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

```
1   Spring_AI_Training_Course/
2   ├── labs.md              # 15 progressive lab exercises
3   ├── src/
4   │   ├── main/java/        # Service implementations
5   │   ├── main/resources/   # Configuration & templates
6   │   └── test/java/        # Test-driven exercises
7   ├── README.md            # Course documentation
8   └── 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 3.5.4**
- **Spring AI 1.0.0**
- **Git** for branch management
- **Redis** (optional, for advanced RAG)

Environment Setup

```
1    # Required API keys
2    export OPENAI_API_KEY=your_key
3    export ANTHROPIC_API_KEY=your_key
4
5    # Optional: Redis for advanced labs
6    docker run -p 6379:6379 redis/redis-stack:latest
7
8    # Clone and start
9    git clone <repo-url>
10   ./gradlew build
11   ./gradlew test
```

Lab 1-3: Foundations

Building Your First AI-Powered Spring Application

Lab 1: Basic Chat Interactions

```java
1  // Step 2: Complete implementation
2  @Service
3  public class ChatService {
4      private final ChatClient chatClient;
5
6      public ChatService(ChatClient.Builder builder) {
7          this.chatClient = builder.build();
8      }
9
10     public String generateResponse(String prompt) {
11         return chatClient.prompt(prompt)
12             .call()
13             .content();
14     }
15 }
```

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

Lab 2: Request/Response Logging

```java
1    @Service
2    public class ChatService {
3        private final ChatClient chatClient;
4
5        public ChatService(ChatClient.Builder builder) {
6
7            // Add logging advisor for debugging
8            this.chatClient = builder
9                .defaultAdvisors(new SimpleLoggerAdvisor())
10               .build();
11       }
12
13       public String generateResponse(String prompt) {
14           return chatClient.prompt(prompt)
15               .call()
16               .content();
17       }
18   }
```

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

Lab 3: Streaming Responses

```
1    @RestController
2    public class ChatController {
3
4        @GetMapping(value = "/chat/stream", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
5        public Flux<String> streamChat(@RequestParam String message) {
6            return chatClient.prompt(message)
7                    .stream()
8                    .content();
9        }
10
11       // Frontend receives real-time token-by-token responses
12       // Perfect for chat interfaces and long AI responses
13       // Uses Spring WebFlux Reactor streams
14   }
```

Experience: Real-time AI responses like ChatGPT interface

Lab 4-6: Structured AI

From Text to Objects

Lab 4: Structured Data Extraction

```java
1    // Define your data structure
2    public record ActorFilms(String actor, List<String> movies) {}
3
4    @Test
5    void shouldGetActorFilms() {
6        // AI converts natural language to structured data
7        ActorFilms actorFilms = chatClient.prompt("Generate the filmography for Tom Hanks")
8            .call()
9            .entity(ActorFilms.class);
10
11       // Assert AI returned proper structure
12       assertThat(actorFilms.actor()).isEqualTo("Tom Hanks");
13       assertThat(actorFilms.movies()).contains("Forrest Gump", "Cast Away");
14   }
```

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

Lab 5: Prompt Templates

```java
1    @Component
2    public class TemplateService {
3
4        @Value("classpath:/prompts/actor-filmography.st")
5        private Resource actorFilmographyTemplate;
6
7        public ActorFilms getActorFilms(String actorName) {
8            return chatClient.prompt()
9                    .user(userSpec -> userSpec
10                           .text(actorFilmographyTemplate)
11                           .param("actor", actorName)
12                           .param("count", 5))
13                   .call()
14                   .entity(ActorFilms.class);
15       }
16   }
```

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

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

Lab 6: Chat Memory

```
1    @Service
2    public class ConversationService {
3        private final ChatClient chatClient;
4
5        public ConversationService(ChatClient.Builder builder) {
6            this.chatClient = builder
7                .defaultAdvisors(new MessageChatMemoryAdvisor(
8                    new InMemoryChatMemory())) // Remembers conversation
9                .build();
10       }
11
12       public String continueConversation(String message) {
13           return chatClient.prompt(message)
14               .call()
15               .content();
16           // AI remembers previous messages in this conversation!
17       }
18   }
```

Try this:

1. "My name is John"

Lab 7-9: Multimodal AI

Beyond Text: Vision and Audio

Lab 7: Vision Capabilities

```java
1    @Test
2    void shouldAnalyzeImage() {
3        var imageResource = new ClassPathResource(
4            "/images/multimodal_test_image.png");
5
6        String response = chatClient.prompt()
7            .user(userSpec -> userSpec
8                .text("What do you see in this image?")
9                .media(MimeTypeUtils.IMAGE_PNG, imageResource))
10           .call()
11           .content();
12
13       assertThat(response.toLowerCase())
14           .contains("dog", "playing");
15   }
```

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)

```java
1   // Step 3: AI automatically calls your tools
2   @Test
3   void shouldCallTool() {
4       String response = chatClient.prompt(
5           "What time is it right now?")
6           .call()
7           .content();
8
9       // AI called getCurrentDateTime() automatically!
10      assertThat(response).contains("2024");
11  }
```

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
1   @Service
2   public class SpeechService {
3       private final AudioSpeechModel speechModel;
4
5       public SpeechService(AudioSpeechModel speechModel) {
6           this.speechModel = speechModel;
7       }
8
9       public byte[] generateSpeech(String text) {
10          AudioSpeechPrompt prompt = new AudioSpeechPrompt(text,
11              AudioSpeechOptionsBuilder.builder()
12                  .withModel("tts-1")
13                  .withVoice(AudioSpeechOptions.Voice.ALLOY)
14                  .build());
15
16          return speechModel.call(prompt).getResult().getOutput();
17      }
18  }
```

Text-to-Speech: Usage

```java
1   // Generate and save audio
2   byte[] audioData = speechService.generateSpeech(
3       "Welcome to Spring AI training!");
4
5   // Save to file for playback
6   Files.write(Paths.get("welcome.mp3"), audioData);
```

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

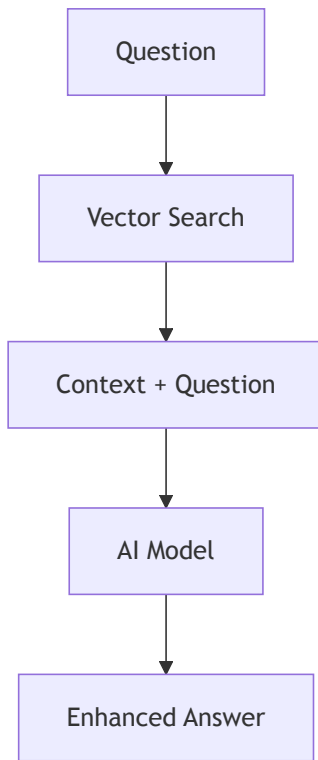# Lab 12-13: RAG Systems

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

```
┌─────────────────────┐
│      Question       │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│    Vector Search    │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│  Context + Question │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│      AI Model       │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│   Enhanced Answer   │
└─────────────────────┘
```

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
1  // Step 3: Testing RAG
2  @Test
3  void shouldAnswerFromDocuments() {
4      String answer = ragService.askQuestion(
5          "What is Spring AI ChatClient?");
6
7      // AI uses loaded PDF content to answer!
8      assertThat(answer).contains("ChatClient", "Spring AI");
9  }
```

Magic: AI answers questions using your PDF documents!

Lab 13: Production RAG with Redis

```java
1   @Configuration
2   @Profile("redis")
3   public class RedisRAGConfig {
4
5       @Bean
6       public VectorStore vectorStore(RedisConnectionFactory factory) {
7           return new RedisVectorStore(factory, embeddingModel());
8       }
9
10      @Bean
11      public ApplicationRunner dataLoader(VectorStore vectorStore) {
12          return args -> loadDocumentsIfEmpty(vectorStore);
13      }
14  }
```

Redis RAG: Smart Data Loading

```
 1    @Bean
 2    public ApplicationRunner dataLoader(VectorStore vectorStore) {
 3        return args -> {
 4            if (vectorStore instanceof RedisVectorStore redis &&
 5                redis.getCollection().isEmpty()) {
 6
 7                // Only load documents if Redis is empty
 8                loadDocuments(vectorStore);
 9                log.info("Loaded {} documents into Redis", count);
10            }
11        };
12    }
```

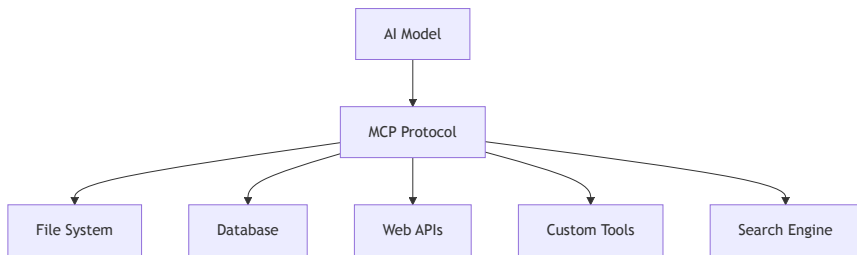**Benefits**: Persistent • Scalable • Fast similarity search

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`):

```
 1  {
 2    "mcpServers": {
 3      "filesystem": {
 4        "command": "npx",
 5        "args": ["-y", "@modelcontextprotocol/server-filesystem", "/tmp"]
 6      },
 7      "search": {
 8        "command": "npx",
 9        "args": ["-y", "@modelcontextprotocol/server-brave-search"]
10      }
11    }
12  }
```

Result: AI can access external tools through standardized protocol

Lab 15: MCP Server

```
1   # Step 3: Run MCP server
2   ./gradlew bootRun --args='--spring.profiles.active=mcp-server'
3
4   # Step 4: Connect from Claude Desktop
5   # Add to Claude's MCP config:
6   {
7     "mcpServers": {
8       "spring-calculator": {
9         "command": "java",
10        "args": ["-jar", "app.jar", "--spring.profiles.active=mcp-server"]
11      }
12    }
13  }
```

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

Production Patterns

Enterprise-Ready Spring AI

Service Layer: REST Controller

```java
1   @RestController
2   @RequestMapping("/api/ai")
3   public class AIController {
4       private final ChatService chatService;
5
6       @PostMapping("/chat")
7       public ResponseEntity<String> chat(@RequestBody ChatRequest request) {
8           try {
9               String response = chatService.generateResponse(request.getMessage());
10              return ResponseEntity.ok(response);
11          } catch (Exception e) {
12              return ResponseEntity.status(500)
13                  .body("AI service temporarily unavailable");
14          }
15      }
```

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
1   @Bean
2   @ConditionalOnProfile({"rag", "redis"})
3   public VectorStore redisVectorStore(RedisConnectionFactory factory) {
4       return new RedisVectorStore(factory, embeddingModel());
5   }
6
7   @Bean
8   @ConditionalOnProfile("mcp")
9   public McpClientConfiguration mcpConfig() {
10      return new McpClientConfiguration();
11  }
```

Run configurations:

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

Error Handling: Retries

```java
1   @Service
2   public class ResilientChatService {
3       private final ChatClient chatClient;
4
5       @Retryable(value = {ApiException.class}, maxAttempts = 3)
6       public String generateResponse(String prompt) {
7           return chatClient.prompt(prompt).call().content();
8       }
9
10      // Automatically retries API failures
11      // Exponential backoff available
12      // Works with Spring Retry
13  }
```

Error Handling: Circuit Breaker

```java
1   @CircuitBreaker(name = "ai-service", fallbackMethod = "fallbackResponse")
2   public String generateResponseWithCircuitBreaker(String prompt) {
3       return chatClient.prompt(prompt).call().content();
4   }
5
6   public String fallbackResponse(String prompt, Exception ex) {
7       log.warn("AI service failed, using fallback", ex);
8       return "I'm temporarily unable to process your request.";
9   }
10
11  // Prevents cascade failures
12  // Automatic recovery when service improves
```

Error Handling: Async Processing

```java
1   @Async
2   public CompletableFuture<String> generateResponseAsync(String prompt) {
3       return CompletableFuture.supplyAsync(() ->
4           chatClient.prompt(prompt).call().content());
5   }
6
7   // Usage
8   CompletableFuture<String> future = service.generateResponseAsync("Hello");
9   String result = future.get(30, TimeUnit.SECONDS);
```

Benefits: Non-blocking • Timeout control • Better UX

Testing: Unit Tests with Mocks

```java
1   @SpringBootTest
2   class ChatServiceTest {
3       @MockitoBean ChatClient chatClient;
4       @MockitoBean ChatClient.CallResponseSpec callSpec;
5
6       @Test
7       void shouldGenerateResponse() {
8           when(chatClient.prompt("Hello")).thenReturn(callSpec);
9           when(callSpec.call()).thenReturn(mockResponse("Hi there!"));
10
11          String result = service.generateResponse("Hello");
12          assertThat(result).isEqualTo("Hi there!");
13      }
14  }
```

Testing: Integration with TestContainers

```java
1    @Testcontainers
2    @SpringBootTest
3    class RAGIntegrationTest {
4
5        @Container
6        static RedisContainer redis = new RedisContainer("redis:7.0")
7            .withExposedPorts(6379);
8
9        @Test
10       void shouldPerformRAGWithRedis() {
11           // Test actual RAG workflow with real Redis
12           String answer = ragService.askQuestion("What is Spring AI?");
13           assertThat(answer).contains("Spring", "AI");
14       }
15   }
```

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

Result: Enterprise-grade Spring AI applications! 🚀

Production Ready

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

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
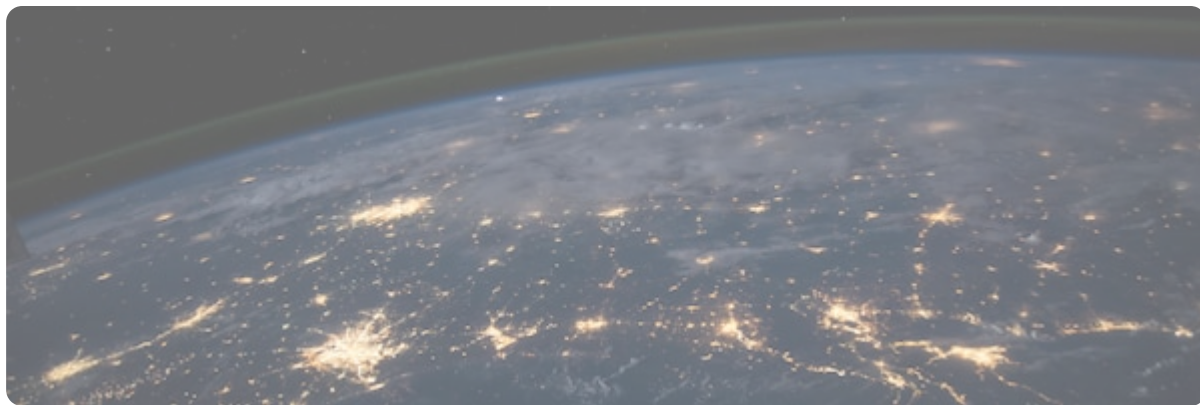- Integration with existing systems

Thank You!

Questions?

Kenneth Kousen

Author, Speaker, Java Champion

kousenit.com | @kenkousen

Ready to build intelligent applications with Spring AI!