Complete Java Spring Boot & HTTP Server Guide

Table of Contents

Java Spring Boot Overview

What is Spring Boot?

Spring Boot is a Java-based framework that simplifies the development of stand-alone, production-grade Spring-based applications. It provides:

- **Auto-configuration**: Automatically configures Spring applications based on dependencies
- **Embedded servers**: Includes Tomcat, Jetty, or Undertow servers
- **Production-ready features**: Health checks, metrics, externalized configuration
- **Opinionated defaults**: Reduces boilerplate configuration

Core Features

- **Dependency Injection**: Manages object dependencies automatically
- **Auto-Configuration**: Configures beans based on classpath dependencies
- **Actuator**: Provides production-ready monitoring and management features
- **Data Access**: Simplified database integration with Spring Data
- **Security**: Built-in security configurations
- **Testing**: Comprehensive testing support

Architecture Components

- **Controllers**: Handle HTTP requests and responses
- **Services**: Business logic layer
- **Repositories**: Data access layer
- **Entities/Models**: Data representation objects
- **Configuration**: Application settings and bean definitions

---

Setting up Spring Boot Project in IntelliJ IDEA

Method 1: Using Spring Initializr (Recommended)

Step 1: Create New Project

1. Open IntelliJ IDEA
2. Click "New Project" or "File" → "New" → "Project"
3. Select "Spring Initializr" from the left panel
4. Configure project settings:
    - **Server URL**: https://start.spring.io
    - **Name**: your-project-name
    - **Location**: project directory path
    - **Language**: Java
    - **Type**: Maven Project (or Gradle)
    - **Group**: com.example
    - **Artifact**: demo
    - **Package name**: com.example.demo
    - **Project SDK**: Java 11+ (recommended Java 17 or 21)

Step 2: Select Dependencies

Choose the following dependencies for a REST API project:

- **Spring Web**: For building web applications and REST APIs
- **Spring Boot DevTools**: For development-time features
- **Spring Data JPA**: For database operations
- **H2 Database**: In-memory database for development
- **Spring Boot Starter Validation**: For input validation

Step 3: Project Structure

After creation, your project structure will look like:

```
src/
├── main/
│   ├── java/
│   │   └── com/example/demo/
│   │       ├── DemoApplication.java
│   │       ├── controller/
│   │       ├── service/
│   │       ├── repository/
│   │       └── model/
│   └── resources/
│       ├── application.properties
│       ├── static/
│       └── templates/
└── test/
    └── java/
        └── com/example/demo/
```

Method 2: Manual Setup

Step 1: Create Maven Project

1. New Project → Maven → Create from archetype
2. Select `maven-archetype-quickstart`
3. Configure GroupId and ArtifactId

Step 2: Configure pom.xml

```xml
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
        <relativePath/>
    </parent>

    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
```

```
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Step 3: Create Main Application Class

```java
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Configuring REST API in Spring Boot

Step 1: Create Entity Class

```java
```

```
package com.example.demo.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Email;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Name is required")
    @Column(nullable = false)
    private String name;

    @Email(message = "Email should be valid")
    @Column(nullable = false, unique = true)
    private String email;

    // Constructors
    public User() {}

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

Step 2: Create Repository Interface

```
java
```

```java
package com.example.demo.repository;

import com.example.demo.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
    boolean existsByEmail(String email);
}
```

Step 3: Create Service Layer

java

```java
package com.example.demo.service;

import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    public Optional<User> getUserById(Long id) {
        return userRepository.findById(id);
    }

    public User createUser(User user) {
        if (userRepository.existsByEmail(user.getEmail())) {
            throw new RuntimeException("Email already exists");
        }
        return userRepository.save(user);
    }

    public User updateUser(Long id, User userDetails) {
        User user = userRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("User not found"));

        user.setName(userDetails.getName());
        user.setEmail(userDetails.getEmail());
        return userRepository.save(user);
    }

    public void deleteUser(Long id) {
        if (!userRepository.existsById(id)) {
            throw new RuntimeException("User not found");
        }
        userRepository.deleteById(id);
    }
}
```

Step 4: Create REST Controller

```
java
```

```java
package com.example.demo.controller;

import com.example.demo.model.User;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import jakarta.validation.Valid;
import java.util.List;

@RestController
@RequestMapping("/api/users")
@Validated
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping
    public ResponseEntity<List<User>> getAllUsers() {
        List<User> users = userService.getAllUsers();
        return ResponseEntity.ok(users);
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
        return userService.getUserById(id)
            .map(user -> ResponseEntity.ok(user))
            .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<User> createUser(@Valid @RequestBody User user) {
        try {
            User createdUser = userService.createUser(user);
            return ResponseEntity.status(HttpStatus.CREATED).body(createdUser);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().build();
        }
    }

    @PutMapping("/{id}")
    public ResponseEntity<User> updateUser(@PathVariable Long id,
                                          @Valid @RequestBody User userDetails) {
```

```
        try {
            User updatedUser = userService.updateUser(id, userDetails);
            return ResponseEntity.ok(updatedUser);
        } catch (RuntimeException e) {
            return ResponseEntity.notFound().build();
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
        try {
            userService.deleteUser(id);
            return ResponseEntity.noContent().build();
        } catch (RuntimeException e) {
            return ResponseEntity.notFound().build();
        }
    }
}
```

Step 5: Configure Application Properties

Create `src/main/resources/application.properties`:

```properties
# Server configuration
server.port=8080

# H2 Database configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# JPA configuration
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true

# H2 Console (for development)
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Step 6: Exception Handling

```java
```

```
package com.example.demo.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>> handleValidationExceptions(
            MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getAllErrors().forEach((error) -> {
            String fieldName = ((FieldError) error).getField();
            String errorMessage = error.getDefaultMessage();
            errors.put(fieldName, errorMessage);
        });
        return ResponseEntity.badRequest().body(errors);
    }

    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<String> handleRuntimeException(RuntimeException ex) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(ex.getMessage());
    }
}
```

Running the Application

1. Right-click on `DemoApplication.java` in IntelliJ
2. Select "Run DemoApplication"
3. Or use Maven: `mvn spring-boot:run`
4. Access API at `http://localhost:8080/api/users`

Testing REST Endpoints

Use tools like Postman or curl:

```bash
```

```
# GET all users
curl -X GET http://localhost:8080/api/users

# POST new user
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name":"John Doe","email":"john@example.com"}'

# GET user by ID
curl -X GET http://localhost:8080/api/users/1

# PUT update user
curl -X PUT http://localhost:8080/api/users/1 \
  -H "Content-Type: application/json" \
  -d '{"name":"Jane Doe","email":"jane@example.com"}'

# DELETE user
curl -X DELETE http://localhost:8080/api/users/1
```

Building HTTP Server from Scratch

Simple HTTP Server Implementation

java

```java
package com.example.httpserver;

import java.io.*;
import java.net.*;
import java.util.*;
import java.util.concurrent.*;

public class SimpleHttpServer {
    private final int port;
    private final ExecutorService threadPool;
    private ServerSocket serverSocket;
    private boolean running;
    private final Map<String, RouteHandler> routes;

    public SimpleHttpServer(int port) {
        this.port = port;
        this.threadPool = Executors.newFixedThreadPool(10);
        this.routes = new HashMap<>();
        this.running = false;
    }

    public void start() throws IOException {
        serverSocket = new ServerSocket(port);
        running = true;

        System.out.println("Server started on port " + port);

        while (running) {
            try {
                Socket clientSocket = serverSocket.accept();
                threadPool.submit(new ClientHandler(clientSocket));
            } catch (IOException e) {
                if (running) {
                    System.err.println("Error accepting client connection: " + e.getMessage());
                }
            }
        }
    }

    public void stop() throws IOException {
        running = false;
        if (serverSocket != null) {
            serverSocket.close();
        }
        threadPool.shutdown();
    }
```

```java
public void addRoute(String method, String path, RouteHandler handler) {
    routes.put(method + " " + path, handler);
}

private class ClientHandler implements Runnable {
    private final Socket clientSocket;

    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
    }

    @Override
    public void run() {
        try (BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
             PrintWriter out = new PrintWriter(
                clientSocket.getOutputStream(), true)) {

            HttpRequest request = parseRequest(in);
            HttpResponse response = handleRequest(request);
            sendResponse(out, response);

        } catch (IOException e) {
            System.err.println("Error handling client: " + e.getMessage());
        } finally {
            try {
                clientSocket.close();
            } catch (IOException e) {
                System.err.println("Error closing client socket: " + e.getMessage());
            }
        }
    }

    private HttpRequest parseRequest(BufferedReader in) throws IOException {
        String requestLine = in.readLine();
        if (requestLine == null) {
            throw new IOException("Empty request");
        }

        String[] parts = requestLine.split(" ");
        if (parts.length != 3) {
            throw new IOException("Invalid request line");
        }

        String method = parts[0];
        String path = parts[1];
        String version = parts[2];

        Map<String, String> headers = new HashMap<>();
        String line;
        while ((line = in.readLine()) != null && !line.isEmpty()) {
            String[] headerParts = line.split(": ", 2);
            if (headerParts.length == 2) {
                headers.put(headerParts[0].toLowerCase(), headerParts[1]);
            }
        }

        StringBuilder bodyBuilder = new StringBuilder();
        if (headers.containsKey("content-length")) {
            int contentLength = Integer.parseInt(headers.get("content-length"));
            char[] buffer = new char[contentLength];
            in.read(buffer, 0, contentLength);
            bodyBuilder.append(buffer);
        }

        return new HttpRequest(method, path, version, headers, bodyBuilder.toString());
    }

    private HttpResponse handleRequest(HttpRequest request) {
        String routeKey = request.getMethod() + " " + request.getPath();
        RouteHandler handler = routes.get(routeKey);

        if (handler != null) {
            return handler.handle(request);
        } else {
            return new HttpResponse(404, "Not Found",
                Map.of("Content-Type", "text/plain"), "404 - Not Found");
        }
    }

    private void sendResponse(PrintWriter out, HttpResponse response) {
        out.println("HTTP/1.1 " + response.getStatusCode() + " " + response.getStatusText());

        for (Map.Entry<String, String> header : response.getHeaders().entrySet()) {
            out.println(header.getKey() + ": " + header.getValue());
        }

        out.println("Content-Length: " + response.getBody().length());
        out.println(); // Empty line to separate headers from body
        out.print(response.getBody());
        out.flush();
    }
}
```

```java
// HTTP Request class
public static class HttpRequest {
    private final String method;
    private final String path;
    private final String version;
    private final Map<String, String> headers;
    private final String body;

    public HttpRequest(String method, String path, String version,
                     Map<String, String> headers, String body) {
        this.method = method;
        this.path = path;
        this.version = version;
        this.headers = headers;
        this.body = body;
    }

    // Getters
    public String getMethod() { return method; }
    public String getPath() { return path; }
    public String getVersion() { return version; }
    public Map<String, String> getHeaders() { return headers; }
    public String getBody() { return body; }

    public String getHeader(String name) {
        return headers.get(name.toLowerCase());
    }
}

// HTTP Response class
public static class HttpResponse {
    private final int statusCode;
    private final String statusText;
    private final Map<String, String> headers;
    private final String body;

    public HttpResponse(int statusCode, String statusText,
                      Map<String, String> headers, String body) {
        this.statusCode = statusCode;
        this.statusText = statusText;
        this.headers = headers;
        this.body = body;
    }

    // Getters
    public int getStatusCode() { return statusCode; }
```

```java
        public String getStatusText() { return statusText; }
        public Map<String, String> getHeaders() { return headers; }
        public String getBody() { return body; }
    }

    // Route Handler interface
    @FunctionalInterface
    public interface RouteHandler {
        HttpResponse handle(HttpRequest request);
    }

    // Main method to demonstrate usage
    public static void main(String[] args) {
        SimpleHttpServer server = new SimpleHttpServer(8080);

        // Add routes
        server.addRoute("GET", "/", (request) -> {
            String html = "<html><body><h1>Welcome to Simple HTTP Server</h1></body></html>";
            return new HttpResponse(200, "OK",
                Map.of("Content-Type", "text/html"), html);
        });

        server.addRoute("GET", "/api/hello", (request) -> {
            String json = "{\"message\": \"Hello, World!\", \"timestamp\": " +
                         System.currentTimeMillis() + "}";
            return new HttpResponse(200, "OK",
                Map.of("Content-Type", "application/json"), json);
        });

        server.addRoute("POST", "/api/echo", (request) -> {
            String json = "{\"echo\": \"" + request.getBody() + "\"}";
            return new HttpResponse(200, "OK",
                Map.of("Content-Type", "application/json"), json);
        });

        // Start server
        try {
            server.start();
        } catch (IOException e) {
            System.err.println("Failed to start server: " + e.getMessage());
        }
    }
}
```

Enhanced HTTP Server with JSON Support

java

```java
package com.example.httpserver;

import com.fasterxml.jackson.databind.ObjectMapper;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.concurrent.*;

public class JsonHttpServer {
    private final SimpleHttpServer server;
    private final ObjectMapper objectMapper;
    private final List<User> users;
    private long nextId = 1;

    public JsonHttpServer(int port) {
        this.server = new SimpleHttpServer(port);
        this.objectMapper = new ObjectMapper();
        this.users = new ArrayList<>();
        setupRoutes();
    }

    private void setupRoutes() {
        // GET all users
        server.addRoute("GET", "/api/users", this::getAllUsers);

        // GET user by ID
        server.addRoute("GET", "/api/users/{id}", this::getUserById);

        // POST create user
        server.addRoute("POST", "/api/users", this::createUser);

        // PUT update user
        server.addRoute("PUT", "/api/users/{id}", this::updateUser);

        // DELETE user
        server.addRoute("DELETE", "/api/users/{id}", this::deleteUser);
    }

    private SimpleHttpServer.HttpResponse getAllUsers(SimpleHttpServer.HttpRequest request) {
        try {
            String json = objectMapper.writeValueAsString(users);
            return new SimpleHttpServer.HttpResponse(200, "OK",
                Map.of("Content-Type", "application/json"), json);
        } catch (Exception e) {
            return errorResponse(500, "Internal Server Error");
        }
```

```java
}

private SimpleHttpServer.HttpResponse getUserById(SimpleHttpServer.HttpRequest request) {
    try {
        Long id = extractIdFromPath(request.getPath());
        User user = users.stream()
            .filter(u -> u.getId().equals(id))
            .findFirst()
            .orElse(null);

        if (user == null) {
            return errorResponse(404, "User not found");
        }

        String json = objectMapper.writeValueAsString(user);
        return new SimpleHttpServer.HttpResponse(200, "OK",
            Map.of("Content-Type", "application/json"), json);
    } catch (Exception e) {
        return errorResponse(400, "Invalid request");
    }
}

private SimpleHttpServer.HttpResponse createUser(SimpleHttpServer.HttpRequest request) {
    try {
        User user = objectMapper.readValue(request.getBody(), User.class);
        user.setId(nextId++);
        users.add(user);

        String json = objectMapper.writeValueAsString(user);
        return new SimpleHttpServer.HttpResponse(201, "Created",
            Map.of("Content-Type", "application/json"), json);
    } catch (Exception e) {
        return errorResponse(400, "Invalid JSON");
    }
}

private SimpleHttpServer.HttpResponse updateUser(SimpleHttpServer.HttpRequest request) {
    try {
        Long id = extractIdFromPath(request.getPath());
        User existingUser = users.stream()
            .filter(u -> u.getId().equals(id))
            .findFirst()
            .orElse(null);

        if (existingUser == null) {
            return errorResponse(404, "User not found");
        }
```

```java
            User updatedUser = objectMapper.readValue(request.getBody(), User.class);
            existingUser.setName(updatedUser.getName());
            existingUser.setEmail(updatedUser.getEmail());

            String json = objectMapper.writeValueAsString(existingUser);
            return new SimpleHttpServer.HttpResponse(200, "OK",
                Map.of("Content-Type", "application/json"), json);
        } catch (Exception e) {
            return errorResponse(400, "Invalid request");
        }
    }

    private SimpleHttpServer.HttpResponse deleteUser(SimpleHttpServer.HttpRequest request) {
        try {
            Long id = extractIdFromPath(request.getPath());
            boolean removed = users.removeIf(u -> u.getId().equals(id));

            if (!removed) {
                return errorResponse(404, "User not found");
            }

            return new SimpleHttpServer.HttpResponse(204, "No Content",
                Map.of(), "");
        } catch (Exception e) {
            return errorResponse(400, "Invalid request");
        }
    }

    private Long extractIdFromPath(String path) {
        String[] parts = path.split("/");
        return Long.parseLong(parts[parts.length - 1]);
    }

    private SimpleHttpServer.HttpResponse errorResponse(int statusCode, String message) {
        String json = "{\"error\": \"" + message + "\"}";
        return new SimpleHttpServer.HttpResponse(statusCode, message,
            Map.of("Content-Type", "application/json"), json);
    }

    public void start() throws IOException {
        server.start();
    }

    public void stop() throws IOException {
        server.stop();
    }
    // User class for JSON serialization
    public static class User {
        private Long id;
        private String name;
        private String email;

        public User() {}

        public User(String name, String email) {
            this.name = name;
            this.email = email;
        }

        // Getters and setters
        public Long getId() { return id; }
        public void setId(Long id) { this.id = id; }

        public String getName() { return name; }
        public void setName(String name) { this.name = name; }

        public String getEmail() { return email; }
        public void setEmail(String email) { this.email = email; }
    }

    public static void main(String[] args) {
        JsonHttpServer server = new JsonHttpServer(8080);

        try {
            System.out.println("Starting JSON HTTP Server on port 8080...");
            server.start();
        } catch (IOException e) {
            System.err.println("Failed to start server: " + e.getMessage());
        }
    }
}
```

Testing the Custom HTTP Server

bash

```bash
# Test GET all users
curl -X GET http://localhost:8080/api/users

# Test POST create user
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name":"John Doe","email":"john@example.com"}'

# Test GET user by ID
curl -X GET http://localhost:8080/api/users/1

# Test PUT update user
curl -X PUT http://localhost:8080/api/users/1 \
  -H "Content-Type: application/json" \
  -d '{"name":"Jane Doe","email":"jane@example.com"}'

# Test DELETE user
curl -X DELETE http://localhost:8080/api/users/1
```

Key Differences Summary

Spring Boot vs Custom HTTP Server

Spring Boot Advantages:

- Auto-configuration and dependency injection
- Built-in security, validation, and error handling
- Database integration with JPA/Hibernate
- Production-ready features (actuator, monitoring)
- Extensive ecosystem and community support

Custom HTTP Server Advantages:

- Full control over implementation
- Lightweight and minimal dependencies
- Educational value - understanding HTTP protocol
- Custom routing and middleware logic
- Specific performance optimizations

Both approaches serve different purposes: Spring Boot for rapid enterprise development, and custom servers for learning, specific requirements, or minimal resource usage scenarios.