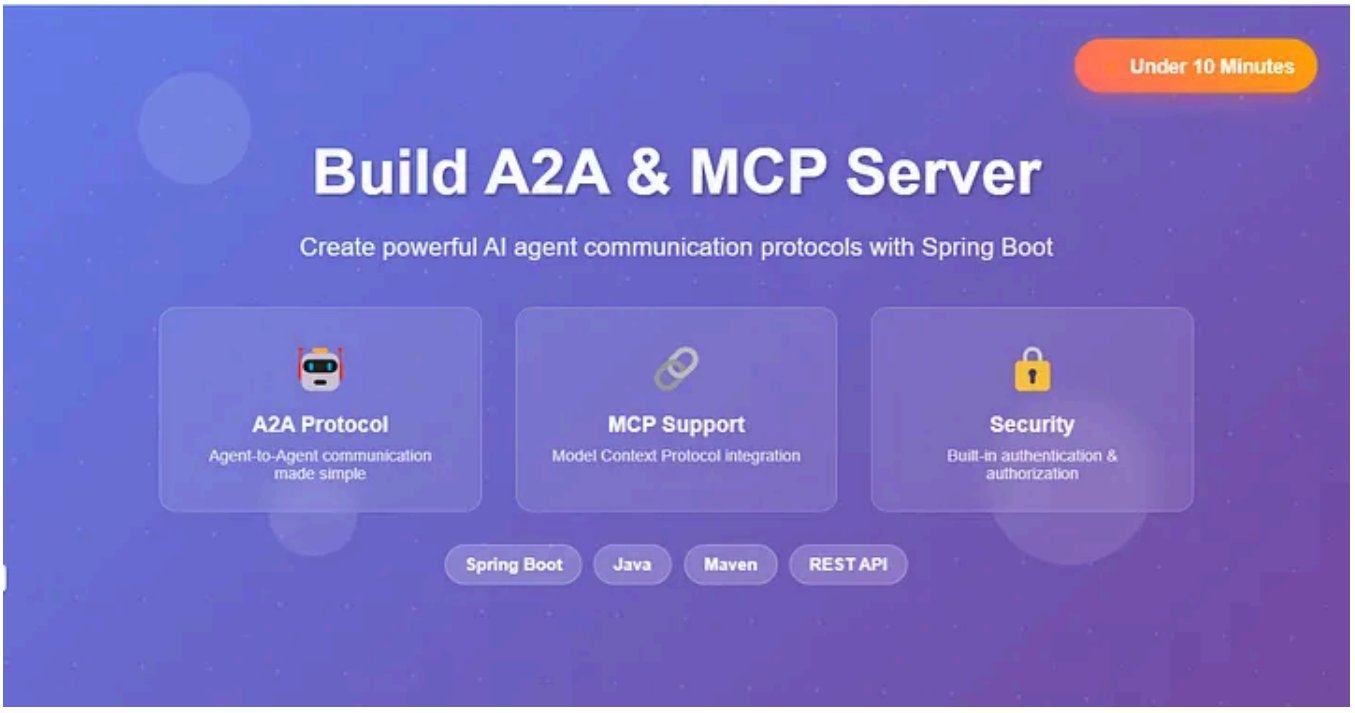

Build an A2A and Model Context Protocol (MCP) Server in Under 10 Minutes

3 min read · May 28, 2025

Vishal Mysore Follow

Listen Share ••• More

This guide will walk you through creating a Spring Boot application that supports both A2A (Agent-to-Agent) and MCP (Model Context Protocol) protocols. You'll learn how to build an AI agent that can communicate using both protocols and implement security features.

Prerequisites

- Java 8 or higher
- Maven
- Basic knowledge of Spring Boot

- IDE (VS Code, IntelliJ, or Eclipse)

Step 1: Create a Spring Boot Project

First, create a new Spring Boot project using your preferred method (Spring Initializr or IDE). Then, add the following dependencies to your `pom.xml`:

```
<!-- Tools4AI dependencies -->
<dependency>
    <groupId>io.github.vishalmysore</groupId>
    <artifactId>a2ajava</artifactId>
    <version>0.1.8.2</version>
</dependency>
<dependency>
    <groupId>io.github.vishalmysore</groupId>
    <artifactId>tools4ai-annotations</artifactId>
    <version>0.0.2</version>
</dependency>
<dependency>
    <groupId>io.github.vishalmysore</groupId>
    <artifactId>tools4ai-security</artifactId>
    <version>0.0.3</version>
</dependency>
<dependency>
    <groupId>io.github.vishalmysore</groupId>
    <artifactId>tools4ai</artifactId>
    <version>1.1.5</version>
    <exclusions>
        <exclusion>
            <groupId>io.swagger.core.v3</groupId>
            <artifactId>swagger-core</artifactId>
        </exclusion>
        <exclusion>
            <groupId>io.swagger.core.v3</groupId>
            <artifactId>swagger-core-jakarta</artifactId>
        </exclusion>
        <exclusion>
            <groupId>io.swagger.parser.v3</groupId>
            <artifactId>swagger-parser</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Step 2: Configure Your Application

Basic Configuration

Create or update your main application class with the required annotations:

```
@SpringBootApplication
@EnableAgent                    // Enables A2A/MCP agent functionality
@EnableAgentSecurity           // Optional: Enables security features
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Security Configuration (Optional)

If you enabled security with `@EnableAgentSecurity`, create a security configuration class:

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
        http
            .csrf(csrf -> csrf.disable())
            .headers(headers -> headers
                .frameOptions(frame -> frame.sameOrigin())
            )
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/public/**", "/swagger-ui/**",
                                 "/v3/api-docs/**", "/.well-known/agent.json",
                                 "/", "/index.html").permitAll()
                .anyRequest().authenticated()
            )
            .httpBasic(Customizer.withDefaults());

        return http.build();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails user = User.builder()
            .username("user")
            .password(passwordEncoder().encode("password"))
            .roles("USER")
            .build();

        UserDetails admin = User.builder()
            .username("admin")
            .password(passwordEncoder().encode("admin"))
```

```
                .roles("ADMIN")
                .build();

        return new InMemoryUserDetailsManager(user, admin);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Step 3: Create Your Agent Service

Create a service class with your agent's functionality:

```
@Log
@Service
@Agent(groupName = "Your Agent Group Name")
public class YourAgentService {
@PreAuthorize("hasRole('USER')")    // Optional: Add security
    @Action(description = "Description of what this action does")
    public String yourAction(String param) {
        // Your action implementation
        return "Action result";
    }
}
```

Step 4: Configure Properties

Create `application.properties`:

```
spring.application.name=your-agent-name
server.port=7860
a2a.persistence=cache
```

Step 5: Testing Your Agent

A2A Protocol Testing

Access your agent's A2A card:

```
curl http://localhost:7860/.well-known/agent.json
```

Test with authentication:

```
curl -u user:password http://localhost:7860/.well-known/agent.json
```

MCP Protocol Testing

List available tools:

```
curl -H "Content-Type: application/json" \
    -d '{"jsonrpc":"2.0","method":"tools/list","params":{},"id":1}' \
    http://localhost:7860/
```

Call a tool:

```
curl -u admin:admin -H "Content-Type: application/json" \
    -d '{
        "jsonrpc": "2.0",
        "method": "tools/call",
        "params": {
            "name": "yourAction",
            "arguments": {
                "param": "value"
            }
        },
        "id": 1
    }' \
    http://localhost:7860/
```

Step 6: MCP Connector Integration

To connect your server with MCP clients, you can use the mcp-connector. Add this to your MCP configuration:

```
{
  "myagent": {
    "command": "java",
    "args": [
      "-jar",
      "/path/to/mcp-connector-full.jar",
      "http://localhost:7860/"
    ],
    "timeout": 30000
  }
}
```

For more details about the MCP connector, visit:
https://github.com/vishalmysore/mcp-connector

Advanced Topics

- OAuth2 integration
- Custom authentication providers
- Advanced action handling
- Callback mechanisms
- File handling and resource management

These topics will be covered in Part 2 of this guide.

Conclusion

You now have a working A2A/MCP agent server with:

- Protocol support for both A2A and MCP
- Security implementation (if enabled)
- Basic action handling
- Ready for extension with your custom functionality

Next Steps

- Add your custom actions
- Implement more complex security

- Add database integration
- Implement file handling
- Add custom protocols

For more examples and detailed implementation, check out the full project at: https://github.com/vishalmysore/a2ajava

Java Google Model Context Protocol A2a Artificial Intelligence

Follow

Written by Vishal Mysore

665 followers · 4 following

Holder of multiple patents in AI and software engineering. Passionate about building scalable systems, optimizing performance, & driving AI-powered innovation.

No responses yet

Bgerby

What are your thoughts?

More from Vishal Mysore

Vishal Mysore

What is BMAD-METHOD™? A Simple Guide to the Future of AI-Driven Development

The BMAD-METHOD™ (Breakthrough Method of Agile AI-Driven Development) framework offers a surprisingly simple, yet highly innovative...

Sep 8

Vishal Mysore

🤖 A Tale of Two Frameworks: BMAD-Method vs. GitHub Spec Kit

BMAD-Method is a multi-agent methodology. Think of it as a complete project team in a box: you have an Analyst Agent, a Product Manager...

Sep 15

Vishal Mysore

GitHub Spec Kit vs BMAD-Method: A Comprehensive Comparison : Part 1

GitHub's open-source Spec Kit is a toolkit for spec-driven development that provides a structured process to bring specification-first...

Sep 15 2

Vishal Mysore

BMAD-Method : From Zero To Hero

The BMAD-METHOD is a way of working with AI that makes projects more organized, repeatable, and team-friendly.

Sep 14

See all from Vishal Mysore

Recommended from Medium

Vishal Mysore

LLM Monitoring for Java Applications: Detect and Audit AI Calls Automatically

Java applications are increasingly integrating Large Language Model (LLM) APIs from OpenAI, Anthropic, Cohere, and other AI providers —...

Oct 1

Udipta Das

Implement an Agentic AI Workflow in Spring Boot with Spring AI Tool Invocation

In this guide, we'll walk through a hands-on example using Spring AI to create an agentic AI flow within a Spring Boot application. This...

Jun 21 3

Gourav Kumar Bhardwaj

Spring Boot with LangChain4j Setup (Part 1)

As Java developers, we've often looked at Python folks with envy. They've had LangChain, LlamaIndex, and an ecosystem of AI-first tools...

Sep 28

In Stackademic by RAKTIM SINGH

AI Orchestration Layer: Why A2A and MCP Aren’t Enough for Multi-Agent Systems

Discover why A2A and MCP protocols alone can’t scale multi-agent AI systems, and how orchestration layers bring safety, governance, and...

Sep 28

Sanjeeb Panda

Model Context Protocol (MCP) in AI Agent Development :Text To Sql

Introduction

Apr 17 2

Saima Khan

From Monolith to Modular (Part 2): How I Integrated MCP Using FastMCP and LangGraph

Plug-and-Play AI with FastMCP and LangGraph

Jul 11