# **RESTful Web Service Implementation with Spring Boot**

## **Project Structure**

src/

├── main/

│ ├── java/

│ │ └── com/

│ │ └── cognizant/

│ │ └── springlearn/

│ │ ├── SpringLearnApplication.java

│ │ ├── controller/

│ │ │ ├── CountryController.java

│ │ │ └── EmployeeController.java

│ │ ├── service/

│ │ │ └── EmployeeService.java

│ │ ├── dao/

│ │ │ └── EmployeeDao.java

│ │ ├── model/

│ │ │ ├── Country.java

│ │ │ ├── Employee.java

│ │ │ ├── Department.java

│ │ │ └── Skill.java

│ │ ├── exception/

│ │ │ ├── EmployeeNotFoundException.java

│ │ │ └── GlobalExceptionHandler.java

│ │ └── config/

│ └── resources/

│ └── application.yml

└── test/

└── java/

└── com/

└── cognizant/

└── springlearn/

└── controller/

├── CountryControllerTest.java

└── EmployeeControllerTest.java

## **Dependencies (pom.xml)**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

## **Model Classes**

### **Country.java**

package com.cognizant.springlearn.model;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

public class Country {

@NotNull(message = "Country code cannot be null")

@Size(min = 2, max = 2, message = "Country code should be 2 characters")

private String code;

private String name;

// Constructors

public Country() {}

public Country(String *code*, String *name*) {

this.code = code;

this.name = name;

}

// Getters and Setters

public String getCode() {

return code;

}

public void setCode(String *code*) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String *name*) {

this.name = name;

}

@Override

public String toString() {

return "Country{" +

"code='" + code + '\'' +

", name='" + name + '\'' +

'}';

}

}

### **Employee.java**

package com.cognizant.springlearn.model;

import com.fasterxml.jackson.annotation.JsonFormat;

import javax.validation.Valid;

import javax.validation.constraints.\*;

import java.time.LocalDate;

import java.util.List;

public class Employee {

@NotNull(message = "Employee ID cannot be null")

@Min(value = 1, message = "Employee ID must be a positive number")

private Integer id;

@NotNull(message = "Employee name cannot be null")

@NotBlank(message = "Employee name cannot be blank")

@Size(min = 1, max = 30, message = "Employee name must be between 1 and 30 characters")

private String name;

@NotNull(message = "Salary cannot be null")

@Min(value = 0, message = "Salary must be zero or above")

private Double salary;

@NotNull(message = "Permanent status cannot be null")

private Boolean permanent;

@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "dd/MM/yyyy")

private LocalDate dateOfBirth;

@Valid

private Department department;

@Valid

private List<Skill> skills;

// Constructors

public Employee() {}

public Employee(Integer *id*, String *name*, Double *salary*, Boolean *permanent*,

LocalDate *dateOfBirth*, Department *department*, List<Skill> *skills*) {

this.id = id;

this.name = name;

this.salary = salary;

this.permanent = permanent;

this.dateOfBirth = dateOfBirth;

this.department = department;

this.skills = skills;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer *id*) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String *name*) {

this.name = name;

}

public Double getSalary() {

return salary;

}

public void setSalary(Double *salary*) {

this.salary = salary;

}

public Boolean getPermanent() {

return permanent;

}

public void setPermanent(Boolean *permanent*) {

this.permanent = permanent;

}

public LocalDate getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(LocalDate *dateOfBirth*) {

this.dateOfBirth = dateOfBirth;

}

public Department getDepartment() {

return department;

}

public void setDepartment(Department *department*) {

this.department = department;

}

public List<Skill> getSkills() {

return skills;

}

public void setSkills(List<Skill> *skills*) {

this.skills = skills;

}

}

### **Department.java**

package com.cognizant.springlearn.model;

import javax.validation.constraints.\*;

public class Department {

@NotNull(message = "Department ID cannot be null")

@Min(value = 1, message = "Department ID must be a positive number")

private Integer id;

@NotNull(message = "Department name cannot be null")

@NotBlank(message = "Department name cannot be blank")

@Size(min = 1, max = 30, message = "Department name must be between 1 and 30 characters")

private String name;

// Constructors

public Department() {}

public Department(Integer *id*, String *name*) {

this.id = id;

this.name = name;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer *id*) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String *name*) {

this.name = name;

}

}

### **Skill.java**

package com.cognizant.springlearn.model;

import javax.validation.constraints.\*;

public class Skill {

@NotNull(message = "Skill ID cannot be null")

@Min(value = 1, message = "Skill ID must be a positive number")

private Integer id;

@NotNull(message = "Skill name cannot be null")

@NotBlank(message = "Skill name cannot be blank")

@Size(min = 1, max = 30, message = "Skill name must be between 1 and 30 characters")

private String name;

// Constructors

public Skill() {}

public Skill(Integer *id*, String *name*) {

this.id = id;

this.name = name;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer *id*) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String *name*) {

this.name = name;

}

}

## **Exception Classes**

**EmployeeNotFoundException.java**

package com.cognizant.springlearn.exception;

import org.springframework.http.HttpStatus;

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

@ResponseStatus(HttpStatus.NOT\_FOUND)

public class EmployeeNotFoundException extends Exception {

private static final long serialVersionUID = 1L;

public EmployeeNotFoundException(String *message*) {

super(message);

}

public EmployeeNotFoundException(String *message*, Throwable *cause*) {

super(message, cause);

}

}

### 

### **GlobalExceptionHandler.java**

package com.cognizant.springlearn.exception;

import com.fasterxml.jackson.databind.exc.InvalidFormatException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.http.converter.HttpMessageNotReadableException;

import org.springframework.web.bind.MethodArgumentNotValidException;

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

import org.springframework.web.context.request.WebRequest;

import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

import java.util.\*;

import java.util.stream.Collectors;

@ControllerAdvice

public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

private static final Logger LOGGER = LoggerFactory.getLogger(GlobalExceptionHandler.class);

@Override

protected ResponseEntity<Object> handleMethodArgumentNotValid(

MethodArgumentNotValidException *ex*,

HttpHeaders *headers*,

HttpStatus *status*,

WebRequest *request*) {

LOGGER.info("Start - Validation error handling");

// Map that contains the error details

Map<String, Object> body = new LinkedHashMap<>();

body.put("timestamp", new Date());

body.put("status", status.value());

// Get all validation errors

List<String> errors = ex.getBindingResult()

.getFieldErrors()

.stream()

.map(x -> x.getDefaultMessage())

.collect(Collectors.toList());

// Add errors to the response map

body.put("errors", errors);

LOGGER.info("End - Validation error handling");

return new ResponseEntity<>(body, headers, status);

}

@Override

protected ResponseEntity<Object> handleHttpMessageNotReadable(

HttpMessageNotReadableException *ex*,

HttpHeaders *headers*,

HttpStatus *status*,

WebRequest *request*) {

LOGGER.info("Start - Message not readable error handling");

Map<String, Object> body = new LinkedHashMap<>();

body.put("timestamp", new Date());

body.put("status", status.value());

body.put("error", "Bad Request");

if (ex.getCause() instanceof InvalidFormatException) {

final Throwable cause = ex.getCause() == null ? ex : ex.getCause();

for (InvalidFormatException.Reference reference :

((InvalidFormatException) cause).getPath()) {

body.put("message", "Incorrect format for field '" +

reference.getFieldName() + "'");

}

}

LOGGER.info("End - Message not readable error handling");

return new ResponseEntity<>(body, headers, status);

}

}

## **Data Access Layer**

EmployeeDao.java

package com.cognizant.springlearn.dao;

import com.cognizant.springlearn.exception.EmployeeNotFoundException;

import com.cognizant.springlearn.model.Department;

import com.cognizant.springlearn.model.Employee;

import com.cognizant.springlearn.model.Skill;

import org.springframework.stereotype.Repository;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

@Repository

public class EmployeeDao {

private static List<Employee> employees = new ArrayList<>();

static {

// Initialize with sample data

Department dept1 = new Department(1, "IT");

Department dept2 = new Department(2, "HR");

List<Skill> skills1 = Arrays.asList(

new Skill(1, "Java"),

new Skill(2, "Spring Boot")

);

List<Skill> skills2 = Arrays.asList(

new Skill(3, "Recruitment"),

new Skill(4, "Training")

);

employees.add(new Employee(1, "John Doe", 75000.0, true,

LocalDate.of(1990, 5, 15), dept1, skills1));

employees.add(new Employee(2, "Jane Smith", 65000.0, true,

LocalDate.of(1985, 8, 20), dept2, skills2));

}

public List<Employee> getAllEmployees() {

return new ArrayList<>(employees);

}

public Optional<Employee> getEmployeeById(Integer *id*) {

return employees.stream()

.filter(emp -> emp.getId().equals(id))

.findFirst();

}

public Employee addEmployee(Employee *employee*) {

employees.add(employee);

return employee;

}

public Employee updateEmployee(Employee *employee*) throws EmployeeNotFoundException {

Optional<Employee> existingEmployee = getEmployeeById(employee.getId());

if (existingEmployee.isPresent()) {

int index = employees.indexOf(existingEmployee.get());

employees.set(index, employee);

return employee;

} else {

throw new EmployeeNotFoundException("Employee with ID " + employee.getId() + " not found");

}

}

public void deleteEmployee(Integer *id*) throws EmployeeNotFoundException {

Optional<Employee> employee = getEmployeeById(id);

if (employee.isPresent()) {

employees.remove(employee.get());

} else {

throw new EmployeeNotFoundException("Employee with ID " + id + " not found");

}

}

}

## **Service Layer**

### **EmployeeService.java**

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.dao.EmployeeDao;

import com.cognizant.springlearn.exception.EmployeeNotFoundException;

import com.cognizant.springlearn.model.Employee;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class EmployeeService {

@Autowired

private EmployeeDao employeeDao;

public List<Employee> getAllEmployees() {

return employeeDao.getAllEmployees();

}

public Optional<Employee> getEmployeeById(Integer *id*) {

return employeeDao.getEmployeeById(id);

}

public Employee addEmployee(Employee *employee*) {

return employeeDao.addEmployee(employee);

}

public Employee updateEmployee(Employee *employee*) throws EmployeeNotFoundException {

return employeeDao.updateEmployee(employee);

}

public void deleteEmployee(Integer *id*) throws EmployeeNotFoundException {

employeeDao.deleteEmployee(id);

}

}

## **Controller Layer**

### **CountryController.java**

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.model.Country;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.\*;

import javax.validation.Valid;

@RestController

@RequestMapping("/countries")

public class CountryController {

private static final Logger LOGGER = LoggerFactory.getLogger(CountryController.class);

@PostMapping

public Country addCountry(@RequestBody @Valid Country *country*) {

LOGGER.info("Start - Adding country: {}", country);

LOGGER.info("Country details: {}", country);

LOGGER.info("End - Country added successfully");

return country;

}

@GetMapping

public String getAllCountries() {

LOGGER.info("Getting all countries");

return "List of all countries";

}

@GetMapping("/{code}")

public String getCountryByCode(@PathVariable String *code*) {

LOGGER.info("Getting country with code: {}", code);

return "Country with code: " + code;

}

@PutMapping

public String updateCountry(@RequestBody @Valid Country *country*) {

LOGGER.info("Updating country: {}", country);

return "Country updated successfully";

}

@DeleteMapping("/{code}")

public String deleteCountry(@PathVariable String *code*) {

LOGGER.info("Deleting country with code: {}", code);

return "Country deleted successfully";

}

}

### **EmployeeController.java**

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.exception.EmployeeNotFoundException;

import com.cognizant.springlearn.model.Employee;

import com.cognizant.springlearn.service.EmployeeService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import javax.validation.Valid;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

private static final Logger LOGGER = LoggerFactory.getLogger(EmployeeController.class);

@Autowired

private EmployeeService employeeService;

@GetMapping

public ResponseEntity<List<Employee>> getAllEmployees() {

LOGGER.info("Start - Getting all employees");

List<Employee> employees = employeeService.getAllEmployees();

LOGGER.info("End - Retrieved {} employees", employees.size());

return ResponseEntity.ok(employees);

}

@GetMapping("/{id}")

public ResponseEntity<Employee> getEmployeeById(@PathVariable Integer *id*) {

LOGGER.info("Start - Getting employee with ID: {}", id);

Optional<Employee> employee = employeeService.getEmployeeById(id);

if (employee.isPresent()) {

LOGGER.info("End - Employee found: {}", employee.get().getName());

return ResponseEntity.ok(employee.get());

} else {

LOGGER.warn("Employee with ID {} not found", id);

return ResponseEntity.notFound().build();

}

}

@PostMapping

public ResponseEntity<Employee> addEmployee(@RequestBody @Valid Employee *employee*) {

LOGGER.info("Start - Adding employee: {}", employee.getName());

Employee savedEmployee = employeeService.addEmployee(employee);

LOGGER.info("End - Employee added successfully with ID: {}", savedEmployee.getId());

return ResponseEntity.status(HttpStatus.CREATED).body(savedEmployee);

}

@PutMapping

public ResponseEntity<Employee> updateEmployee(@RequestBody @Valid Employee *employee*)

throws EmployeeNotFoundException {

LOGGER.info("Start - Updating employee with ID: {}", employee.getId());

Employee updatedEmployee = employeeService.updateEmployee(employee);

LOGGER.info("End - Employee updated successfully: {}", updatedEmployee.getName());

return ResponseEntity.ok(updatedEmployee);

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEmployee(@PathVariable Integer *id*)

throws EmployeeNotFoundException {

LOGGER.info("Start - Deleting employee with ID: {}", id);

employeeService.deleteEmployee(id);

LOGGER.info("End - Employee deleted successfully");

return ResponseEntity.noContent().build();

}

}

## **Configuration**

### **application.yml**

server:

port: 8090

logging:

level:

com.cognizant.springlearn: INFO

pattern:

console: "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"

## **Main Application Class**

### **SpringLearnApplication.java**

package com.cognizant.springlearn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] *args*) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

## 

## 

## 

## **Test Classes**

### **CountryControllerTest.java**

package com.cognizant.springlearn.controller;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.cognizant.springlearn.model.Country;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(CountryController.class)

public class CountryControllerTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private ObjectMapper objectMapper;

@Test

public void testAddCountrySuccess() throws Exception {

Country country = new Country("IN", "India");

mockMvc.perform(post("/countries")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(country)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.code").value("IN"))

.andExpect(jsonPath("$.name").value("India"));

}

@Test

public void testAddCountryValidationError() throws Exception {

Country country = new Country("I", "India"); // Invalid code length

mockMvc.perform(post("/countries")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(country)))

.andExpect(status().isBadRequest())

.andExpect(jsonPath("$.errors[0]").value("Country code should be 2 characters"));

}

}

### 

### **EmployeeControllerTest.java**

package com.cognizant.springlearn.controller;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;

import com.cognizant.springlearn.exception.EmployeeNotFoundException;

import com.cognizant.springlearn.model.Department;

import com.cognizant.springlearn.model.Employee;

import com.cognizant.springlearn.model.Skill;

import com.cognizant.springlearn.service.EmployeeService;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import java.time.LocalDate;

import java.util.Arrays;

import java.util.List;

import static org.mockito.ArgumentMatchers.any;

import static org.mockito.ArgumentMatchers.anyInt;

import static org.mockito.Mockito.\*;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(EmployeeController.class)

public class EmployeeControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private EmployeeService employeeService;

private ObjectMapper objectMapper;

private Employee testEmployee;

@BeforeEach

public void setUp() {

objectMapper = new ObjectMapper();

objectMapper.registerModule(new JavaTimeModule());

Department department = new Department(1, "IT");

List<Skill> skills = Arrays.asList(new Skill(1, "Java"));

testEmployee = new Employee(1, "John Doe", 75000.0, true,

LocalDate.of(1990, 5, 15), department, skills);

}

@Test

public void testUpdateEmployeeSuccess() throws Exception {

when(employeeService.updateEmployee(any(Employee.class))).thenReturn(testEmployee);

mockMvc.perform(put("/employees")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(testEmployee)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(1))

.andExpect(jsonPath("$.name").value("John Doe"));

}

@Test

public void testUpdateEmployeeNotFound() throws Exception {

when(employeeService.updateEmployee(any(Employee.class)))

.thenThrow(new EmployeeNotFoundException("Employee not found"));

mockMvc.perform(put("/employees")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(testEmployee)))

.andExpect(status().isNotFound());

}

@Test

public void testDeleteEmployeeSuccess() throws Exception {

doNothing().when(employeeService).deleteEmployee(anyInt());

mockMvc.perform(delete("/employees/1"))

.andExpect(status().isNoContent());

}

@Test

public void testDeleteEmployeeNotFound() throws Exception {

doThrow(new EmployeeNotFoundException("Employee not found"))

.when(employeeService).deleteEmployee(anyInt());

mockMvc.perform(delete("/employees/1"))

.andExpect(status().isNotFound());

}

@Test

public void testAddEmployeeValidationError() throws Exception {

// Create employee with invalid data

Employee invalidEmployee = new Employee();

invalidEmployee.setId(null); // Invalid - null ID

invalidEmployee.setName(""); // Invalid - blank name

invalidEmployee.setSalary(-1000.0); // Invalid - negative salary

mockMvc.perform(post("/employees")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(invalidEmployee)))

.andExpected(status().isBadRequest());

}

}

## **API Testing with cURL Commands**

### **Country API Testing**

# Test POST with valid data

curl -i -H 'Content-Type: application/json' -X POST -s -d '{"code":"IN","name":"India"}' http://localhost:8090/countries

# Test POST with invalid data (validation error)

curl -i -H 'Content-Type: application/json' -X POST -s -d '{"code":"I","name":"India"}' http://localhost:8090/countries

# Test GET all countries

curl -i -X GET http://localhost:8090/countries

# Test GET specific country

curl -i -X GET http://localhost:8090/countries/IN

# Test PUT (update)

curl -i -H 'Content-Type: application/json' -X PUT -s -d '{"code":"US","name":"United States"}' http://localhost:8090/countries

# Test DELETE

curl -i -X DELETE http://localhost:8090/countries/IN

### **Employee API Testing**

# Test GET all employees

curl -i -X GET http://localhost:8090/employees

# Test GET specific employee

curl -i -X GET http://localhost:8090/employees/1

# Test POST (create employee)

curl -i -H 'Content-Type: application/json' -X POST -s -d '{

"id": 3,

"name": "Alice Johnson",

"salary": 80000.0,

"permanent": true,

"dateOfBirth": "15/03/1988",

"department": {

"id": 1,

"name": "IT"

},

"skills": [

{

"id": 1,

"name": "React"

},

{

"id": 2,

"name": "Node.js"

}

]

}' http://localhost:8090/employees

# Test PUT (update employee)

curl -i -H 'Content-Type: application/json' -X PUT -s -d '{

"id": 1,

"name": "John Doe Updated",

"salary": 85000.0,

"permanent": true,

"dateOfBirth": "15/05/1990",

"department": {

"id": 1,

"name": "IT"

},

"skills": [

{

"id": 1,

"name": "Java"

},

{

"id": 2,

"name": "Spring Boot"

}

]

}' http://localhost:8090/employees

# Test DELETE

curl -i -X DELETE http://localhost:8090/employees/1

## **Key Features Implemented**

1. **RESTful URL Design**: Following REST conventions with proper HTTP methods and URL patterns
2. **Input Validation**: Comprehensive validation using Bean Validation annotations
3. **Global Exception Handling**: Centralized error handling with meaningful error messages
4. **Layered Architecture**: Proper separation of concerns with Controller, Service, and DAO layers
5. **Logging**: Structured logging for debugging and monitoring
6. **Testing**: Unit tests for controllers with MockMvc
7. **JSON Serialization**: Proper JSON handling with Jackson annotations
8. **Error Responses**: Standardized error response format

## 

## 

## **Best Practices Followed**

* **Single Responsibility Principle**: Each class has a clear, single purpose
* **Dependency Injection**: Using Spring's @Autowired for loose coupling
* **Exception Handling**: Custom exceptions with appropriate HTTP status codes
* **Validation**: Server-side validation with meaningful error messages
* **Logging**: Comprehensive logging for debugging and monitoring
* **Testing**: Comprehensive test coverage for critical paths
* **Configuration**: Externalized configuration using application.yml
* **Code Quality**: Clean, readable code with proper naming conventions