Spring AI: Building Intelligent Applications

From basic chat to advanced RAG and MCP →

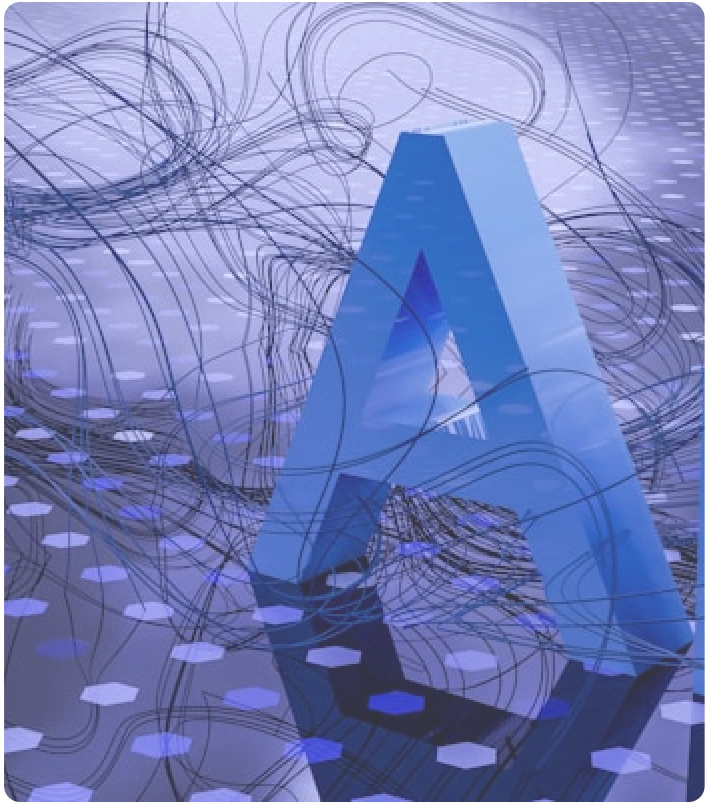
Contact Info

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

- **Basic AI Integration**: ChatClient fundamentals
- **Streaming Responses**: Real-time AI interactions
- **Structured Data**: AI-powered object extraction
- **Multimodal AI**: Vision and audio capabilities
- **Function Calling**: Extend AI with custom tools
- **RAG Systems**: Knowledge-augmented AI
- **MCP Protocol**: Model Context Protocol implementation
- **Production Patterns**: Enterprise-ready architectures



Repository Structure

```
Spring_AI_Training_Course/
├── labs.md             # 15 progressive lab exercises
├── src/
│   ├── main/java/      # Service implementations
│   ├── main/resources/ # Configuration & templates
│   └── test/java/      # Test-driven exercises
├── README.md           # Course documentation
└── slides.md           # This presentation
```

- **Start**: `main` branch with guided TODOs
- **Reference**: `solutions` branch when needed
- **Learn by doing**: Implement each lab incrementally
- **15 Labs**: From basic chat to advanced MCP servers

Spring AI Ecosystem

AI Providers

- OpenAI (GPT, DALL-E)
- Anthropic (Claude)
- Azure OpenAI
- Google Vertex AI
- Local models (Ollama)

Vector Stores

- SimpleVectorStore (in-memory)
- Redis Vector Store
- Pinecone, Weaviate
- PgVector, Chroma

Capabilities

- Text generation
- Image analysis/generation
- Speech-to-text/text-to-speech
- Function calling
- RAG workflows

Prerequisites

Technical Requirements

- **Java 17+**
- **Spring Boot 3.5.8**
- **Spring AI 1.1.0**
- **Git** for branch management
- **Redis** (optional, for advanced RAG)

Environment Setup

```
# Required API keys
export OPENAI_API_KEY=your_key
export ANTHROPIC_API_KEY=your_key

# Optional: Redis for advanced labs
docker run -p 6379:6379 redis/redis-stack:latest

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

Note: Using `gpt-5-nano` and `claude-opus-4-1`

Lab 1-3: Foundations

Building Your First AI-Powered Spring Application

Lab 1: Basic Chat Interactions

```java
// Step 2: Complete implementation
@Service
public class ChatService {
    private final ChatClient chatClient;

    public ChatService(ChatClient.Builder builder) {
        this.chatClient = builder.build();
    }

    public String generateResponse(String prompt) {
        return chatClient.prompt(prompt)
            .call()
            .content();
    }
}
```

Result: AI-powered responses in your Spring application! 🎉

Lab 2: Request/Response Logging

```java
@Service
public class ChatService {
    private final ChatClient chatClient;

    public ChatService(ChatClient.Builder builder) {

        // Add logging advisor for debugging
        this.chatClient = builder
            .defaultAdvisors(new SimpleLoggerAdvisor())
            .build();
    }

    public String generateResponse(String prompt) {
        return chatClient.prompt(prompt)
            .call()
            .content();
    }
}
```

Debug Output: See exactly what’s sent to and received from AI models

Lab 3: Streaming Responses

```java
@RestController
public class ChatController {

    @GetMapping(value = "/chat/stream", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
    public Flux<String> streamChat(@RequestParam String message) {
        return chatClient.prompt(message)
            .stream()
            .content();
    }

    // Frontend receives real-time token-by-token responses
    // Perfect for chat interfaces and long AI responses
    // Uses Spring WebFlux Reactor streams
}
```

Experience: Real-time AI responses like ChatGPT interface

Lab 4-6: Structured AI

From Text to Objects

Lab 4: Structured Data Extraction

```java
// Define your data structure
public record ActorFilms(String actor, List<String> movies) {}

@Test
void shouldGetActorFilms() {
    // AI converts natural language to structured data
    ActorFilms actorFilms = chatClient.prompt("Generate the filmography for Tom Hanks")
        .call()
        .entity(ActorFilms.class);

    // Assert AI returned proper structure
    assertThat(actorFilms.actor()).isEqualTo("Tom Hanks");
    assertThat(actorFilms.movies()).contains("Forrest Gump", "Cast Away");
}
```

Magic: AI understands your Java objects and populates them correctly!

Lab 5: Prompt Templates

```java
@Component
public class TemplateService {

    @Value("classpath:/prompts/actor-filmography.st")
    private Resource actorFilmographyTemplate;

    public ActorFilms getActorFilms(String actorName) {
        return chatClient.prompt()
            .user(userSpec -> userSpec
                .text(actorFilmographyTemplate)
                .param("actor", actorName)
                .param("count", 5))
            .call()
            .entity(ActorFilms.class);
    }
}
```

Template File (`actor-filmography.st`):

```
Generate a filmography for {actor}.
Include exactly {count} of their most famous movies.
Format as JSON with actor name and movies array.
```

Lab 6: Chat Memory

```java
@Service
public class ConversationService {
    private final ChatClient chatClient;

    public ConversationService(ChatClient.Builder builder) {
        this.chatClient = builder
            .defaultAdvisors(new MessageChatMemoryAdvisor(
                new InMemoryChatMemory())) // Remembers conversation
            .build();
    }

    public String continueConversation(String message) {
        return chatClient.prompt(message)
            .call()
            .content();
        // AI remembers previous messages in this conversation!
    }
}
```

Try this:

1. "My name is John"

Lab 7-9: Multimodal AI

Beyond Text: Vision and Audio

Lab 7: Vision Capabilities

```java
@Test
void shouldAnalyzeImage() {
    var imageResource = new ClassPathResource(
        "/images/multimodal_test_image.png");

    String response = chatClient.prompt()
        .user(userSpec -> userSpec
            .text("What do you see in this image?")
            .media(MimeTypeUtils.IMAGE_PNG, imageResource))
        .call()
        .content();

    assertThat(response.toLowerCase())
        .contains("dog", "playing");
}
```

Vision: What AI Can Analyze

- **Photos and diagrams**
- **Charts and graphs**
- **Screenshots and UI mockups**
- **Medical images**
- **Technical drawings**

Lab 8: Image Generation

```java
@Service
public class ImageService {
    private final ImageModel imageModel;

    public ImageService(ImageModel imageModel) {
        this.imageModel = imageModel;
    }

    public String generateImage(String prompt) {
        ImageResponse response = imageModel.call(
            new ImagePrompt(prompt,
                ImageOptionsBuilder.builder()
                    .withModel("dall-e-3")
                    .withHeight(1024)
                    .withWidth(1024)
                    .build()));

        return response.getResult()
            .getOutput()
            .getUrl();
    }
}
```

Create: AI-generated images from text descriptions

Lab 9: AI Tools (Function Calling)

```
// Step 3: AI automatically calls your tools
@Test
void shouldCallTool() {
    String response = chatClient.prompt(
        "What time is it right now?")
        .call()
        .content();

    // AI called getCurrentDateTime() automatically!
    assertThat(response).contains("2024");
}
```

Result: AI can execute your Java methods when needed!

Lab 10-11: Audio Processing

Speech-to-Text and Text-to-Speech

Lab 10: Audio Transcription

```java
@Service
public class AudioService {
    private final AudioTranscriptionModel transcriptionModel;

    public AudioService(AudioTranscriptionModel model) {
        this.transcriptionModel = model;
    }

    public String transcribeAudio(Resource audioFile) {
        AudioTranscriptionPrompt prompt =
            new AudioTranscriptionPrompt(audioFile);

        AudioTranscriptionResponse response =
            transcriptionModel.call(prompt);

        return response.getResult().getOutput();
    }
}
```

Capability: Convert speech files (MP3, WAV) to accurate text transcription

Lab 11: Text-to-Speech

```java
@Service
public class SpeechService {
    private final AudioSpeechModel speechModel;

    public SpeechService(AudioSpeechModel speechModel) {
        this.speechModel = speechModel;
    }

    public byte[] generateSpeech(String text) {
        AudioSpeechPrompt prompt = new AudioSpeechPrompt(text,
            AudioSpeechOptionsBuilder.builder()
                .withModel("tts-1")
                .withVoice(AudioSpeechOptions.Voice.ALLOY)
                .build());

        return speechModel.call(prompt).getResult().getOutput();
    }
}
```

Text-to-Speech: Usage

```java
// Generate and save audio
byte[] audioData = speechService.generateSpeech(
    "Welcome to Spring AI training!");

// Save to file for playback
Files.write(Paths.get("welcome.mp3"), audioData);
```

Output: High-quality AI-generated speech from any text

Lab 12-13: RAG Systems

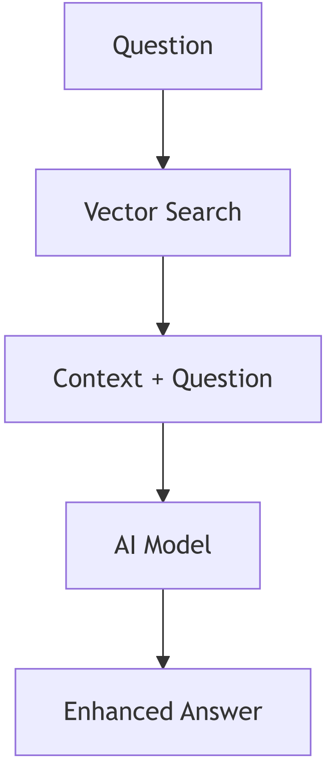
Knowledge-Augmented AI

Lab 12: The RAG Problem

Traditional AI Limitations

- Knowledge cutoff dates
- Can’t access your documents
- No real-time information
- Generic responses only

RAG Solution



RAG: How It Works

1. **Document Processing**: Split documents into chunks
2. **Embedding Generation**: Convert chunks to vectors
3. **Vector Storage**: Store in searchable database
4. **Query Processing**: Find relevant chunks for question
5. **Context Enhancement**: Add found content to AI prompt
6. **Enhanced Response**: AI answers using your data

Result: AI answers using YOUR documents and data! 🎯

Spring AI: Supported Providers

Chat Models

- OpenAI • Azure OpenAI
- Anthropic • Google VertexAI
- Amazon Bedrock • Ollama
- Hugging Face • Mistral AI
- Groq • NVIDIA • Perplexity
- DeepSeek • Moonshot AI
- QianFan • ZhiPu AI • MiniMax

Embedding Models

- OpenAI • Azure OpenAI
- Amazon Bedrock • VertexAI
- Ollama • Mistral AI
- PostgresML • ONNX
- QianFan • ZhiPu AI
- OCI GenAI • MiniMax

Image & Audio

- **Images**: OpenAI DALL-E
- **Images**: Stability AI, ZhiPu AI
- **Speech-to-Text**: OpenAI Whisper
- **Text-to-Speech**: OpenAI TTS
- **Moderation**: OpenAI, Mistral

Spring AI advantage: Portable API - switch providers with configuration only!

Understanding Embeddings & Vector Search

What are Embeddings?

- **Numerical representation** of text meaning
- **High-dimensional vectors** (typically 1536+ dimensions)
- **Semantic similarity** via distance calculations
- **Context-aware** - same words, different meanings

Chunking Strategies

- **Token-based**: Split by token count (GPT tokenizer)
- **Sentence-based**: Preserve sentence boundaries
- **Semantic**: Split by topic/meaning changes
- **Overlapping**: Chunks share context at boundaries

RAG Implementation

```java
// Step 3: Testing RAG
@Test
void shouldAnswerFromDocuments() {
    String answer = ragService.askQuestion(
        "What is Spring AI ChatClient?");

    // AI uses loaded PDF content to answer!
    assertThat(answer).contains("ChatClient", "Spring AI");
}
```

Magic: AI answers questions using your PDF documents!

Lab 13: Production RAG with Redis

```java
@Configuration
@Profile("redis")
public class RedisRAGConfig {

    @Bean
    public VectorStore redisVectorStore(EmbeddingModel embeddingModel) {
        // Spring AI 1.1.0 requires JedisPooled as first parameter
        return RedisVectorStore.builder(new JedisPooled("localhost", 6379), embeddingModel)
                .indexName("spring-ai-index")
                .initializeSchema(true)
                .build();
    }

    @Bean
    public ApplicationRunner dataLoader(VectorStore vectorStore) {
        return args -> loadDocumentsIfEmpty(vectorStore);
    }
}
```

Redis RAG: Smart Data Loading

```
@Bean
public ApplicationRunner dataLoader(VectorStore vectorStore) {
    return args -> {
        if (vectorStore instanceof RedisVectorStore redis &&
            redis.getCollection().isEmpty()) {

            // Only load documents if Redis is empty
            loadDocuments(vectorStore);
            log.info("Loaded {} documents into Redis", count);
        }
    };
}
```

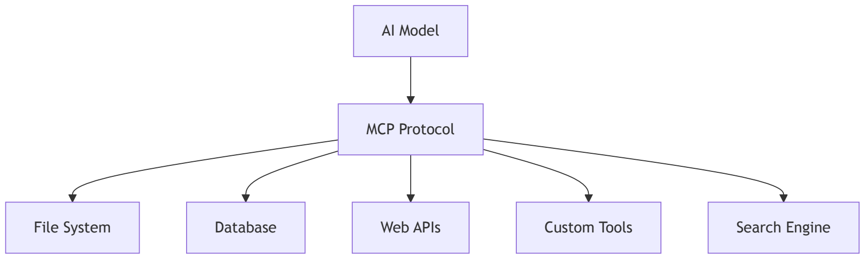
Benefits: Persistent • Scalable • Fast similarity search

Lab 14-15: Model Context Protocol

The Future of AI Tool Integration

What is MCP?

- **Standardized protocol** for AI tool communication
- **Open source** by Anthropic
- **Universal interface** between AI and tools
- **Secure sandbox** for AI operations
- **Growing ecosystem** of MCP servers



MCP enables AI to securely access external tools and data sources

Lab 14: MCP Client

```java
@Service
@Profile("mcp")
public class McpClientService {
    private final ChatClient chatClient;

    public McpClientService(ChatClient.Builder builder) {
        // Spring AI auto-discovers MCP servers from config
        this.chatClient = builder.build();
    }

    public String executeWithTools(String request) {
        return chatClient.prompt(request).call().content();
        // AI can now use filesystem, search, etc.
    }
}
```

MCP Client: Configuration

Configuration (`mcp-servers-config.json`):

```json
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "/tmp"]
    },
    "search": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-brave-search"]
    }
  }
}
```

Result: AI can access external tools through standardized protocol

Lab 15: MCP Server

```
# Step 3: Run MCP server
./gradlew bootRun --args='--spring.profiles.active=mcp-server'

# Step 4: Connect from Claude Desktop
# Add to Claude's MCP config:
{
  "mcpServers": {
    "spring-calculator": {
      "command": "java",
      "args": ["-jar", "app.jar", "--spring.profiles.active=mcp-server"]
    }
  }
}
```

Result: Claude Desktop can use your Java methods as tools!

Production Patterns

Enterprise-Ready Spring AI

Service Layer: REST Controller

```java
@RestController
@RequestMapping("/api/ai")
public class AIController {
    private final ChatService chatService;

    @PostMapping("/chat")
    public ResponseEntity<String> chat(@RequestBody ChatRequest request) {
        try {
            String response = chatService.generateResponse(request.getMessage());
            return ResponseEntity.ok(response);
        } catch (Exception e) {
            return ResponseEntity.status(500)
                .body("AI service temporarily unavailable");
        }
    }
```

Service Layer: Streaming Support

```java
@GetMapping("/chat/stream")
public ResponseEntity<Flux<String>> streamChat(@RequestParam String message) {
    Flux<String> stream = chatService.streamResponse(message)
        .onErrorReturn("Error occurred during streaming");

    return ResponseEntity.ok()
        .contentType(MediaType.TEXT_EVENT_STREAM)
        .body(stream);
}

// Clean separation of concerns
// Proper error handling
// Stream-ready architecture
// Production-ready patterns
```

Configuration Management

```java
@Configuration
public class AIConfiguration {

    @Bean
    @Primary
    @ConditionalOnProperty("spring.ai.openai.api-key")
    public ChatModel primaryChatModel(OpenAiChatModel openAiModel) {
        return openAiModel;
    }

    @Bean
    @ConditionalOnProfile("rag")
    public VectorStore vectorStore() {
        return new SimpleVectorStore();
    }
}
```

Profile-Based Feature Activation

```java
@Bean
@Profile("redis")
public VectorStore redisVectorStore(EmbeddingModel embeddingModel) {
    return RedisVectorStore.builder(new JedisPooled("localhost", 6379), embeddingModel)
            .indexName("spring-ai-index")
            .initializeSchema(true)
            .build();
}

@Bean
@Profile("mcp")
public McpClientConfiguration mcpConfig() {
    return new McpClientConfiguration();
}
```

Run configurations:

```bash
./gradlew bootRun                                    # Basic AI
./gradlew bootRun --args='--spring.profiles.active=rag,redis'  # RAG
./gradlew bootRun --args='--spring.profiles.active=mcp'        # MCP
```

Error Handling: Retries

```java
@Service
public class ResilientChatService {
    private final ChatClient chatClient;

    @Retryable(value = {ApiException.class}, maxAttempts = 3)
    public String generateResponse(String prompt) {
        return chatClient.prompt(prompt).call().content();
    }

    // Automatically retries API failures
    // Exponential backoff available
    // Works with Spring Retry
}
```

Error Handling: Circuit Breaker

```java
@CircuitBreaker(name = "ai-service", fallbackMethod = "fallbackResponse")
public String generateResponseWithCircuitBreaker(String prompt) {
    return chatClient.prompt(prompt).call().content();
}

public String fallbackResponse(String prompt, Exception ex) {
    log.warn("AI service failed, using fallback", ex);
    return "I'm temporarily unable to process your request.";
}

// Prevents cascade failures
// Automatic recovery when service improves
```

Error Handling: Async Processing

```java
@Async
public CompletableFuture<String> generateResponseAsync(String prompt) {
    return CompletableFuture.supplyAsync(() ->
        chatClient.prompt(prompt).call().content());
}

// Usage
CompletableFuture<String> future = service.generateResponseAsync("Hello");
String result = future.get(30, TimeUnit.SECONDS);
```

Benefits: Non-blocking • Timeout control • Better UX

Testing: Unit Tests with Mocks

```java
@SpringBootTest
class ChatServiceTest {
    @MockitoBean ChatClient chatClient;
    @MockitoBean ChatClient.CallResponseSpec callSpec;

    @Test
    void shouldGenerateResponse() {
        when(chatClient.prompt("Hello")).thenReturn(callSpec);
        when(callSpec.call()).thenReturn(mockResponse("Hi there!"));

        String result = service.generateResponse("Hello");
        assertThat(result).isEqualTo("Hi there!");
    }
}
```

Testing: Integration with TestContainers

```java
@Testcontainers
@SpringBootTest
class RAGIntegrationTest {

    @Container
    static RedisContainer redis = new RedisContainer("redis:7.0")
        .withExposedPorts(6379);

    @Test
    void shouldPerformRAGWithRedis() {
        // Test actual RAG workflow with real Redis
        String answer = ragService.askQuestion("What is Spring AI?");
        assertThat(answer).contains("Spring", "AI");
    }
}
```

Benefits: Real dependencies • Isolated environments • CI/CD ready

Cost & Performance Optimization

Model Selection

- **GPT-4**: Complex reasoning, higher cost
- **GPT-3.5/4o**: Faster, cost-effective for simple tasks
- **Claude**: Strong for analysis, coding
- **Local models**: Privacy, no API costs

Optimization Strategies

- **Token Management**: Monitor usage, optimize prompts
- **Embedding Caching**: Store frequently used vectors
- **Request Batching**: Combine operations when possible
- **Smart Chunking**: Optimize document splitting

Security & Observability

Security Best Practices

- **API Key Management**: Environment variables, vaults
- **Data Privacy**: Local processing when possible
- **Input Validation**: Sanitize user prompts
- **Output Filtering**: Check AI responses

Monitoring & Observability

- **Logging**: All AI interactions and costs
- **Metrics**: Response times, token usage
- **Tracing**: Request flows through services
- **Alerting**: High costs, failed requests

Course Summary

What You’ve Built

Your AI-Powered Application Stack

Foundation & Advanced

- ChatClient integration
- Multiple AI providers
- Streaming responses
- Multimodal capabilities
- Function calling

Production Ready

- RAG systems
- MCP protocol
- Service architecture
- Error handling
- Comprehensive testing

Result: Enterprise-grade Spring AI applications! 🚀

Key Takeaways

1. **Spring AI simplifies AI integration** - No complex API calls or JSON parsing
2. **Start simple, add complexity gradually** - From basic chat to advanced RAG
3. **Leverage Spring's strengths** - Auto-configuration, profiles, testing
4. **Think beyond text** - Vision, audio, and structured data open new possibilities
5. **MCP is the future** - Standardized tool integration across AI platforms
6. **Production requires planning** - Error handling, monitoring, and resilience
7. **Test everything** - AI responses are non-deterministic but testable

Next Steps: Continue Learning

Hands-On Practice

- Explore the GitHub repository
- Try advanced RAG techniques
- Build custom MCP servers
- Integrate with existing apps

Key Resources

- Spring AI Docs
- Model Context Protocol
- Course Repository

Production Considerations

Operational Excellence

- **API Cost Management**: Monitor token usage
- **Rate Limiting**: Handle API quotas
- **Data Privacy**: Keep sensitive data secure
- **Monitoring**: Track performance and errors

Advanced Topics

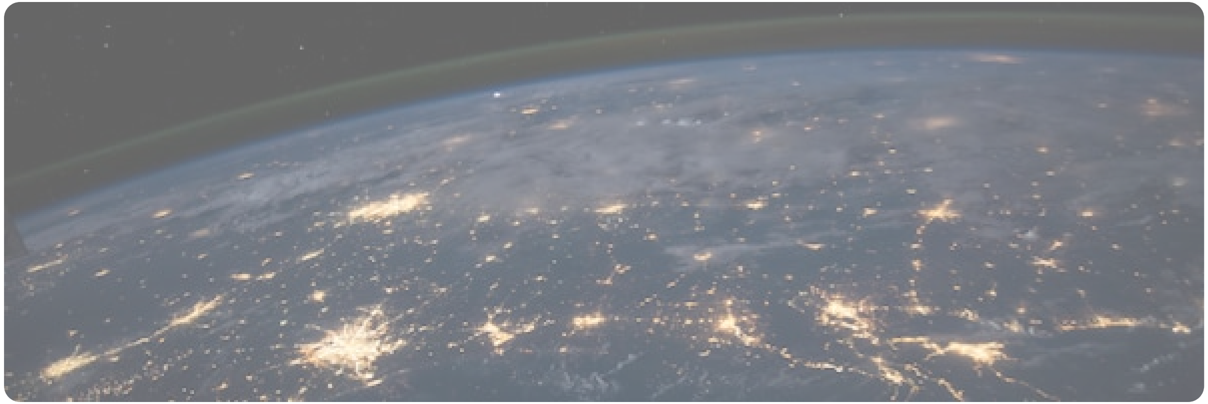
- Custom embedding models
- Multi-agent systems
- AI-powered workflows
- Integration with existing systems

Thank You!

Questions?

Kenneth Kousen
Author, Speaker, Java Champion

kousenit.com | @kenkousen



Ready to build intelligent applications with Spring AI!