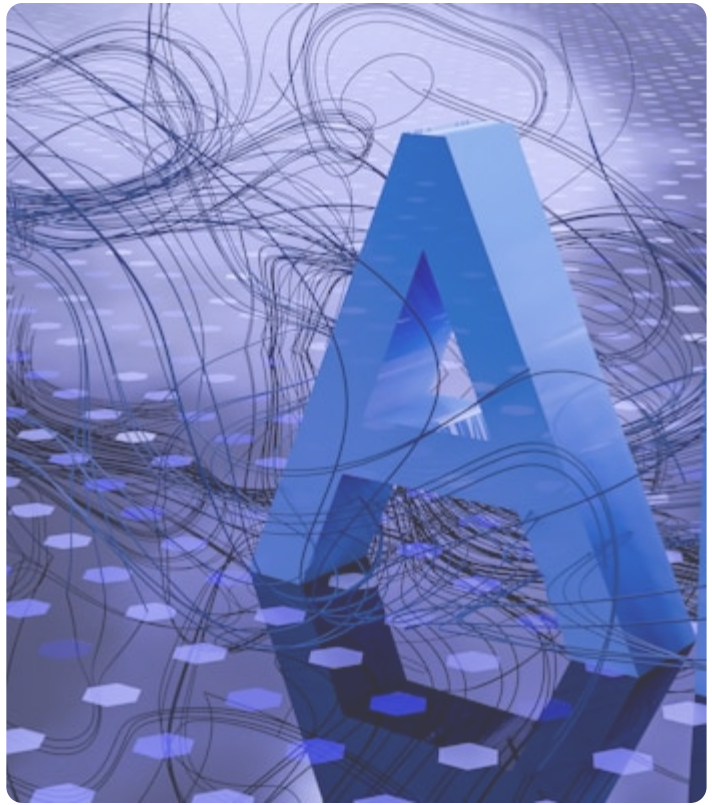
- ken.kousen@kousenit.com
- http://www.kousenit.com
- http://kousenit.org (blog)
- Social Media:
  - @kenkousen (Twitter)
  - @kousenit.com (Bluesky)
  - https://www.linkedin.com/in/kenkousen/ (LinkedIn)
- *Tales from the jar side* (free newsletter)
  - https://kenkousen.substack.com
  - https://youtube.com/@talesfromthejarside

What You'll Learn



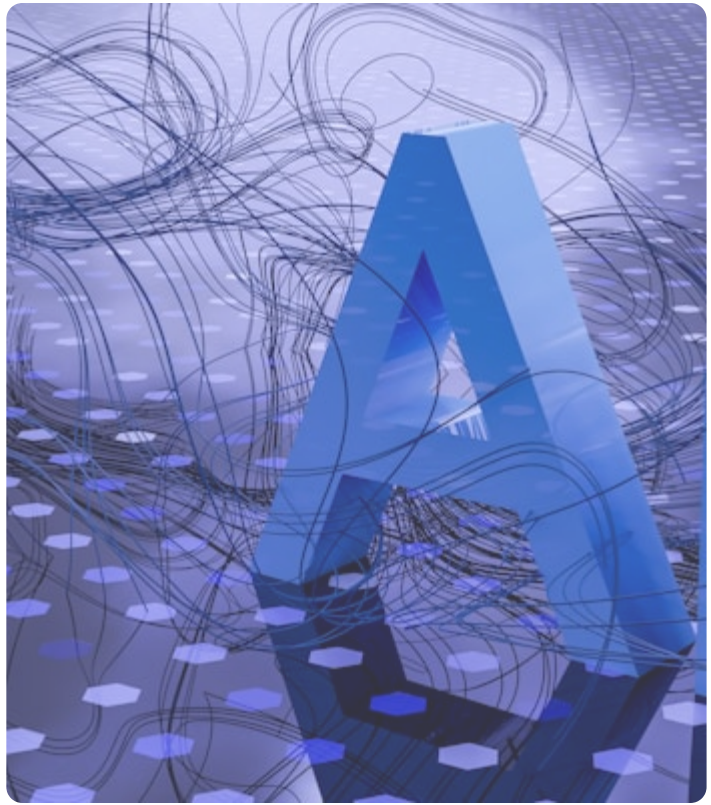
What You’ll Learn

- **HTTP Fundamentals**: Raw API integration patterns



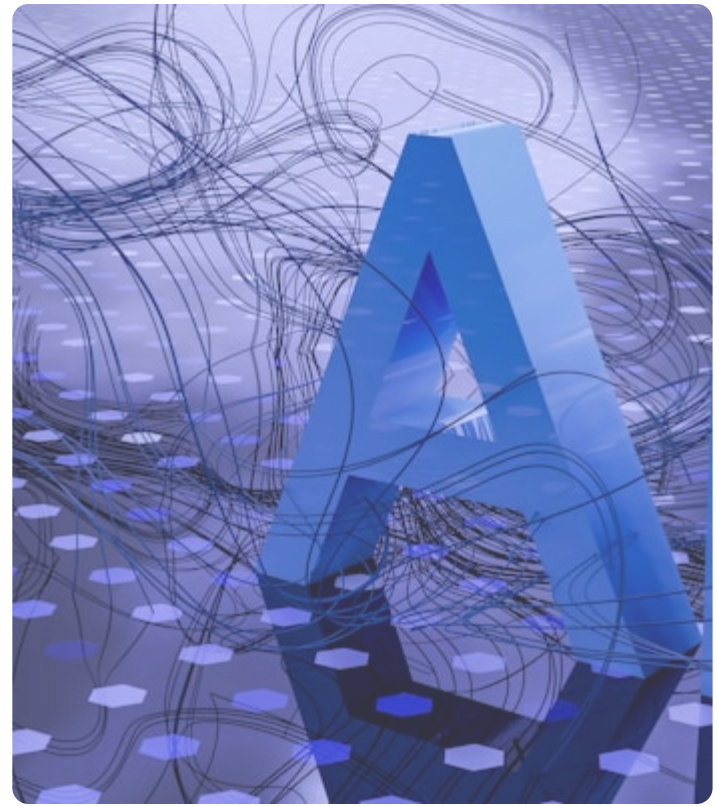
What You'll Learn

- **HTTP Fundamentals**: Raw API integration patterns
- **LangChain4j Framework**: High-level AI abstractions



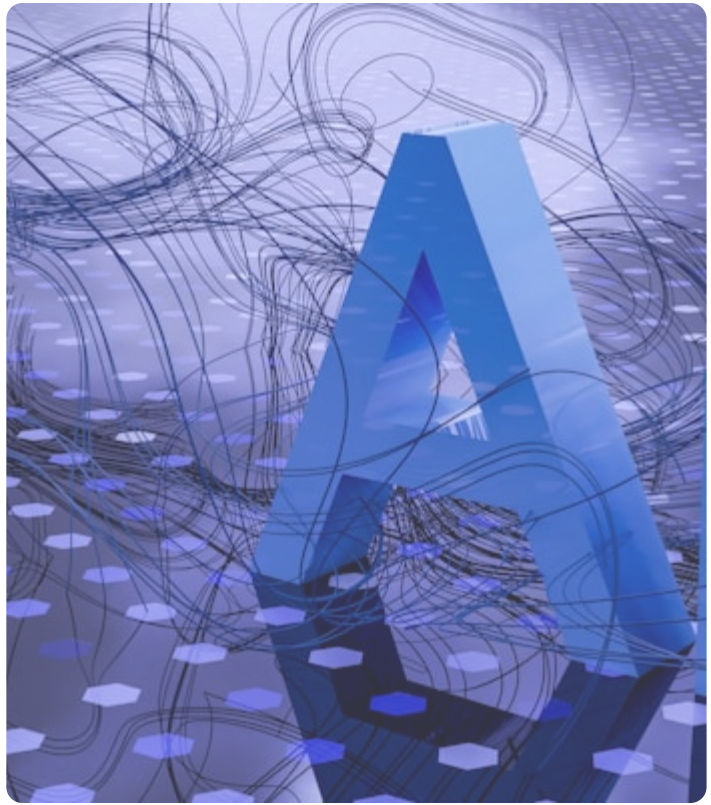
What You'll Learn

- **HTTP Fundamentals**: Raw API integration patterns
- **LangChain4j Framework**: High-level AI abstractions
- **Text Generation**: OpenAI and Ollama models



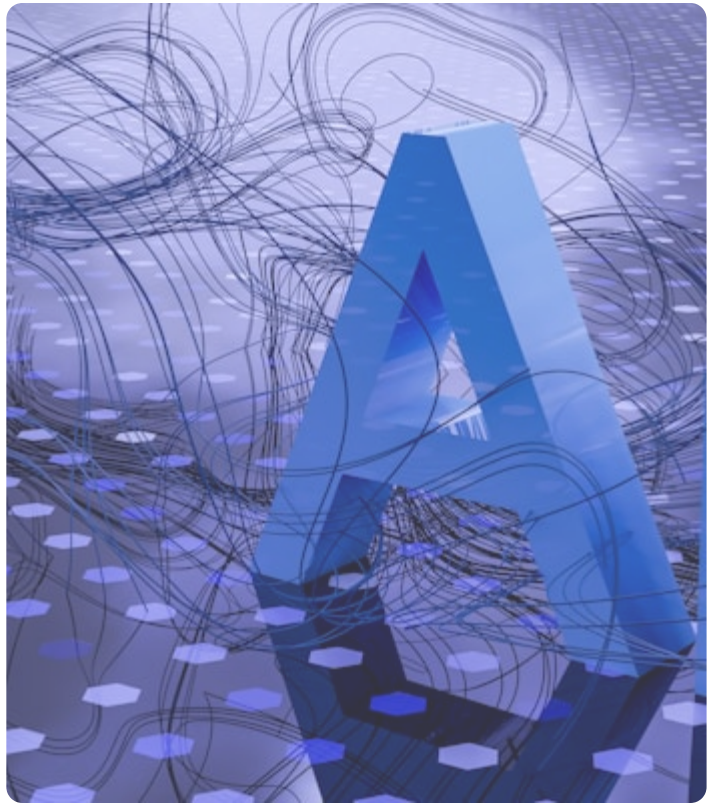
What You'll Learn

- **HTTP Fundamentals**: Raw API integration patterns
- **LangChain4j Framework**: High-level AI abstractions
- **Text Generation**: OpenAI and Ollama models
- **Streaming Responses**: Real-time token-by-token output



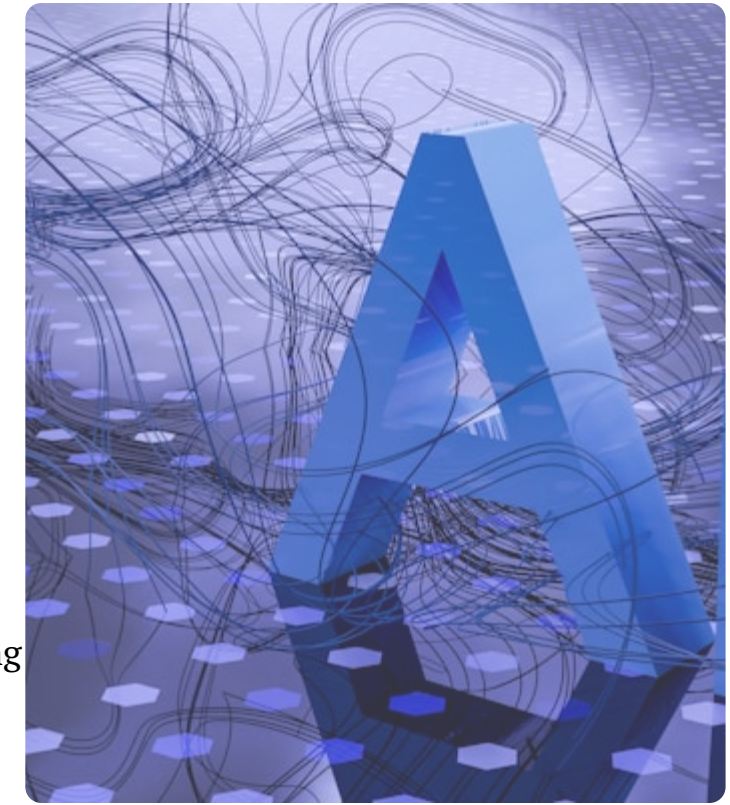
What You'll Learn

- **HTTP Fundamentals**: Raw API integration patterns
- **LangChain4j Framework**: High-level AI abstractions
- **Text Generation**: OpenAI and Ollama models
- **Streaming Responses**: Real-time token-by-token output
- **Multimodal AI**: Vision, audio, and image generation



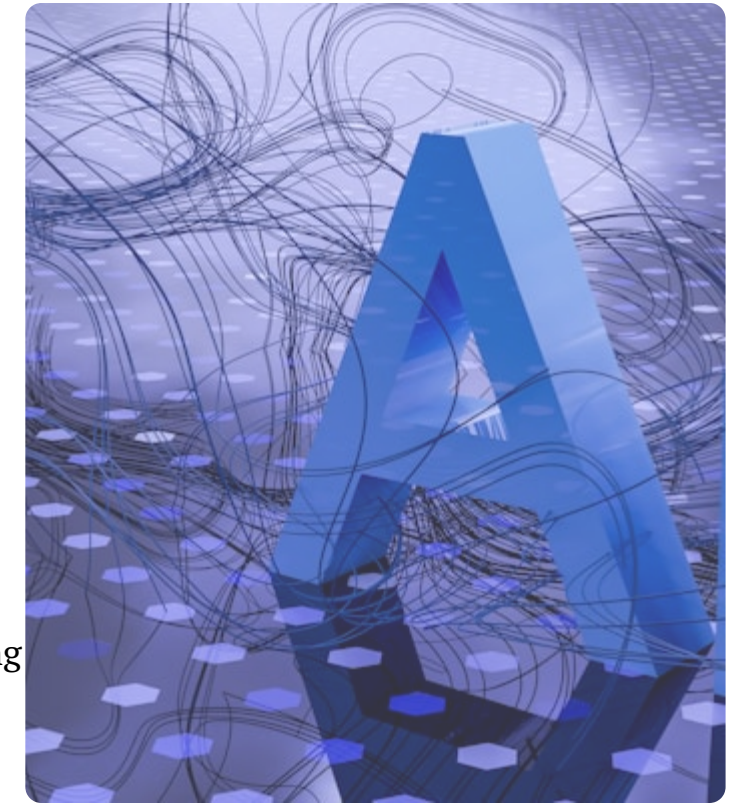
What You'll Learn

- **HTTP Fundamentals**: Raw API integration patterns
- **LangChain4j Framework**: High-level AI abstractions
- **Text Generation**: OpenAI and Ollama models
- **Streaming Responses**: Real-time token-by-token output
- **Multimodal AI**: Vision, audio, and image generation
- **RAG Systems**: Document-based question answering



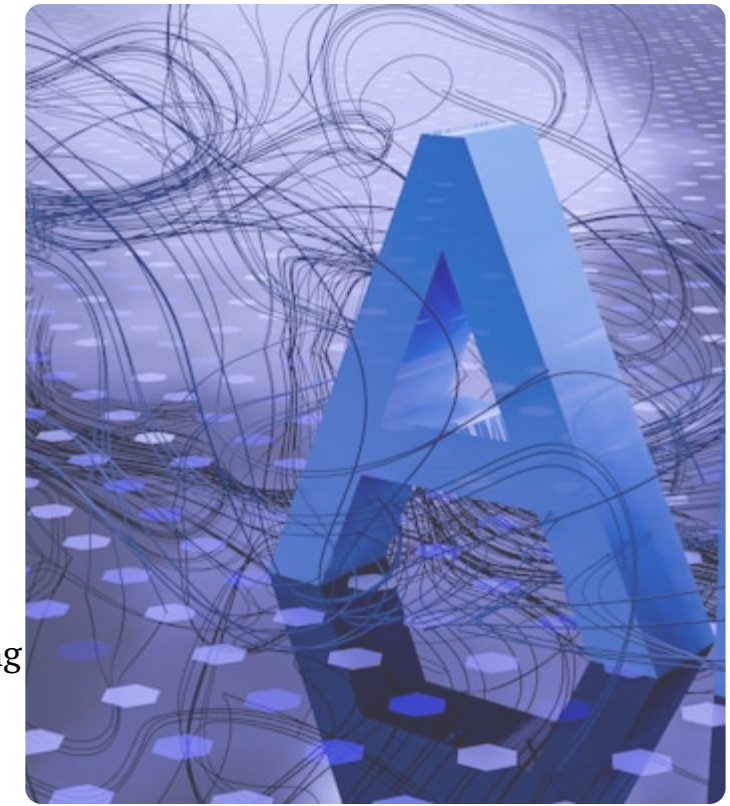
What You'll Learn

- **HTTP Fundamentals**: Raw API integration patterns
- **LangChain4j Framework**: High-level AI abstractions
- **Text Generation**: OpenAI and Ollama models
- **Streaming Responses**: Real-time token-by-token output
- **Multimodal AI**: Vision, audio, and image generation
- **RAG Systems**: Document-based question answering
- **Cost Management**: Efficient testing and model selection



What You'll Learn

- **HTTP Fundamentals**: Raw API integration patterns
- **LangChain4j Framework**: High-level AI abstractions
- **Text Generation**: OpenAI and Ollama models
- **Streaming Responses**: Real-time token-by-token output
- **Multimodal AI**: Vision, audio, and image generation
- **RAG Systems**: Document-based question answering
- **Cost Management**: Efficient testing and model selection
- **Modern Java**: Records, sealed interfaces, pattern matching



Repository Structure

```
AiJavaLabs/
├── labs.md                    # 15 progressive lab exercises
├── src/
│   ├── main/java/com/kousenit/
│   │   ├── demos/
│   │   │   ├── QuickChatDemo.java      # Fast OpenAI demo
│   │   │   ├── TextToSpeechDemo.java   # TTS generation
│   │   │   ├── LocalOllamaDemo.java    # Local AI models
│   │   │   ├── MultiModelDemo.java     # Provider comparison
│   │   │   ├── ResponsesApiDemo.java   # Raw HTTP examples
│   │   │   ├── ApiComparisonDemo.java  # Framework vs raw
│   │   │   └── StreamingDemo.java      # Real-time responses
│   │   ├── EasyRAGDemo.java           # Document Q&A
│   │   ├── *Service.java              # Core implementations
│   │   └── *Records.java              # Data models
│   └── test/java/              # Comprehensive test suite
├── build.gradle.kts           # Test categories for cost control
└── slides.md                  # This presentation
```

Repository Structure

```
AiJavaLabs/
├── labs.md                    # 15 progressive lab exercises
├── src/
│   ├── main/java/com/kousenit/
│   │   ├── demos/
│   │   │   ├── QuickChatDemo.java      # Fast OpenAI demo
│   │   │   ├── TextToSpeechDemo.java   # TTS generation
│   │   │   ├── LocalOllamaDemo.java    # Local AI models
│   │   │   ├── MultiModelDemo.java     # Provider comparison
│   │   │   ├── ResponsesApiDemo.java   # Raw HTTP examples
│   │   │   ├── ApiComparisonDemo.java  # Framework vs raw
│   │   │   └── StreamingDemo.java      # Real-time responses
│   │   ├── EasyRAGDemo.java           # Document Q&A
│   │   ├── *Service.java              # Core implementations
│   │   └── *Records.java              # Data models
│   └── test/java/              # Comprehensive test suite
├── build.gradle.kts           # Test categories for cost control
└── slides.md                  # This presentation
```

- **8 Live Demos**: Ready-to-run examples for each topic

Repository Structure

```
AiJavaLabs/
├── labs.md                    # 15 progressive lab exercises
├── src/
│   ├── main/java/com/kousenit/
│   │   ├── demos/
│   │   │   ├── QuickChatDemo.java      # Fast OpenAI demo
│   │   │   ├── TextToSpeechDemo.java   # TTS generation
│   │   │   ├── LocalOllamaDemo.java    # Local AI models
│   │   │   ├── MultiModelDemo.java     # Provider comparison
│   │   │   ├── ResponsesApiDemo.java   # Raw HTTP examples
│   │   │   ├── ApiComparisonDemo.java  # Framework vs raw
│   │   │   └── StreamingDemo.java      # Real-time responses
│   │   ├── EasyRAGDemo.java           # Document Q&A
│   │   ├── *Service.java              # Core implementations
│   │   └── *Records.java              # Data models
│   └── test/java/              # Comprehensive test suite
├── build.gradle.kts           # Test categories for cost control
└── slides.md                  # This presentation
```

- **8 Live Demos**: Ready-to-run examples for each topic
- **Cost-Controlled Testing**: Free local tests, cheap API tests

Repository Structure

```
AiJavaLabs/
├── labs.md                    # 15 progressive lab exercises
├── src/
│   ├── main/java/com/kousenit/
│   │   ├── demos/
│   │   │   ├── QuickChatDemo.java      # Fast OpenAI demo
│   │   │   ├── TextToSpeechDemo.java   # TTS generation
│   │   │   ├── LocalOllamaDemo.java    # Local AI models
│   │   │   ├── MultiModelDemo.java     # Provider comparison
│   │   │   ├── ResponsesApiDemo.java   # Raw HTTP examples
│   │   │   ├── ApiComparisonDemo.java  # Framework vs raw
│   │   │   └── StreamingDemo.java      # Real-time responses
│   │   ├── EasyRAGDemo.java           # Document Q&A
│   │   ├── *Service.java              # Core implementations
│   │   └── *Records.java              # Data models
│   └── test/java/              # Comprehensive test suite
├── build.gradle.kts           # Test categories for cost control
└── slides.md                  # This presentation
```

- **8 Live Demos**: Ready-to-run examples for each topic
- **Cost-Controlled Testing**: Free local tests, cheap API tests
- **Modern Java 21**: Records, sealed interfaces, pattern matching

Repository Structure

```
AiJavaLabs/
├── labs.md                    # 15 progressive lab exercises
├── src/
│   ├── main/java/com/kousenit/
│   │   ├── demos/
│   │   │   ├── QuickChatDemo.java      # Fast OpenAI demo
│   │   │   ├── TextToSpeechDemo.java   # TTS generation
│   │   │   ├── LocalOllamaDemo.java    # Local AI models
│   │   │   ├── MultiModelDemo.java     # Provider comparison
│   │   │   ├── ResponsesApiDemo.java   # Raw HTTP examples
│   │   │   ├── ApiComparisonDemo.java  # Framework vs raw
│   │   │   └── StreamingDemo.java      # Real-time responses
│   │   ├── EasyRAGDemo.java           # Document Q&A
│   │   ├── *Service.java              # Core implementations
│   │   └── *Records.java              # Data models
│   └── test/java/              # Comprehensive test suite
├── build.gradle.kts           # Test categories for cost control
└── slides.md                  # This presentation
```

- **8 Live Demos**: Ready-to-run examples for each topic
- **Cost-Controlled Testing**: Free local tests, cheap API tests
- **Modern Java 21**: Records, sealed interfaces, pattern matching

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First? **Then Master Frameworks**

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood

Then Master Frameworks

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood
- **Debugging Skills**: When frameworks fail, you can troubleshoot

Then Master Frameworks

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood
- **Debugging Skills**: When frameworks fail, you can troubleshoot
- **Flexibility**: Not locked into any single framework

Then Master Frameworks

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood
- **Debugging Skills**: When frameworks fail, you can troubleshoot
- **Flexibility**: Not locked into any single framework
- **Performance**: Optimize for your specific needs

Then Master Frameworks

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood
- **Debugging Skills**: When frameworks fail, you can troubleshoot
- **Flexibility**: Not locked into any single framework
- **Performance**: Optimize for your specific needs

Then Master Frameworks

- **Productivity**: LangChain4j handles boilerplate

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood
- **Debugging Skills**: When frameworks fail, you can troubleshoot
- **Flexibility**: Not locked into any single framework
- **Performance**: Optimize for your specific needs

Then Master Frameworks

- **Productivity**: LangChain4j handles boilerplate
- **Best Practices**: Proven patterns and abstractions

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood
- **Debugging Skills**: When frameworks fail, you can troubleshoot
- **Flexibility**: Not locked into any single framework
- **Performance**: Optimize for your specific needs

Then Master Frameworks

- **Productivity**: LangChain4j handles boilerplate
- **Best Practices**: Proven patterns and abstractions
- **Advanced Features**: RAG, streaming, memory

Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood
- **Debugging Skills**: When frameworks fail, you can troubleshoot
- **Flexibility**: Not locked into any single framework
- **Performance**: Optimize for your specific needs

Then Master Frameworks

- **Productivity**: LangChain4j handles boilerplate
- **Best Practices**: Proven patterns and abstractions
- **Advanced Features**: RAG, streaming, memory
- **Rapid Development**: Focus on business logic

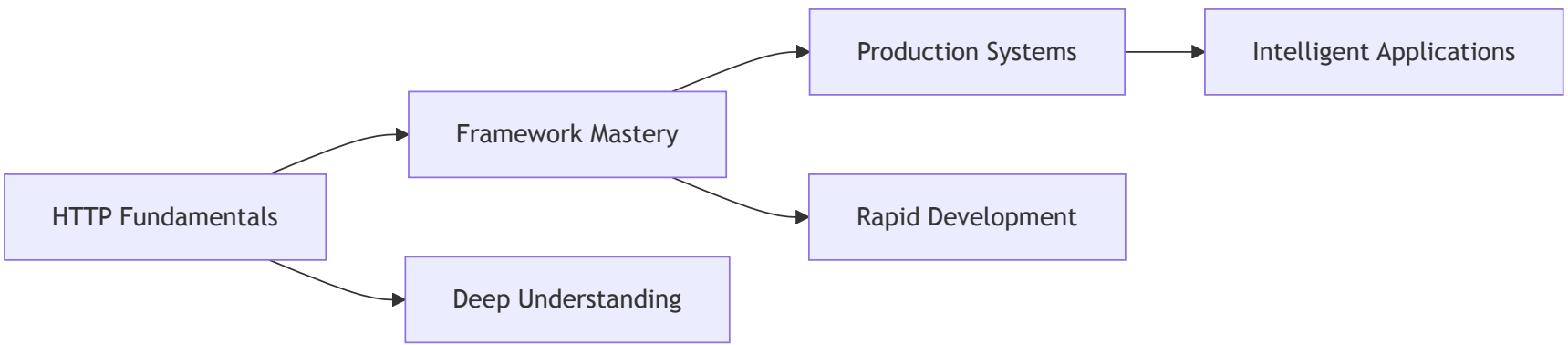
Course Philosophy: Understanding Both Levels

Why Learn Raw HTTP First?

- **True Understanding**: Know what frameworks do under the hood
- **Debugging Skills**: When frameworks fail, you can troubleshoot
- **Flexibility**: Not locked into any single framework
- **Performance**: Optimize for your specific needs

Then Master Frameworks

- **Productivity**: LangChain4j handles boilerplate
- **Best Practices**: Proven patterns and abstractions
- **Advanced Features**: RAG, streaming, memory
- **Rapid Development**: Focus on business logic



Prerequisites

Technical Requirements

Environment Setup

```
# Required API keys
export OPENAI_API_KEY=your_key

# Optional: Local AI models
curl -fsSL https://ollama.com/install.sh | sh
ollama pull gemma3
ollama pull moondream  # For vision

# Clone and start
git clone <repo-url>
./gradlew build
./gradlew testDemo  # Quick validation
```

Prerequisites

Technical Requirements

- **Java 21+** (Records, sealed interfaces, pattern matching)

Environment Setup

```
# Required API keys
export OPENAI_API_KEY=your_key

# Optional: Local AI models
curl -fsSL https://ollama.com/install.sh | sh
ollama pull gemma3
ollama pull moondream  # For vision

# Clone and start
git clone <repo-url>
./gradlew build
./gradlew testDemo  # Quick validation
```

Prerequisites

Technical Requirements

- **Java 21+** (Records, sealed interfaces, pattern matching)
- **Gradle 8.4+** (Kotlin DSL, test categories)

Environment Setup

```
# Required API keys
export OPENAI_API_KEY=your_key

# Optional: Local AI models
curl -fsSL https://ollama.com/install.sh | sh
ollama pull gemma3
ollama pull moondream  # For vision

# Clone and start
git clone <repo-url>
./gradlew build
./gradlew testDemo  # Quick validation
```

Prerequisites

Technical Requirements

- **Java 21+** (Records, sealed interfaces, pattern matching)
- **Gradle 8.4+** (Kotlin DSL, test categories)
- **LangChain4j 1.4.0** (Latest AI framework)

Environment Setup

```
# Required API keys
export OPENAI_API_KEY=your_key

# Optional: Local AI models
curl -fsSL https://ollama.com/install.sh | sh
ollama pull gemma3
ollama pull moondream  # For vision

# Clone and start
git clone <repo-url>
./gradlew build
./gradlew testDemo  # Quick validation
```

Prerequisites

Technical Requirements

- **Java 21+** (Records, sealed interfaces, pattern matching)
- **Gradle 8.4+** (Kotlin DSL, test categories)
- **LangChain4j 1.4.0** (Latest AI framework)
- **Git** for repository management

Environment Setup

```
# Required API keys
export OPENAI_API_KEY=your_key

# Optional: Local AI models
curl -fsSL https://ollama.com/install.sh | sh
ollama pull gemma3
ollama pull moondream  # For vision

# Clone and start
git clone <repo-url>
./gradlew build
./gradlew testDemo  # Quick validation
```

Prerequisites

Technical Requirements

- **Java 21+** (Records, sealed interfaces, pattern matching)
- **Gradle 8.4+** (Kotlin DSL, test categories)
- **LangChain4j 1.4.0** (Latest AI framework)
- **Git** for repository management
- **Ollama** (optional, for local AI models)

Environment Setup

```
# Required API keys
export OPENAI_API_KEY=your_key

# Optional: Local AI models
curl -fsSL https://ollama.com/install.sh | sh
ollama pull gemma3
ollama pull moondream  # For vision

# Clone and start
git clone <repo-url>
./gradlew build
./gradlew testDemo  # Quick validation
```

Cost Management Strategy

Test Categories

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests
- `@Tag("cheap")` - **Low-cost** nano models

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests
- `@Tag("cheap")` - **Low-cost** nano models
- `@Tag("demo")` - **Fast** live demos

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests
- `@Tag("cheap")` - **Low-cost** nano models
- `@Tag("demo")` - **Fast** live demos
- `@Tag("expensive")` - **Avoided** by default

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests
- `@Tag("cheap")` - **Low-cost** nano models
- `@Tag("demo")` - **Fast** live demos
- `@Tag("expensive")` - **Avoided** by default

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

- gpt-4.1-nano: $0.150/1M input tokens

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests
- `@Tag("cheap")` - **Low-cost** nano models
- `@Tag("demo")` - **Fast** live demos
- `@Tag("expensive")` - **Avoided** by default

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

- **gpt-4.1-nano**: $0.150/1M input tokens
- **gemma3 (local)**: Free with Ollama

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests
- `@Tag("cheap")` - **Low-cost** nano models
- `@Tag("demo")` - **Fast** live demos
- `@Tag("expensive")` - **Avoided** by default

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

- **gpt-4.1-nano**: $0.150/1M input tokens
- **gemma3 (local)**: Free with Ollama
- **dall-e-3**: $0.040 per image

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests
- `@Tag("cheap")` - **Low-cost** nano models
- `@Tag("demo")` - **Fast** live demos
- `@Tag("expensive")` - **Avoided** by default

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

- **gpt-4.1-nano**: $0.150/1M input tokens
- **gemma3 (local)**: Free with Ollama
- **dall-e-3**: $0.040 per image
- **tts-1**: $0.015 per 1K chars

Cost Management Strategy

Test Categories

- `@Tag("local")` - **Free** Ollama tests
- `@Tag("cheap")` - **Low-cost** nano models
- `@Tag("demo")` - **Fast** live demos
- `@Tag("expensive")` - **Avoided** by default

Gradle Tasks

```
./gradlew testLocal       # $0
./gradlew testCheap       # ~$0.0
./gradlew testDemo        # ~$0.0
./gradlew testNotExpensive
./gradlew testOpenAI
```

Model Costs

- **gpt-4.1-nano**: $0.150/1M input tokens
- **gemma3 (local)**: Free with Ollama
- **dall-e-3**: $0.040 per image
- **tts-1**: $0.015 per 1K chars

Smart Strategy: Develop with free local models, deploy with optimal cloud models

Demo 1-2: HTTP Fundamentals

Understanding Raw API Integration

Demo 1: Text-to-Speech with Raw HTTP

```java
// Raw HTTP approach - Understanding the fundamentals
public class TextToSpeechService {
    private static final String OPENAI_API_KEY = System.getenv("OPENAI_API_KEY");
    private static final HttpClient client = HttpClient.newHttpClient();

    public Path generateMp3(String model, String input, String voice) {
        // TODO: How do we construct the request?
    }
}
```

Demo 1: Text-to-Speech with Raw HTTP

```
// Step 1: Create JSON payload manually
String payload = """
    {
        "model": "%s",
        "input": "%s",
        "voice": "%s"
    }
    """.formatted(model, input.replaceAll("\\s+", " ").trim(), voice);
```

Demo 1: Text-to-Speech with Raw HTTP

```
// Step 2: Build HTTP request with proper headers
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create("https://api.openai.com/v1/audio/speech"))
    .header("Authorization", "Bearer %s".formatted(OPENAI_API_KEY))
    .header("Content-Type", "application/json")
    .header("Accept", "audio/mpeg")
    .POST(HttpRequest.BodyPublishers.ofString(payload))
    .build();
```

Demo 1: Text-to-Speech with Raw HTTP

```
// Step 3: Complete implementation with file handling
public Path generateMp3(String model, String input, String voice) {
    String payload = """
        {
            "model": "%s",
            "input": "%s",
            "voice": "%s"
        }
        """.formatted(model, input.replaceAll("\\s+", " ").trim(), voice);

    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("https://api.openai.com/v1/audio/speech"))
        .header("Authorization", "Bearer %s".formatted(OPENAI_API_KEY))
        .header("Content-Type", "application/json")
        .header("Accept", "audio/mpeg")
        .POST(HttpRequest.BodyPublishers.ofString(payload))
        .build();

    try {
        HttpResponse<Path> response =
            client.send(request, HttpResponse.BodyHandlers.ofFile(getFilePath()));
        return response.body();
    } catch (IOException | InterruptedException e) {
        throw new RuntimeException(e);
```

Demo 1: Text-to-Speech with Raw HTTP

```
// Step 3: Complete implementation with file handling
public Path generateMp3(String model, String input, String voice) {
    String payload = """
        {
            "model": "%s",
            "input": "%s",
            "voice": "%s"
        }
        """.formatted(model, input.replaceAll("\\s+", " ").trim(), voice);

    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("https://api.openai.com/v1/audio/speech"))
        .header("Authorization", "Bearer %s".formatted(OPENAI_API_KEY))
        .header("Content-Type", "application/json")
        .header("Accept", "audio/mpeg")
        .POST(HttpRequest.BodyPublishers.ofString(payload))
        .build();

    try {
        HttpResponse<Path> response =
            client.send(request, HttpResponse.BodyHandlers.ofFile(getFilePath()));
        return response.body();
    } catch (IOException | InterruptedException e) {
        throw new RuntimeException(e);
```

Demo 2: Model Discovery & JSON Parsing

```java
// Understanding API responses with Java records
public class OpenAiRecords {
    public record ModelList(List<Model> data) {
        public record Model(
                String id,
                long created,
                @SerializedName("owned_by") String ownedBy) {
        }
    }
}
```

Demo 2: Model Discovery & JSON Parsing

```java
// Service implementation with Gson parsing
public class OpenAiService {
    private static final HttpClient client = HttpClient.newHttpClient();
    private final Gson gson = new GsonBuilder()
            .setFieldNamingPolicy(FieldNamingPolicy.LOWER_CASE_WITH_UNDERSCORES)
            .create();

    public ModelList listModels() throws IOException, InterruptedException {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create("https://api.openai.com/v1/models"))
            .header("Authorization", "Bearer " + System.getenv("OPENAI_API_KEY"))
            .GET()
            .build();

        HttpResponse<String> response = client.send(request,
                HttpResponse.BodyHandlers.ofString());
        return gson.fromJson(response.body(), ModelList.class);
    }
}
            .build();

        try {
            HttpResponse<String> response =
                client.send(request, HttpResponse.BodyHandlers.ofString());
```

Demo 2: Model Discovery & JSON Parsing

```java
// Service implementation with Gson parsing
public class OpenAiService {
    private static final HttpClient client = HttpClient.newHttpClient();
    private final Gson gson = new GsonBuilder()
            .setFieldNamingPolicy(FieldNamingPolicy.LOWER_CASE_WITH_UNDERSCORES)
            .create();

    public ModelList listModels() throws IOException, InterruptedException {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create("https://api.openai.com/v1/models"))
            .header("Authorization", "Bearer " + System.getenv("OPENAI_API_KEY"))
            .GET()
            .build();

        HttpResponse<String> response = client.send(request,
                HttpResponse.BodyHandlers.ofString());
        return gson.fromJson(response.body(), ModelList.class);
    }
}
            .build();

        try {
            HttpResponse<String> response =
                client.send(request, HttpResponse.BodyHandlers.ofString());
```

JSON Parsing: Gson vs Jackson

Gson Approach

```
// Using JsonElement tree navigation
JsonElement root = JsonParser.parseString(json);
JsonObject obj = root.getAsJsonObject();
JsonArray outputs = obj.getAsJsonArray("output");

for (JsonElement elem : outputs) {
    JsonObject item = elem.getAsJsonObject();
    if ("message".equals(item.get("type").getAsString
        // Extract nested content...
    }
}
```

Jackson Approach

```
// Using JsonNode with at() method
ObjectMapper mapper = new ObjectMapper();
JsonNode root = mapper.readTree(json);

// Direct path with JSON Pointer!
String text = root.at("/output/0/content/0/text")
                  .asText();

// Or safe navigation with path()
root.path("output")
    .path(0)
    .path("content")
    .path(0)
    .path("text");
```

JSON Parsing: Gson vs Jackson

Gson Approach

```
// Using JsonElement tree navigation
JsonElement root = JsonParser.parseString(json);
JsonObject obj = root.getAsJsonObject();
JsonArray outputs = obj.getAsJsonArray("output");

for (JsonElement elem : outputs) {
    JsonObject item = elem.getAsJsonObject();
    if ("message".equals(item.get("type").getAsString
        // Extract nested content...
    }
}
```

- Type-safe navigation
- Explicit type conversions
- Manual null checking needed

Jackson Approach

```
// Using JsonNode with at() method
ObjectMapper mapper = new ObjectMapper();
JsonNode root = mapper.readTree(json);

// Direct path with JSON Pointer!
String text = root.at("/output/0/content/0/text")
                  .asText();

// Or safe navigation with path()
root.path("output")
    .path(0)
    .path("content")
    .path(0)
    .path("text");
```

JSON Parsing: Gson vs Jackson

Gson Approach

```java
// Using JsonElement tree navigation
JsonElement root = JsonParser.parseString(json);
JsonObject obj = root.getAsJsonObject();
JsonArray outputs = obj.getAsJsonArray("output");

for (JsonElement elem : outputs) {
    JsonObject item = elem.getAsJsonObject();
    if ("message".equals(item.get("type").getAsString
        // Extract nested content...
    }
}
```

- Type-safe navigation
- Explicit type conversions
- Manual null checking needed

Jackson Approach

```java
// Using JsonNode with at() method
ObjectMapper mapper = new ObjectMapper();
JsonNode root = mapper.readTree(json);

// Direct path with JSON Pointer!
String text = root.at("/output/0/content/0/text")
                  .asText();

// Or safe navigation with path()
root.path("output")
    .path(0)
    .path("content")
    .path(0)
    .path("text");
```

- JSON Pointer navigation (`at()`)
- Safe chaining with `path()`
- No null pointer exceptions

JSON Pointer: Elegant Deep Navigation

RFC 6901 JSON Pointer Syntax

JSON Pointer: Elegant Deep Navigation

RFC 6901 JSON Pointer Syntax

- **Direct path access**: `/output/0/content/0/text`

JSON Pointer: Elegant Deep Navigation

RFC 6901 JSON Pointer Syntax

- **Direct path access**: `/output/0/content/0/text`
- **Array indexing**: Use numbers for array elements

JSON Pointer: Elegant Deep Navigation

RFC 6901 JSON Pointer Syntax

- **Direct path access**: `/output/0/content/0/text`
- **Array indexing**: Use numbers for array elements
- **Nested objects**: Chain with `/` separator

JSON Pointer: Elegant Deep Navigation

RFC 6901 JSON Pointer Syntax

- **Direct path access**: `/output/0/content/0/text`
- **Array indexing**: Use numbers for array elements
- **Nested objects**: Chain with `/` separator

```java
// Complex nested structure navigation - one line!
JsonNode root = mapper.readTree(response.body());

// Instead of multiple loops and null checks:
String text = root.at("/data/results/0/attributes/name").asText();

// With fallback:
String value = root.at("/missing/path").asText("default");
```

JSON Pointer: Elegant Deep Navigation

RFC 6901 JSON Pointer Syntax

- **Direct path access**: `/output/0/content/0/text`
- **Array indexing**: Use numbers for array elements
- **Nested objects**: Chain with `/` separator

```
// Complex nested structure navigation - one line!
JsonNode root = mapper.readTree(response.body());

// Instead of multiple loops and null checks:
String text = root.at("/data/results/0/attributes/name").asText();

// With fallback:
String value = root.at("/missing/path").asText("default");
```

When to Use Which?

- **Gson**: Already in project, simpler API, Google ecosystem
- **Jackson**: Spring Boot default, JSON Pointer support, more features
- **Both**: Can coexist - use what fits your needs!

HTTP vs Framework Comparison

Raw HTTP Approach

```
// Manual request construction
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create(url))
    .header("Authorization", "Bearer " + key)
    .header("Content-Type", "application/json")
    .POST(HttpRequest.BodyPublishers.ofString(
        gson.toJson(payload)))
    .build();

// Manual response handling
HttpResponse<String> response =
    client.send(request, HttpResponse.BodyHandlers.
return gson.fromJson(response.body(), ResponseClass
```

LangChain4j Framework

```
// High-level abstraction
ChatModel model = OpenAiChatModel.builder()
    .apiKey(System.getenv("OPENAI_API_KEY"))
    .modelName("gpt-4.1-nano")
    .build();

// Simple method call
String response = model.chat("Hello, world!");
```

HTTP vs Framework Comparison

Raw HTTP Approach

```
// Manual request construction
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create(url))
    .header("Authorization", "Bearer " + key)
    .header("Content-Type", "application/json")
    .POST(HttpRequest.BodyPublishers.ofString(
        gson.toJson(payload)))
    .build();

// Manual response handling
HttpResponse<String> response =
    client.send(request, HttpResponse.BodyHandlers.
return gson.fromJson(response.body(), ResponseClass
```

LangChain4j Framework

```
// High-level abstraction
ChatModel model = OpenAiChatModel.builder()
    .apiKey(System.getenv("OPENAI_API_KEY"))
    .modelName("gpt-4.1-nano")
    .build();

// Simple method call
String response = model.chat("Hello, world!");
```

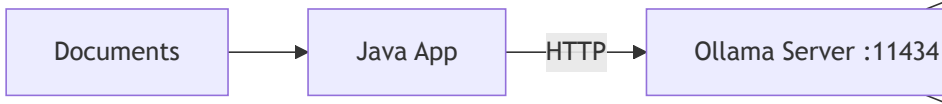
Both are valuable: Raw HTTP for understanding, frameworks for productivity

Demo 3-4: Local AI with Ollama

Privacy-First AI Integration

Demo 3: Ollama Setup and Architecture

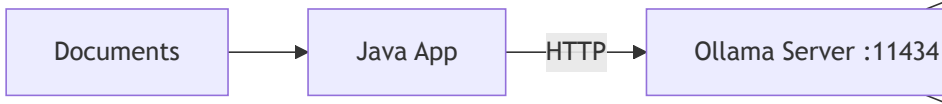
Why Ollama?



Demo 3: Ollama Setup and Architecture

Why Ollama?

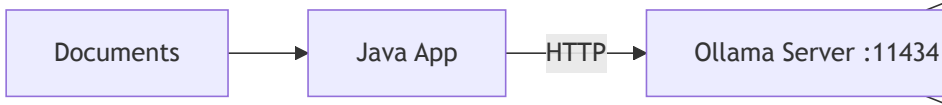
- **Privacy**: No data leaves your machine



Demo 3: Ollama Setup and Architecture

Why Ollama?

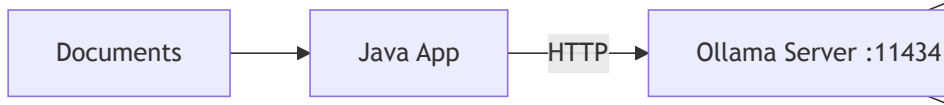
- **Privacy**: No data leaves your machine
- **Cost**: Zero API costs after setup



Demo 3: Ollama Setup and Architecture

Why Ollama?

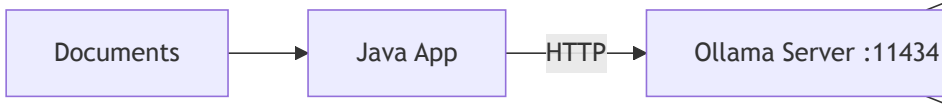
- **Privacy**: No data leaves your machine
- **Cost**: Zero API costs after setup
- **Control**: Full model customization



Demo 3: Ollama Setup and Architecture

Why Ollama?

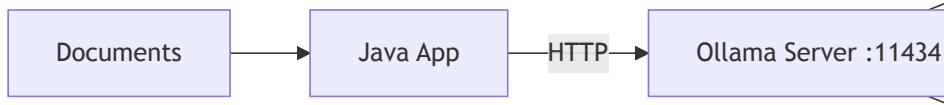
- **Privacy**: No data leaves your machine
- **Cost**: Zero API costs after setup
- **Control**: Full model customization
- **Speed**: Local inference, no network delays



Demo 3: Ollama Setup and Architecture

Why Ollama?

- **Privacy**: No data leaves your machine
- **Cost**: Zero API costs after setup
- **Control**: Full model customization
- **Speed**: Local inference, no network delays



Setup Commands:

```
ollama pull gemma3        # Text generation
ollama pull moondream     # Vision analysis
ollama serve              # Start server (automatic on install)
```

Demo 4: Modern Java with Sealed Interfaces

```java
// Step 1: Model different request types with records
public class OllamaRecords {
    public record OllamaTextRequest(
            String model,
            String prompt,
            boolean stream) {}

    public record OllamaVisionRequest(
            String model,
            String prompt,
            boolean stream,
            List<String> images) {}
}
```

Demo 4: Modern Java with Sealed Interfaces

```java
// Step 2: Use sealed interfaces for type safety (Java 17+)
public sealed interface OllamaRequest
    permits OllamaTextRequest, OllamaVisionRequest {}

public record OllamaTextRequest(
        String model, String prompt, boolean stream)
        implements OllamaRequest {}

public record OllamaVisionRequest(
        String model, String prompt, boolean stream, List<String> images)
        implements OllamaRequest {

    // Compact constructor with Base64 encoding
    public OllamaVisionRequest {
        images = images.stream()
                .map(this::encodeImage)
                .collect(Collectors.toList());
    }
}
```

Demo 4: Modern Java with Sealed Interfaces

```java
// Step 3: Pattern matching with switch expressions (Java 21)
public OllamaResponse generate(OllamaRequest request) {
    switch (request) {
        case OllamaTextRequest textRequest -> {
            logger.log(INFO, "Text request: {0}", textRequest.prompt());
        }
        case OllamaVisionRequest visionRequest -> {
            logger.log(INFO, "Vision request with {0} images",
                visionRequest.images().size());
        }
        // Exhaustive - no default needed with sealed types!
    }

    // Same HTTP logic for both request types
    return sendRequest(request);
}
```

Demo 4: Modern Java with Sealed Interfaces

```java
// Step 3: Pattern matching with switch expressions (Java 21)
public OllamaResponse generate(OllamaRequest request) {
    switch (request) {
        case OllamaTextRequest textRequest -> {
            logger.log(INFO, "Text request: {0}", textRequest.prompt());
        }
        case OllamaVisionRequest visionRequest -> {
            logger.log(INFO, "Vision request with {0} images",
                visionRequest.images().size());
        }
        // Exhaustive - no default needed with sealed types!
    }

    // Same HTTP logic for both request types
    return sendRequest(request);
}
```

Modern Java Benefits: Type safety, exhaustive pattern matching, compact constructors

Demo 5-6: Framework Integration

LangChain4j Power and Simplicity

Demo 5: Quick Chat with LangChain4j

```java
// From 50+ lines of HTTP code to this:
import dev.langchain4j.model.chat.ChatModel;
import dev.langchain4j.model.openai.OpenAiChatModel;

public class QuickChatDemo {
    public static void main(String[] args) {
        // TODO: How do we create a chat model?
    }
}
```

Demo 5: Quick Chat with LangChain4j

```
// Step 1: Build the model with fluent API
ChatModel model = OpenAiChatModel.builder()
        .apiKey(System.getenv("OPENAI_API_KEY"))
        .modelName("gpt-4.1-nano")
        .logRequests(false)  // Toggle for debugging
        .logResponses(false) // Toggle for debugging
        .build();
```

Demo 5: Quick Chat with LangChain4j

```java
// Step 2: Complete implementation - just one method call!
public class QuickChatDemo {
    public static void main(String[] args) {
        ChatModel model = OpenAiChatModel.builder()
                .apiKey(System.getenv("OPENAI_API_KEY"))
                .modelName("gpt-4.1-nano")
                .build();

        String prompt = "Explain what LangChain4j is in 2-3 sentences.";
        System.out.println("Answer: " + model.chat(prompt));
    }
}
```

Demo 5: Quick Chat with LangChain4j

```java
// Step 2: Complete implementation - just one method call!
public class QuickChatDemo {
    public static void main(String[] args) {
        ChatModel model = OpenAiChatModel.builder()
                .apiKey(System.getenv("OPENAI_API_KEY"))
                .modelName("gpt-4.1-nano")
                .build();

        String prompt = "Explain what LangChain4j is in 2-3 sentences.";
        System.out.println("Answer: " + model.chat(prompt));
    }
}
```

Live Demo: Run `QuickChatDemo.java` - from complex HTTP to simple method call!

Demo 6: Multi-Model Provider Support

```
// Step 1: OpenAI implementation
ChatModel openai = OpenAiChatModel.builder()
        .apiKey(System.getenv("OPENAI_API_KEY"))
        .modelName("gpt-4.1-nano")
        .build();

String response = openai.chat("What is the capital of France?");
System.out.println("OpenAI: " + response);
```

Demo 6: Multi-Model Provider Support

```
// Step 2: Switch to Google Gemini - same interface!
ChatModel gemini = GoogleAiGeminiChatModel.builder()
        .apiKey(System.getenv("GOOGLEAI_API_KEY"))
        .modelName("gemini-2.0-flash-exp")
        .build();

String response = gemini.chat("What is the capital of France?");
System.out.println("Gemini: " + response);
```

Demo 6: Multi-Model Provider Support

```
// Step 3: Or use local Ollama - still same interface!
ChatModel ollama = OllamaChatModel.builder()
        .baseUrl("http://localhost:11434")
        .modelName("gemma3")
        .build();

String response = ollama.chat("What is the capital of France?");
System.out.println("Ollama: " + response);
```

Demo 6: Multi-Model Provider Support

```java
// Step 3: Or use local Ollama - still same interface!
ChatModel ollama = OllamaChatModel.builder()
        .baseUrl("http://localhost:11434")
        .modelName("gemma3")
        .build();

String response = ollama.chat("What is the capital of France?");
System.out.println("Ollama: " + response);
```

Key Insight: Same `ChatModel` interface = vendor-agnostic code!

Provider Ecosystem

Cloud Providers

Local Models

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3

Local Models

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM

Local Models

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet

Local Models

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT

Local Models

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral
- **Hugging Face** • Open source models

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral
- **Hugging Face** • Open source models
- **vLLM** • High-performance inference

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral
- **Hugging Face** • Open source models
- **vLLM** • High-performance inference
- **LocalAI** • OpenAI-compatible API

Specialized Services

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral
- **Hugging Face** • Open source models
- **vLLM** • High-performance inference
- **LocalAI** • OpenAI-compatible API

Specialized Services

- **Vision** • moondream, LLaVA

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral
- **Hugging Face** • Open source models
- **vLLM** • High-performance inference
- **LocalAI** • OpenAI-compatible API

Specialized Services

- **Vision** • moondream, LLaVA
- **Audio** • Whisper, TTS models

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral
- **Hugging Face** • Open source models
- **vLLM** • High-performance inference
- **LocalAI** • OpenAI-compatible API

Specialized Services

- **Vision** • moondream, LLaVA
- **Audio** • Whisper, TTS models
- **Embeddings** • sentence-transformers

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral
- **Hugging Face** • Open source models
- **vLLM** • High-performance inference
- **LocalAI** • OpenAI-compatible API

Specialized Services

- **Vision** • moondream, LLaVA
- **Audio** • Whisper, TTS models
- **Embeddings** • sentence-transformers
- **Code** • CodeLlama, StarCoder

Provider Ecosystem

Cloud Providers

- **OpenAI** • GPT-4.1-nano, DALL-E 3
- **Google AI** • Gemini 2.0, PaLM
- **Anthropic** • Claude 3.5 Sonnet
- **Azure OpenAI** • Enterprise GPT
- **AWS Bedrock** • Multiple models

Local Models

- **Ollama** • gemma3, llama3.1, mistral
- **Hugging Face** • Open source models
- **vLLM** • High-performance inference
- **LocalAI** • OpenAI-compatible API

Specialized Services

- **Vision** • moondream, LLaVA
- **Audio** • Whisper, TTS models
- **Embeddings** • sentence-transformers
- **Code** • CodeLlama, StarCoder

LangChain4j **Advantage**: Switch providers with just configuration changes!

Demo 7: Streaming Responses

Real-Time AI Interactions

Demo 7: Clean Streaming with Lambda Handlers

```
// Old way: Complex anonymous inner classes
public void oldStreamingApproach() {
    model.chat("Tell me a story", new StreamingChatResponseHandler() {
        @Override
        public void onPartialResponse(String token) {
            System.out.print(token);
        }

        @Override
        public void onError(Throwable error) {
            System.err.println("Error: " + error);
        }

        @Override
        public void onCompleted() {
            System.out.println("\nDone!");
        }
    });
}
```

Demo 7: Clean Streaming with Lambda Handlers

```java
// LangChain4j utilities: Clean lambda approach
import static dev.langchain4j.model.LambdaStreamingResponseHandler.*;

public class StreamingDemo {
    public static void main(String[] args) {
        var ollama = OllamaStreamingChatModel.builder()
                .baseUrl("http://localhost:11434")
                .modelName("gemma3")
                .build();

        // One-liner for simple streaming
        ollama.chat("Tell me a haiku about Java",
            onPartialResponse(System.out::print));
    }
}
```

Demo 7: Clean Streaming with Lambda Handlers

```
// With error handling - still clean and functional
var openai = OpenAiStreamingChatModel.builder()
        .apiKey(System.getenv("OPENAI_API_KEY"))
        .modelName("gpt-4o-mini")
        .build();

// Handle both tokens and errors elegantly
openai.chat("Why is the sky blue?",
    onPartialResponseAndError(
        System.out::print,  // Handle each token
        error -> System.err.println("Error: " + error.getMessage())
    ));
```

Demo 7: Clean Streaming with Lambda Handlers

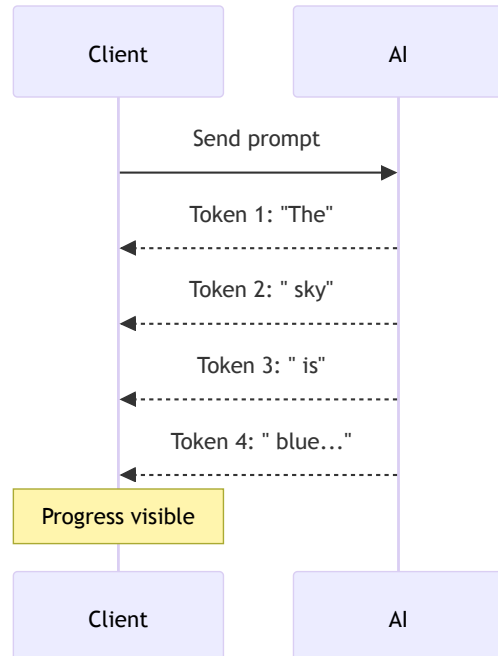
```
// With error handling - still clean and functional
var openai = OpenAiStreamingChatModel.builder()
        .apiKey(System.getenv("OPENAI_API_KEY"))
        .modelName("gpt-4o-mini")
        .build();

// Handle both tokens and errors elegantly
openai.chat("Why is the sky blue?",
    onPartialResponseAndError(
        System.out::print,  // Handle each token
        error -> System.err.println("Error: " + error.getMessage())
    ));
```

Live Demo: Run `StreamingDemo.java` to see tokens appear in real-time!

Streaming Architecture Benefits

User Experience

Technical Benefits

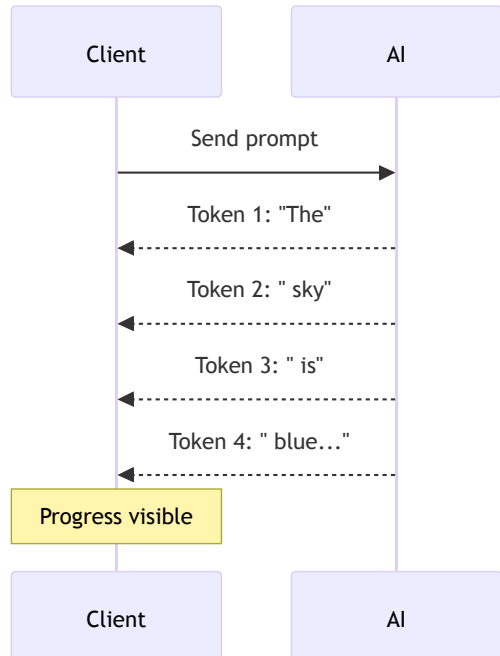


Streaming Architecture Benefits

User Experience

- **Immediate Feedback**: See progress instantly

Technical Benefits

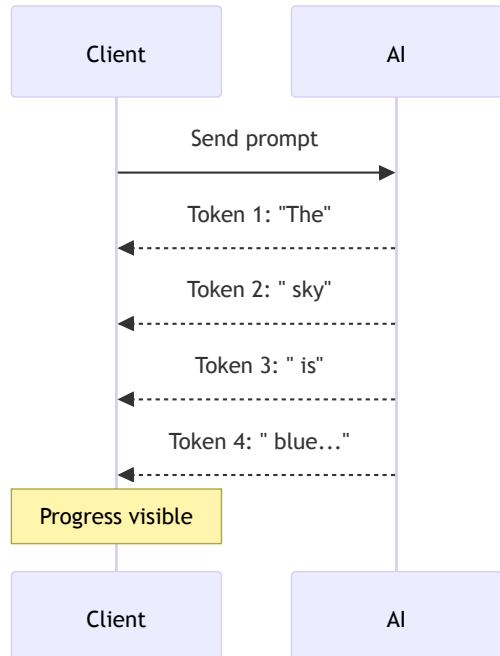


Streaming Architecture Benefits

User Experience

- **Immediate Feedback**: See progress instantly
- **Perceived Speed**: Feels faster

Technical Benefits

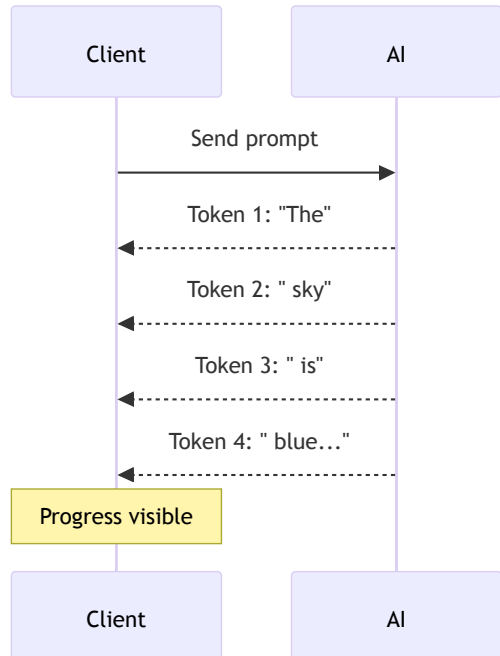


Streaming Architecture Benefits

User Experience

- **Immediate Feedback**: See progress instantly
- **Perceived Speed**: Feels faster
- **Interruptible**: Stop generation early

Technical Benefits

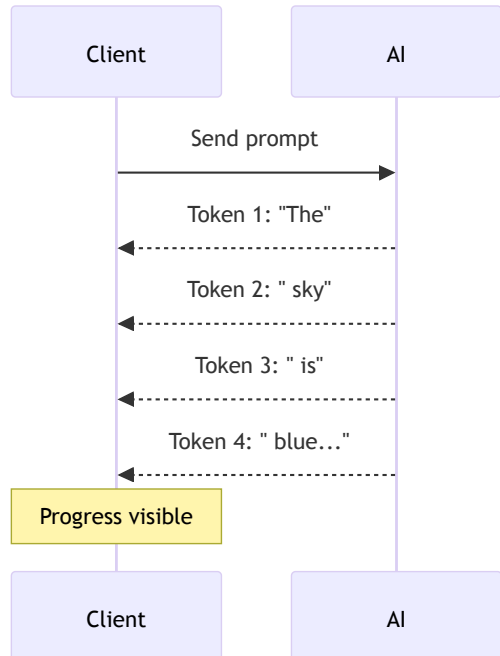


Streaming Architecture Benefits

User Experience

- **Immediate Feedback**: See progress instantly
- **Perceived Speed**: Feels faster
- **Interruptible**: Stop generation early
- **Engaging**: Watch AI "think"

Technical Benefits



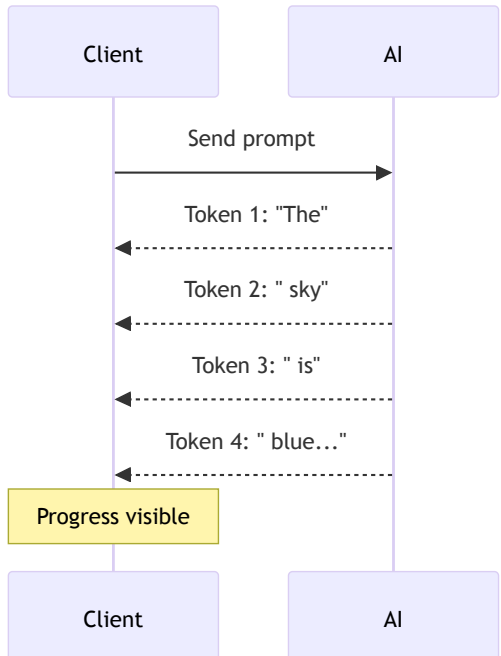
Streaming Architecture Benefits

User Experience

- **Immediate Feedback**: See progress instantly
- **Perceived Speed**: Feels faster
- **Interruptible**: Stop generation early
- **Engaging**: Watch AI "think"

Technical Benefits

- **Lower Latency**: First token quickly



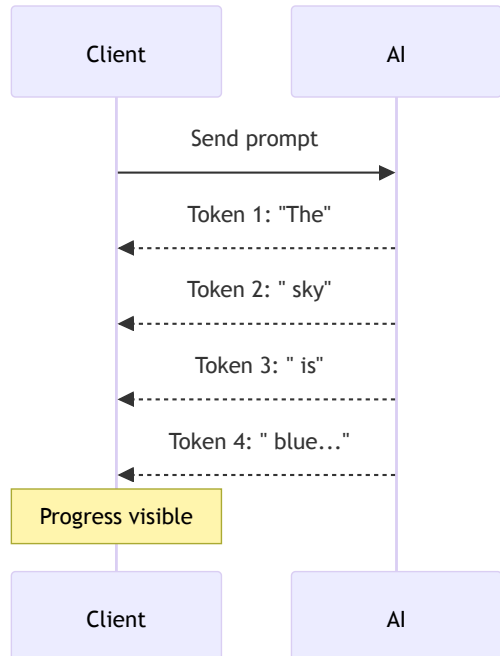
Streaming Architecture Benefits

User Experience

- **Immediate Feedback**: See progress instantly
- **Perceived Speed**: Feels faster
- **Interruptible**: Stop generation early
- **Engaging**: Watch AI "think"

Technical Benefits

- **Lower Latency**: First token quickly
- **Better Resources**: Process as arriving



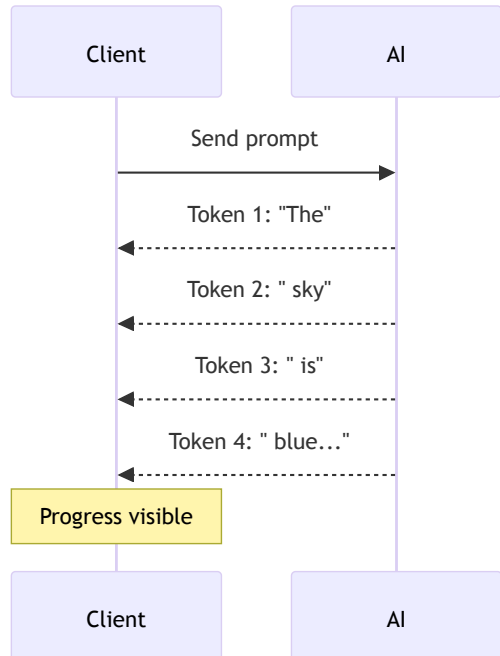
Streaming Architecture Benefits

User Experience

- **Immediate Feedback**: See progress instantly
- **Perceived Speed**: Feels faster
- **Interruptible**: Stop generation early
- **Engaging**: Watch AI "think"

Technical Benefits

- **Lower Latency**: First token quickly
- **Better Resources**: Process as arriving
- **Scalability**: Multiple streams



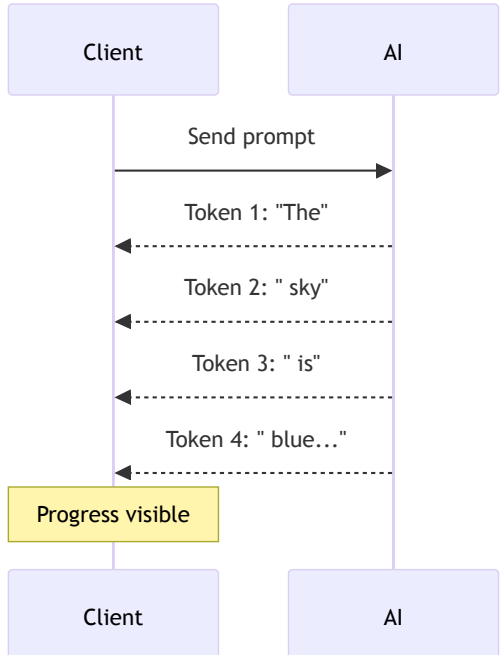
Streaming Architecture Benefits

User Experience

- **Immediate Feedback**: See progress instantly
- **Perceived Speed**: Feels faster
- **Interruptible**: Stop generation early
- **Engaging**: Watch AI "think"

Technical Benefits

- **Lower Latency**: First token quickly
- **Better Resources**: Process as arriving
- **Scalability**: Multiple streams
- **Error Recovery**: Graceful failures



Demo 8: Retrieval-Augmented Generation

Document-Powered AI

RAG: The Knowledge Problem

RAG: The Knowledge Problem

RAG: The Knowledge Problem

RAG: The Knowledge Problem

RAG: The Knowledge Problem

RAG: The Knowledge Problem

RAG Implementation with LangChain4j

```java
// Step 1: Load and process documents
import static dev.langchain4j.data.document.loader.FileSystemDocumentLoader.loadDocuments;

List<Document> documents = loadDocuments(
    toPath("documents/"),
    glob("*.txt")
);
```

RAG Implementation with LangChain4j

```
// Step 2: Create vector store and ingest documents
InMemoryEmbeddingStore<TextSegment> embeddingStore =
    new InMemoryEmbeddingStore<>();

// Magic happens here - documents become searchable vectors
EmbeddingStoreIngestor.ingest(documents, embeddingStore);

ContentRetriever retriever =
    EmbeddingStoreContentRetriever.from(embeddingStore);
```

RAG Implementation with LangChain4j

```java
// Step 3: Create AI assistant with document access
public interface Assistant {
    String chat(String userMessage);
}

Assistant assistant = AiServices.builder(Assistant.class)
        .chatModel(chatModel)
        .chatMemory(MessageWindowChatMemory.withMaxMessages(10))
        .contentRetriever(retriever)  // The magic ingredient!
         .build();
```

RAG Implementation with LangChain4j

```java
// Step 4: Complete EasyRAGDemo implementation
public class EasyRAGDemo {
    public static void main(String[] args) {
        List<Document> documents = loadDocuments(toPath("documents/"), glob("*.txt"));

        ChatModel chatModel = OpenAiChatModel.builder()
                .apiKey(System.getenv("OPENAI_API_KEY"))
                .modelName("gpt-4.1-nano")
                .build();

        Assistant assistant = AiServices.builder(Assistant.class)
                .chatModel(chatModel)
                .chatMemory(MessageWindowChatMemory.withMaxMessages(10))
                .contentRetriever(createContentRetriever(documents))
                .build();

        // AI can now answer questions about your documents!
        startConversationWith(assistant);
    }
}
```

RAG Implementation with LangChain4j

```java
// Step 4: Complete EasyRAGDemo implementation
public class EasyRAGDemo {
    public static void main(String[] args) {
        List<Document> documents = loadDocuments(toPath("documents/"), glob("*.txt"));

        ChatModel chatModel = OpenAiChatModel.builder()
                .apiKey(System.getenv("OPENAI_API_KEY"))
                .modelName("gpt-4.1-nano")
                .build();

        Assistant assistant = AiServices.builder(Assistant.class)
                .chatModel(chatModel)
                .chatMemory(MessageWindowChatMemory.withMaxMessages(10))
                .contentRetriever(createContentRetriever(documents))
                .build();

        // AI can now answer questions about your documents!
        startConversationWith(assistant);
    }
}
```

Live Demo: Run `EasyRAGDemo.java` - AI answers questions about YOUR documents!

RAG Deep Dive: How It Works

Vector Embeddings

Document Processing

RAG Deep Dive: How It Works

Vector Embeddings

- **Numerical Representation**: Text → vectors (1536+ dimensions)

Document Processing

RAG Deep Dive: How It Works

Vector Embeddings

- **Numerical Representation**: Text → vectors (1536+ dimensions)
- **Semantic Similarity**: Similar meaning = closer vectors

Document Processing

RAG Deep Dive: How It Works

Vector Embeddings

- **Numerical Representation**: Text → vectors (1536+ dimensions)
- **Semantic Similarity**: Similar meaning = closer vectors
- **Searchable**: Find relevant content by vector distance

Document Processing

RAG Deep Dive: How It Works

Vector Embeddings

- **Numerical Representation**: Text → vectors (1536+ dimensions)
- **Semantic Similarity**: Similar meaning = closer vectors
- **Searchable**: Find relevant content by vector distance

Document Processing

- **Chunking**: Split documents into manageable pieces

RAG Deep Dive: How It Works

Vector Embeddings

- **Numerical Representation**: Text → vectors (1536+ dimensions)
- **Semantic Similarity**: Similar meaning = closer vectors
- **Searchable**: Find relevant content by vector distance

Document Processing

- **Chunking**: Split documents into manageable pieces
- **Embedding**: Convert chunks to vectors

RAG Deep Dive: How It Works

Vector Embeddings

- **Numerical Representation**: Text → vectors (1536+ dimensions)
- **Semantic Similarity**: Similar meaning = closer vectors
- **Searchable**: Find relevant content by vector distance

Document Processing

- **Chunking**: Split documents into manageable pieces
- **Embedding**: Convert chunks to vectors
- **Storage**: Index in vector database

RAG Deep Dive: How It Works

Vector Embeddings

- **Numerical Representation**: Text → vectors (1536+ dimensions)
- **Semantic Similarity**: Similar meaning = closer vectors
- **Searchable**: Find relevant content by vector distance

Document Processing

- **Chunking**: Split documents into manageable pieces
- **Embedding**: Convert chunks to vectors
- **Storage**: Index in vector database
- **Retrieval**: Find most relevant chunks for query

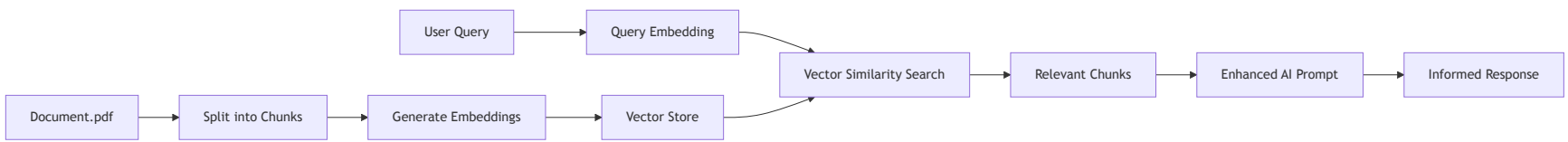
RAG Deep Dive: How It Works

Vector Embeddings

- **Numerical Representation**: Text → vectors (1536+ dimensions)
- **Semantic Similarity**: Similar meaning = closer vectors
- **Searchable**: Find relevant content by vector distance

Document Processing

- **Chunking**: Split documents into manageable pieces
- **Embedding**: Convert chunks to vectors
- **Storage**: Index in vector database
- **Retrieval**: Find most relevant chunks for query



Advanced Topics Overview

Production Considerations

Integration Patterns

Advanced Topics Overview

Production Considerations

- **Vector Databases**: Redis, Pinecone, Weaviate

Integration Patterns

Advanced Topics Overview

Production Considerations

- **Vector Databases**: Redis, Pinecone, Weaviate
- **Chunking Strategies**: Token-based, semantic, overlapping

Integration Patterns

Advanced Topics Overview

Production Considerations

- **Vector Databases**: Redis, Pinecone, Weaviate
- **Chunking Strategies**: Token-based, semantic, overlapping
- **Embedding Models**: OpenAI, sentence-transformers

Integration Patterns

Advanced Topics Overview

Production Considerations

- **Vector Databases**: Redis, Pinecone, Weaviate
- **Chunking Strategies**: Token-based, semantic, overlapping
- **Embedding Models**: OpenAI, sentence-transformers
- **Hybrid Search**: Vector + keyword search

Integration Patterns

Advanced Topics Overview

Production Considerations

- **Vector Databases**: Redis, Pinecone, Weaviate
- **Chunking Strategies**: Token-based, semantic, overlapping
- **Embedding Models**: OpenAI, sentence-transformers
- **Hybrid Search**: Vector + keyword search

Integration Patterns

- **Memory Management**: Conversation history

Advanced Topics Overview

Production Considerations

- **Vector Databases**: Redis, Pinecone, Weaviate
- **Chunking Strategies**: Token-based, semantic, overlapping
- **Embedding Models**: OpenAI, sentence-transformers
- **Hybrid Search**: Vector + keyword search

Integration Patterns

- **Memory Management**: Conversation history
- **Multi-Modal RAG**: Text, images, audio

Advanced Topics Overview

Production Considerations

- **Vector Databases**: Redis, Pinecone, Weaviate
- **Chunking Strategies**: Token-based, semantic, overlapping
- **Embedding Models**: OpenAI, sentence-transformers
- **Hybrid Search**: Vector + keyword search

Integration Patterns

- **Memory Management**: Conversation history
- **Multi-Modal RAG**: Text, images, audio
- **Real-Time Updates**: Dynamic document ingestion

Advanced Topics Overview

Production Considerations

- **Vector Databases**: Redis, Pinecone, Weaviate
- **Chunking Strategies**: Token-based, semantic, overlapping
- **Embedding Models**: OpenAI, sentence-transformers
- **Hybrid Search**: Vector + keyword search

Integration Patterns

- **Memory Management**: Conversation history
- **Multi-Modal RAG**: Text, images, audio
- **Real-Time Updates**: Dynamic document ingestion
- **Security**: Access control, data privacy

Testing Strategy

Smart Cost Control and Quality Assurance

Test Categories for Cost Control

Test Tags

```java
@Test
@Tag("local")  // Free
void testOllama() {
    // $0 cost
}

@Test
@Tag("cheap")  // Low cost
void testNano() {
    // ~$0.001
}

@Test
@Tag("demo")   // Fast
void quickDemo() {
    // Optimized
}
```

Gradle Commands

```bash
# Free tests only
./gradlew testLocal

# Low-cost tests
./gradlew testCheap

# Demo tests
./gradlew testDemo

# Exclude expensive
./gradlew testNotExpensive
```

Test Categories for Cost Control

Test Tags

```java
@Test
@Tag("local")  // Free
void testOllama() {
    // $0 cost
}

@Test
@Tag("cheap")  // Low cost
void testNano() {
    // ~$0.001
}

@Test
@Tag("demo")   // Fast
void quickDemo() {
    // Optimized
}
```

Gradle Commands

```bash
# Free tests only
./gradlew testLocal

# Low-cost tests
./gradlew testCheap

# Demo tests
./gradlew testDemo

# Exclude expensive
./gradlew testNotExpensive
```

Strategy: Develop with `testLocal`, validate with `testCheap`

Gradle Test Tasks

Cost-Controlled Testing

```
# Free tests only
./gradlew testLocal

# Low-cost tests
./gradlew testCheap

# Exclude expensive tests
./gradlew testNotExpensive

# Quick live demos
./gradlew testDemo
```

Provider-Specific Testing

```
# OpenAI API tests only
./gradlew testOpenAI

# All tests (including expensive)
./gradlew test

# Custom test selection
./gradlew test --tests "*OpenAi*"
```

Gradle Test Tasks

Cost-Controlled Testing

```
# Free tests only
./gradlew testLocal

# Low-cost tests
./gradlew testCheap

# Exclude expensive tests
./gradlew testNotExpensive

# Quick live demos
./gradlew testDemo
```

Provider-Specific Testing

```
# OpenAI API tests only
./gradlew testOpenAI

# All tests (including expensive)
./gradlew test

# Custom test selection
./gradlew test --tests "*OpenAi*"
```

Smart Development: Use free local models for TDD, validate with cloud models before deployment

Testing Patterns for AI Applications

Unit Testing Approaches

```java
@Test
void shouldParseModelList() {
    String json = """
        {"data": [
            {"id": "gpt-4", "created": 123}
        ]}
        """;
    ModelList result = gson.fromJson(json, ModelList
    assertThat(result.data()).hasSize(1);
}

@Test
void shouldHandleApiError() {
    // Test error scenarios
    assertThrows(RuntimeException.class,
        () -> service.generateWithInvalidKey());
}
```

Integration Testing

```java
@Test
@Tag("local")
void shouldStreamTokens() {
    List<String> tokens = new ArrayList<>();

    ollamaService.generateStreaming("gemma3",
        "Count to 5",
        token -> tokens.add(token)
    );

    assertThat(tokens).isNotEmpty();
    assertThat(String.join("", tokens))
        .containsPattern("1.*2.*3.*4.*5");
}
```

Quality Assurance for Non-Deterministic Systems

```java
@Test
@Tag("cheap")
void shouldAnswerSkyColorQuestion() {
    String response = chatModel.chat("Why is the sky blue?");

    // Flexible content assertion
    assertThat(response.toLowerCase())
        .containsAnyOf("scattering", "wavelength", "atmosphere", "light");

    // Structural assertion
    assertThat(response).hasSizeGreaterThan(10);
    assertThat(response).doesNotContain("I don't know");
}
```

Quality Assurance for Non-Deterministic Systems

- **Content Assertions**: Check for key concepts, not exact strings

```java
@Test
@Tag("cheap")
void shouldAnswerSkyColorQuestion() {
    String response = chatModel.chat("Why is the sky blue?");

    // Flexible content assertion
    assertThat(response.toLowerCase())
        .containsAnyOf("scattering", "wavelength", "atmosphere", "light");

    // Structural assertion
    assertThat(response).hasSizeGreaterThan(10);
    assertThat(response).doesNotContain("I don't know");
}
```

Quality Assurance for Non-Deterministic Systems

- **Content Assertions**: Check for key concepts, not exact strings
- **Structural Validation**: Verify JSON format, required fields

```java
@Test
@Tag("cheap")
void shouldAnswerSkyColorQuestion() {
    String response = chatModel.chat("Why is the sky blue?");

    // Flexible content assertion
    assertThat(response.toLowerCase())
        .containsAnyOf("scattering", "wavelength", "atmosphere", "light");

    // Structural assertion
    assertThat(response).hasSizeGreaterThan(10);
    assertThat(response).doesNotContain("I don't know");
}
```

Quality Assurance for Non-Deterministic Systems

- **Content Assertions**: Check for key concepts, not exact strings
- **Structural Validation**: Verify JSON format, required fields
- **Error Handling**: Test timeout, rate limits, invalid input

```java
@Test
@Tag("cheap")
void shouldAnswerSkyColorQuestion() {
    String response = chatModel.chat("Why is the sky blue?");

    // Flexible content assertion
    assertThat(response.toLowerCase())
        .containsAnyOf("scattering", "wavelength", "atmosphere", "light");

    // Structural assertion
    assertThat(response).hasSizeGreaterThan(10);
    assertThat(response).doesNotContain("I don't know");
}
```

Quality Assurance for Non-Deterministic Systems

- **Content Assertions**: Check for key concepts, not exact strings
- **Structural Validation**: Verify JSON format, required fields
- **Error Handling**: Test timeout, rate limits, invalid input
- **Performance Bounds**: Response time, token limits

```java
@Test
@Tag("cheap")
void shouldAnswerSkyColorQuestion() {
    String response = chatModel.chat("Why is the sky blue?");

    // Flexible content assertion
    assertThat(response.toLowerCase())
        .containsAnyOf("scattering", "wavelength", "atmosphere", "light");

    // Structural assertion
    assertThat(response).hasSizeGreaterThan(10);
    assertThat(response).doesNotContain("I don't know");
}
```

Quality Assurance for Non-Deterministic Systems

- **Content Assertions**: Check for key concepts, not exact strings
- **Structural Validation**: Verify JSON format, required fields
- **Error Handling**: Test timeout, rate limits, invalid input
- **Performance Bounds**: Response time, token limits
- **Cost Monitoring**: Track API usage in tests

```java
@Test
@Tag("cheap")
void shouldAnswerSkyColorQuestion() {
    String response = chatModel.chat("Why is the sky blue?");

    // Flexible content assertion
    assertThat(response.toLowerCase())
        .containsAnyOf("scattering", "wavelength", "atmosphere", "light");

    // Structural assertion
    assertThat(response).hasSizeGreaterThan(10);
    assertThat(response).doesNotContain("I don't know");
}
```

Course Summary

From HTTP to Intelligent Applications

Your AI Java Journey

Foundation Built

Advanced Capabilities

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns

Advanced Capabilities

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns
- **Modern Java**: Records, sealed interfaces, pattern matching

Advanced Capabilities

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns
- **Modern Java**: Records, sealed interfaces, pattern matching
- **Framework Mastery**: LangChain4j abstractions

Advanced Capabilities

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns
- **Modern Java**: Records, sealed interfaces, pattern matching
- **Framework Mastery**: LangChain4j abstractions
- **Cost Management**: Smart testing strategies

Advanced Capabilities

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns
- **Modern Java**: Records, sealed interfaces, pattern matching
- **Framework Mastery**: LangChain4j abstractions
- **Cost Management**: Smart testing strategies

Advanced Capabilities

- **Multi-Modal AI**: Text, vision, audio integration

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns
- **Modern Java**: Records, sealed interfaces, pattern matching
- **Framework Mastery**: LangChain4j abstractions
- **Cost Management**: Smart testing strategies

Advanced Capabilities

- **Multi-Modal AI**: Text, vision, audio integration
- **Streaming Responses**: Real-time user experiences

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns
- **Modern Java**: Records, sealed interfaces, pattern matching
- **Framework Mastery**: LangChain4j abstractions
- **Cost Management**: Smart testing strategies

Advanced Capabilities

- **Multi-Modal AI**: Text, vision, audio integration
- **Streaming Responses**: Real-time user experiences
- **RAG Systems**: Document-powered AI

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns
- **Modern Java**: Records, sealed interfaces, pattern matching
- **Framework Mastery**: LangChain4j abstractions
- **Cost Management**: Smart testing strategies

Advanced Capabilities

- **Multi-Modal AI**: Text, vision, audio integration
- **Streaming Responses**: Real-time user experiences
- **RAG Systems**: Document-powered AI
- **Production Patterns**: Error handling, monitoring

Your AI Java Journey

Foundation Built

- **HTTP Fundamentals**: Raw API integration patterns
- **Modern Java**: Records, sealed interfaces, pattern matching
- **Framework Mastery**: LangChain4j abstractions
- **Cost Management**: Smart testing strategies

Advanced Capabilities

- **Multi-Modal AI**: Text, vision, audio integration
- **Streaming Responses**: Real-time user experiences
- **RAG Systems**: Document-powered AI
- **Production Patterns**: Error handling, monitoring

Result: You can build intelligent Java applications with confidence! 🚀

Demo Classes Mastered

Core Demos

- **QuickChatDemo** - Fast OpenAI integration
- **TextToSpeechDemo** - Audio generation
- **LocalOllamaDemo** - Privacy-first AI
- **MultiModelDemo** - Provider comparison

Advanced Demos

- **StreamingDemo** - Real-time responses
- **ResponsesApiDemo** - Raw HTTP patterns
- **ApiComparisonDemo** - Framework vs raw
- **EasyRAGDemo** - Document Q&A

Demo Classes Mastered

Core Demos

- **QuickChatDemo** - Fast OpenAI integration
- **TextToSpeechDemo** - Audio generation
- **LocalOllamaDemo** - Privacy-first AI
- **MultiModelDemo** - Provider comparison

Advanced Demos

- **StreamingDemo** - Real-time responses
- **ResponsesApiDemo** - Raw HTTP patterns
- **ApiComparisonDemo** - Framework vs raw
- **EasyRAGDemo** - Document Q&A

All Ready for Live Demonstration: Each demo is optimized for training delivery

Key Takeaways

Key Takeaways

1. **Understand Both Levels**: Raw HTTP gives you debugging superpowers, frameworks give you productivity

Key Takeaways

1. **Understand Both Levels**: Raw HTTP gives you debugging superpowers, frameworks give you productivity
2. **Cost Management Matters**: Use test categories and local models for development

Key Takeaways

1. **Understand Both Levels**: Raw HTTP gives you debugging superpowers, frameworks give you productivity
2. **Cost Management Matters**: Use test categories and local models for development
3. **Modern Java Shines**: Records, sealed interfaces, and pattern matching are perfect for AI APIs

Key Takeaways

1. **Understand Both Levels**: Raw HTTP gives you debugging superpowers, frameworks give you productivity
2. **Cost Management Matters**: Use test categories and local models for development
3. **Modern Java Shines**: Records, sealed interfaces, and pattern matching are perfect for AI APIs
4. **LangChain4j is Powerful**: Unified API across all providers enables true portability

Key Takeaways

1. **Understand Both Levels**: Raw HTTP gives you debugging superpowers, frameworks give you productivity
2. **Cost Management Matters**: Use test categories and local models for development
3. **Modern Java Shines**: Records, sealed interfaces, and pattern matching are perfect for AI APIs
4. **LangChain4j is Powerful**: Unified API across all providers enables true portability
5. **RAG Changes Everything**: Give AI access to your documents and data

Key Takeaways

1. **Understand Both Levels**: Raw HTTP gives you debugging superpowers, frameworks give you productivity
2. **Cost Management Matters**: Use test categories and local models for development
3. **Modern Java Shines**: Records, sealed interfaces, and pattern matching are perfect for AI APIs
4. **LangChain4j is Powerful**: Unified API across all providers enables true portability
5. **RAG Changes Everything**: Give AI access to your documents and data
6. **Streaming Improves UX**: Real-time responses feel much faster to users

Key Takeaways

1. **Understand Both Levels**: Raw HTTP gives you debugging superpowers, frameworks give you productivity
2. **Cost Management Matters**: Use test categories and local models for development
3. **Modern Java Shines**: Records, sealed interfaces, and pattern matching are perfect for AI APIs
4. **LangChain4j is Powerful**: Unified API across all providers enables true portability
5. **RAG Changes Everything**: Give AI access to your documents and data
6. **Streaming Improves UX**: Real-time responses feel much faster to users
7. **Test Smart, Not Hard**: Use local models for TDD, cloud models for validation

Next Steps: Continue Your AI Journey

Hands-On Practice

Advanced Topics

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes

Advanced Topics

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes
- **Complete the labs**: 15 progressive exercises

Advanced Topics

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes
- **Complete the labs**: 15 progressive exercises
- **Build custom integrations**: Add your own AI providers

Advanced Topics

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes
- **Complete the labs**: 15 progressive exercises
- **Build custom integrations**: Add your own AI providers
- **Production deployment**: Scale with real applications

Advanced Topics

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes
- **Complete the labs**: 15 progressive exercises
- **Build custom integrations**: Add your own AI providers
- **Production deployment**: Scale with real applications

Advanced Topics

- **Vector databases**: Redis, Pinecone, Weaviate

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes
- **Complete the labs**: 15 progressive exercises
- **Build custom integrations**: Add your own AI providers
- **Production deployment**: Scale with real applications

Advanced Topics

- **Vector databases**: Redis, Pinecone, Weaviate
- **Multi-agent systems**: Coordinated AI workflows

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes
- **Complete the labs**: 15 progressive exercises
- **Build custom integrations**: Add your own AI providers
- **Production deployment**: Scale with real applications

Advanced Topics

- **Vector databases**: Redis, Pinecone, Weaviate
- **Multi-agent systems**: Coordinated AI workflows
- **Function calling**: Give AI access to your APIs

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes
- **Complete the labs**: 15 progressive exercises
- **Build custom integrations**: Add your own AI providers
- **Production deployment**: Scale with real applications

Advanced Topics

- **Vector databases**: Redis, Pinecone, Weaviate
- **Multi-agent systems**: Coordinated AI workflows
- **Function calling**: Give AI access to your APIs
- **Custom embeddings**: Domain-specific models

Next Steps: Continue Your AI Journey

Hands-On Practice

- **Explore the repository**: Try all 8 demo classes
- **Complete the labs**: 15 progressive exercises
- **Build custom integrations**: Add your own AI providers
- **Production deployment**: Scale with real applications

Resources:

- LangChain4j Documentation
- Course Repository
- Ollama Models

Advanced Topics

- **Vector databases**: Redis, Pinecone, Weaviate
- **Multi-agent systems**: Coordinated AI workflows
- **Function calling**: Give AI access to your APIs
- **Custom embeddings**: Domain-specific models

Production Considerations

Operational Excellence

Performance Optimization

Production Considerations

Operational Excellence

- **API Cost Monitoring**: Track usage across models

Performance Optimization

Production Considerations

Operational Excellence

- **API Cost Monitoring**: Track usage across models
- **Rate Limiting**: Handle quotas gracefully

Performance Optimization

Production Considerations

Operational Excellence

- **API Cost Monitoring**: Track usage across models
- **Rate Limiting**: Handle quotas gracefully
- **Error Recovery**: Circuit breakers, retries

Performance Optimization

Production Considerations

Operational Excellence

- **API Cost Monitoring**: Track usage across models
- **Rate Limiting**: Handle quotas gracefully
- **Error Recovery**: Circuit breakers, retries
- **Security**: API key management, input validation

Performance Optimization

Production Considerations

Operational Excellence

- **API Cost Monitoring**: Track usage across models
- **Rate Limiting**: Handle quotas gracefully
- **Error Recovery**: Circuit breakers, retries
- **Security**: API key management, input validation

Performance Optimization

- **Model Selection**: Right model for the task

Production Considerations

Operational Excellence

- **API Cost Monitoring**: Track usage across models
- **Rate Limiting**: Handle quotas gracefully
- **Error Recovery**: Circuit breakers, retries
- **Security**: API key management, input validation

Performance Optimization

- **Model Selection**: Right model for the task
- **Caching Strategies**: Cache embeddings and responses

Production Considerations

Operational Excellence

- **API Cost Monitoring**: Track usage across models
- **Rate Limiting**: Handle quotas gracefully
- **Error Recovery**: Circuit breakers, retries
- **Security**: API key management, input validation

Performance Optimization

- **Model Selection**: Right model for the task
- **Caching Strategies**: Cache embeddings and responses
- **Batch Processing**: Optimize API usage

Production Considerations

Operational Excellence

- **API Cost Monitoring**: Track usage across models
- **Rate Limiting**: Handle quotas gracefully
- **Error Recovery**: Circuit breakers, retries
- **Security**: API key management, input validation

Performance Optimization

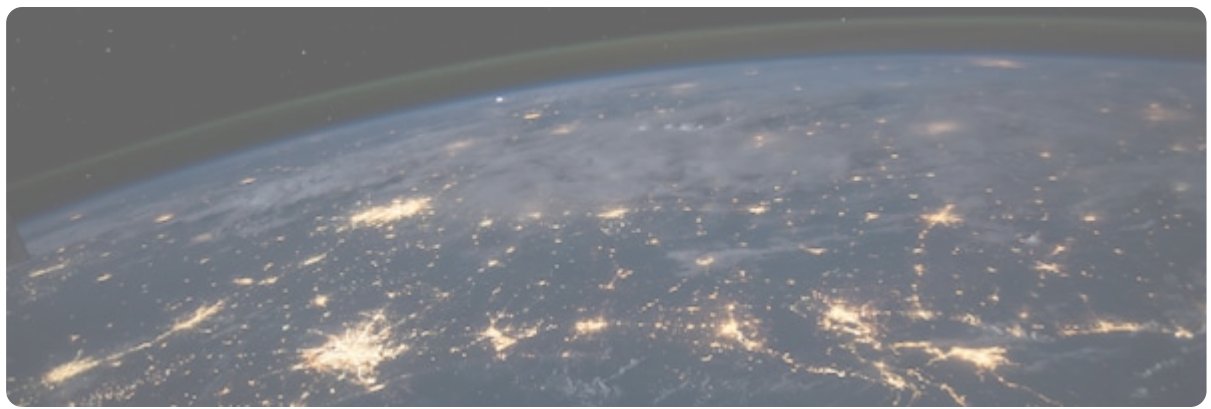
- **Model Selection**: Right model for the task
- **Caching Strategies**: Cache embeddings and responses
- **Batch Processing**: Optimize API usage
- **Local Models**: Privacy and cost benefits

Thank You!

Questions?

Kenneth Kousen
Java Champion, Author, Speaker

kousenit.com | @kenkousen



Ready to build intelligent Java applications!