**Spring Boot Project Setup Guide**

**Overview**

This guide walks through creating a basic Spring Boot web application from scratch using IntelliJ IDEA. The resulting project provides a foundation for building REST APIs with security, documentation, and pagination support.

**Prerequisites**

**Required Software**

* **IntelliJ IDEA** (Ultimate or Community Edition)
* **Java Development Kit (JDK) 17 or higher**
* **Internet connection** (for downloading dependencies)

**Verification Steps**

1. Open IntelliJ IDEA → **File → Project Structure → SDKs**
2. Verify JDK 17+ is available and selected
3. Note: Maven installation is not required (project includes Maven Wrapper)

**Project Creation**

**Step 1: Initialize Spring Boot Project**

**In IntelliJ IDEA:**

1. **File → New → Project**
2. Select **Spring Boot** from the left panel
3. Configure project settings:
   * **Type:** Maven
   * **Group:** com.example
   * **Artifact:** library-management-system
   * **Package name:** com.example.librarymanagementsystem
   * **JDK:** 17 (or higher)
   * **Spring Boot Version:** 3.x (latest stable)
   * **Packaging:** Jar

**Step 2: Select Dependencies**

**Required Dependencies:**

* **Web → Spring Web** (REST API support)
* **Security → Spring Security** (Authentication/authorization)
* **I/O → Validation** (Input validation)
* **Developer Tools → Spring Boot DevTools** (Hot reload during development)

**Click Create** to generate the project

**Essential Configuration**

**Step 3: Add API Documentation Support**

**Add to pom.xml** (inside <dependencies> section):

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.6.0</version>

</dependency>

This enables Swagger UI for interactive API testing at /swagger-ui.html

**Step 4: Configure Application Properties**

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

# Server Configuration

server.port=8085

# Session Management

server.servlet.session.timeout=30m

# Application Info

spring.application.name=Library Management System

# Logging

logging.level.org.springframework.security=INFO

logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %msg%n

# Swagger/OpenAPI

springdoc.api-docs.path=/v3/api-docs

springdoc.swagger-ui.path=/swagger-ui.html

**Core Application Structure**

**Step 5: Main Application Class**

**File:** src/main/java/com/example/librarymanagementsystem/Application.java

package com.example.librarymanagementsystem;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

/\*\*

\* Spring Boot Application Entry Point

\*

\* @SpringBootApplication combines:

\* - @Configuration: Marks this as a configuration class

\* - @EnableAutoConfiguration: Enables Spring Boot's auto-configuration

\* - @ComponentScan: Scans this package and subpackages for components

\*/

@SpringBootApplication

public class Application implements CommandLineRunner {

public static void main(String[] args) {

// Starts Spring Boot application context and embedded server

SpringApplication.run(Application.class, args);

}

/\*\*

\* CommandLineRunner allows execution of code after application startup

\* Useful for initialization, demos, or one-time setup tasks

\*/

@Override

public void run(String... args) {

System.out.println("Application started successfully!");

System.out.println("Access Swagger UI at: http://localhost:8085/swagger-ui.html");

System.out.println("Health check at: http://localhost:8085/health");

}

}

**Step 6: Basic Security Configuration**

**File:** src/main/java/com/example/librarymanagementsystem/config/SecurityConfig.java

package com.example.librarymanagementsystem.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;

/\*\*

\* Spring Security Configuration

\* Defines authentication and authorization rules for the application

\*/

@Configuration

public class SecurityConfig {

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

return http

// Disable CSRF for JSON API development

.csrf(csrf -> csrf.disable())

// Define URL access rules

.authorizeHttpRequests(auth -> auth

// Public endpoints (no authentication required)

.requestMatchers("/swagger-ui/\*\*", "/swagger-ui.html", "/v3/api-docs/\*\*").permitAll()

.requestMatchers("/auth/\*\*", "/health").permitAll()

// Protected endpoints (authentication required)

.requestMatchers("/admin/\*\*").hasRole("ADMIN")

.requestMatchers("/users/\*\*").hasAnyRole("USER", "ADMIN")

.requestMatchers("/books/\*\*").hasAnyRole("USER", "ADMIN")

// All other endpoints require authentication

.anyRequest().authenticated()

)

.build();

}

/\*\*

\* Password encoder for secure password hashing

\* BCrypt with strength 10 provides good security/performance balance

\*/

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder(10);

}

}

**Step 7: Health Check Endpoint**

**File:** src/main/java/com/example/librarymanagementsystem/controller/HealthController.java

package com.example.librarymanagementsystem.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDateTime;

import java.util.Map;

/\*\*

\* Health check endpoint for monitoring application status

\*/

@RestController

public class HealthController {

@GetMapping("/health")

public Map<String, Object> health() {

return Map.of(

"status", "UP",

"timestamp", LocalDateTime.now(),

"application", "Library Management System"

);

}

}

**Step 8: Pagination Response Template**

**File:** src/main/java/com/example/librarymanagementsystem/dto/PagedResponse.java

package com.example.librarymanagementsystem.dto;

import java.util.List;

/\*\*

\* Standard pagination wrapper for all list endpoints

\* Provides consistent response format across the API

\*/

public class PagedResponse<T> {

private final List<T> content;

private final int page;

private final int size;

private final long total;

private final int totalPages;

private final boolean hasNext;

private final boolean hasPrevious;

public PagedResponse(List<T> content, int page, int size, long total) {

this.content = content;

this.page = page;

this.size = size;

this.total = total;

this.totalPages = (int) Math.ceil((double) total / size);

this.hasNext = page < totalPages - 1;

this.hasPrevious = page > 0;

}

// Getters

public List<T> getContent() { return content; }

public int getPage() { return page; }

public int getSize() { return size; }

public long getTotal() { return total; }

public int getTotalPages() { return totalPages; }

public boolean isHasNext() { return hasNext; }

public boolean isHasPrevious() { return hasPrevious; }

// Default pagination constants

public static final int DEFAULT\_PAGE = 0;

public static final int DEFAULT\_SIZE = 20;

public static final int MAX\_SIZE = 100;

}

**Running the Application**

**Method 1: IntelliJ IDEA**

1. Open Application.java
2. Click the green arrow next to the main method
3. Or right-click the file → **Run 'Application.main()'**

**Method 2: Command Line**

**Using Maven Wrapper (recommended):**

# Windows

.\mvnw spring-boot:run

# macOS/Linux

./mvnw spring-boot:run

**Using packaged JAR:**

# Build the application

./mvnw clean package -DskipTests

# Run the JAR file

java -jar target/library-management-system-\*.jar

**Verification Steps**

**Confirm Application Startup**

**Expected console output:**

Started Application in X.XXX seconds

Application started successfully!

Access Swagger UI at: http://localhost:8085/swagger-ui.html

Health check at: http://localhost:8085/health

**Test Endpoints**

1. **Health Check:** http://localhost:8085/health
   * Should return: {"status":"UP","timestamp":"...","application":"Library Management System"}
2. **Swagger UI:** http://localhost:8085/swagger-ui.html
   * Should display interactive API documentation interface
3. **API Documentation:** http://localhost:8085/v3/api-docs
   * Should return OpenAPI specification in JSON format

**Project Structure**

**After setup completion:**

library-management-system/

├── src/

│ └── main/

│ ├── java/com/example/librarymanagementsystem/

│ │ ├── Application.java

│ │ ├── config/

│ │ │ └── SecurityConfig.java

│ │ ├── controller/

│ │ │ └── HealthController.java

│ │ └── dto/

│ │ └── PagedResponse.java

│ └── resources/

│ └── application.properties

├── pom.xml

├── mvnw

└── mvnw.cmd

**Common Issues & Solutions**

**Port Conflicts**

**Problem:** Port 8085 is already in use **Solution:** Change port in application.properties:

server.port=8086

**Dependency Resolution**

**Problem:** Cannot resolve dependencies **Solution:**

1. Check internet connection
2. **File → Invalidate Caches and Restart**
3. Run ./mvnw clean install

**Swagger Not Loading**

**Problem:** 404 error on /swagger-ui.html **Solutions:**

1. Verify springdoc dependency in pom.xml
2. Check SecurityConfig permits swagger URLs
3. Restart application

**Multiple Main Classes**

**Problem:** Multiple @SpringBootApplication found **Solution:** Keep only one main class with @SpringBootApplication annotation

**Next Steps**

This foundation provides:

* ✅ Spring Boot web application
* ✅ Security configuration framework
* ✅ API documentation with Swagger
* ✅ Health monitoring endpoint
* ✅ Pagination response template

**Ready for implementation of:**

* Entity models (User, Book, Rental)
* Repository layer (data access)
* Service layer (business logic)
* Controller endpoints (REST API)
* Authentication system

The application is now ready for feature development following your specific business requirements and technical design documents.