Laboratory Work 1

Spring Boot Project Setup and Configuration

Web Component Development (Java EE)
Week 1

Textbook: Pro Spring Boot 3 with Kotlin, 3rd Edition, Chapter 1

Institution: IITU

Department: Information Systems

Total Points: 100 (Lab 1.1: 50 + Lab 1.2: 50)

Developed by Syrym Zhakypbekov IITU

2026

Contents

Requirements

Requirements

- All code must be written manually (no code generation tools)
- Use camelCase for variables and methods
- Add comments explaining your code
- Follow Git discipline: proper folder structure, conventional commits
- Minimum 3 meaningful commits per lab

Lab 1.1: Creating Spring Boot Project with Spring Initializr (50 Points)

Objective

Objective

Create a Spring Boot project using Spring Initializr, understand project structure, configure build files, and set up the development environment.

Background

Spring Initializr is the official tool for generating Spring Boot projects. It creates a project skeleton with proper structure, dependencies, and configuration files. Understanding how to use Spring Initializr and the generated project structure is essential for Spring Boot development.

Vocabulary

Vocabulary

- **Spring Initializr:** Web-based project generator (`start.spring.io`)
- **Build Tool:** Maven or Gradle for dependency management and building
- **Starter:** Pre-configured dependency sets (e.g., `spring-boot-starter-web`)
- **Auto-configuration:** Automatic setup based on classpath dependencies
- **Embedded Server:** Web server (Tomcat) bundled with application
- **application.properties:** Configuration file for application settings

Task 1: Generate Project from Spring Initializr (20 points)

Task

Steps:

1. Navigate to `https://start.spring.io`
2. Configure project settings:
   - Project: Gradle - Groovy (or Maven if preferred)
   - Language: Java
   - Spring Boot: 3.2.x (or latest stable)
   - Project Metadata:
     - Group: `com.iitu.webdev`
     - Artifact: `student-info-app`
     - Name: Student Info App
     - Package name: `com.iitu.webdev.studentinfoapp`
     - Packaging: JAR
     - Java: 17 or 21
3. Add Dependencies:
   - Spring Web (for web application support)
   - Spring Boot DevTools (for development convenience)
4. Generate and download the project
5. Extract the ZIP file to your workspace

Deliverable

- Screenshot of Spring Initializr configuration page
- Extracted project folder
- Brief description of what each dependency provides

Task 2: Import and Explore Project Structure (15 points)

Task

Steps:

1. Import project into IDE (IntelliJ IDEA, Eclipse, or VS Code)
2. Wait for build tool sync to complete
3. Examine the project structure:
   - `src/main/java`: Application source code
   - `src/main/resources`: Configuration and static resources
   - `src/test/java`: Test code
   - `build.gradle` (or `pom.xml`): Build configuration
4. Open and read these files:
   - Main application class (`StudentInfoAppApplication.java`)
   - `application.properties`
   - `build.gradle` (or `pom.xml`)
5. Identify:
   - What annotation is on the main class?
   - What is the purpose of the `main()` method?
   - What dependencies are declared in `build.gradle/pom.xml`?

Deliverable

- Screenshot of project structure in IDE
- Answers to the three questions above
- Brief explanation of each directory's purpose

Task 3: Configure Application Properties (15 points)

Task

Steps:

1. Open `application.properties` in `src/main/resources/`
2. Add the following configurations:

```
# Application name
spring.application.name=Student Info App

# Server configuration
server.port=8080
server.servlet.context-path=/api

# Logging
logging.level.root=INFO
logging.level.com.iitu.webdev=DEBUG
```

3. Explain what each configuration does:
   - `spring.application.name`
   - `server.port`
   - `server.servlet.context-path`
   - `logging.level`
4. Test the configuration by running the application

Deliverable

- `application.properties` file with configurations
- Explanation of each configuration property (2–3 sentences each)
- Screenshot showing application starts successfully

Grading Rubric

Component	Points
Task 1: Project generation	20
Task 2: Project exploration	15
Task 3: Configuration	15
Code quality and comments	5
Git commits	5
Total	**50**

Lab 1.2: Building Your First REST Controller (50 Points)

Objective

Objective

Create a REST controller with multiple endpoints, understand Spring MVC annotations, and test the REST API using browser and command-line tools.

Background

REST controllers are the entry points for handling HTTP requests in Spring Boot applications. They use annotations to map URLs to methods and return data in various formats (JSON, XML, plain text). This lab introduces basic REST API development.

Vocabulary

Vocabulary

- **@RestController:** Annotation marking class as REST controller
- **@RequestMapping:** Maps URLs to controller methods
- **@GetMapping:** Maps HTTP GET requests
- **@PostMapping:** Maps HTTP POST requests
- **Endpoint:** URL path that triggers a method
- **HTTP Method:** GET, POST, PUT, DELETE, etc.

Task 1: Create Student Model Class (10 points)

Task

Steps:

1. Create package: `com.iitu.webdev.studentinfoapp.model`
2. Create `Student.java` class:

```java
package com.iitu.webdev.studentinfoapp.model;

public class Student {
    private String id;
    private String name;
    private String email;
    private String major;

    // Constructors
    public Student() {}

    public Student(String id, String name, String email, String
        major) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.major = major;
    }

    // Getters and Setters
    public String getId() { return id; }
    public void setId(String 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 String getMajor() { return major; }
    public void setMajor(String major) { this.major = major; }
}
```

Deliverable

- `Student.java` source code
- Brief explanation of why we need getters and setters

Task 2: Create REST Controller (25 points)

Task

Steps:

1. Create package: `com.iitu.webdev.studentinfoapp.controller`
2. Create `StudentController.java`:

```java
package com.iitu.webdev.studentinfoapp.controller;

import com.iitu.webdev.studentinfoapp.model.Student;
import org.springframework.web.bind.annotation.*;
import java.util.*;

@RestController
@RequestMapping("/students")
public class StudentController {

    // In-memory storage (will be replaced with database later)
    private Map<String, Student> students = new HashMap<>();

    // Initialize with sample data
    public StudentController() {
        students.put("S001", new Student("S001", "Alice Johnson",
                        "alice@iitu.edu.kz", "Computer Science"));
        students.put("S002", new Student("S002", "Bob Smith",
                        "bob@iitu.edu.kz", "Software Engineering")
                            );
    }

    // GET /students - Get all students
    @GetMapping
    public List<Student> getAllStudents() {
        return new ArrayList<>(students.values());
    }

    // GET /students/{id} - Get student by ID
    @GetMapping("/{id}")
    public Student getStudentById(@PathVariable String id) {
        return students.get(id);
    }

    // POST /students - Create new student
    @PostMapping
    public Student createStudent(@RequestBody Student student) {
        students.put(student.getId(), student);
        return student;
    }

    // GET /students/health - Health check endpoint
    @GetMapping("/health")
    public String health() {
        return "Student API is running!";
    }
}
```

3. Add comments explaining:
   - What `@RestController` does
   - What `@RequestMapping("/students")` does
   - What `@GetMapping`, `@PostMapping` do
   - What `@PathVariable` and `@RequestBody` do

Deliverable

- `StudentController.java` with comments
- Explanation of each annotation (2–3 sentences each)

Task 3: Test REST API Endpoints (15 points)

Task

Steps:

1. Run the application:
   - Right-click `StudentInfoAppApplication` → Run
   - Or: `./gradlew bootRun` (Gradle) or `mvn spring-boot:run` (Maven)
2. Test endpoints using browser:
   - `http://localhost:8080/api/students/health`
   - `http://localhost:8080/api/students`
   - `http://localhost:8080/api/students/S001`
3. Test POST endpoint using curl:

```
curl -X POST http://localhost:8080/api/students \
  -H "Content-Type: application/json" \
  -d '{"id":"S003","name":"Charlie Brown","email":"charlie@iitu.
     edu.kz","major":"Data Science"}'
```

4. Verify new student was added:
   - `http://localhost:8080/api/students`

Deliverable

- Screenshots of browser showing each GET endpoint response
- Screenshot of curl command and response
- Screenshot showing all students including the new one

Grading Rubric

Component	Points
Task 1: Student model class	10
Task 2: REST controller implementation	25
Task 3: API testing	15
Code quality, comments, structure	5
Git commits	5
Total	**50**

Submission Instructions

1. Create folder structure:

```
Lab_01/
  task1/
    [Spring Initializr project files]
  task2/
    [REST controller files]
```

2. Git commits:

```
git init
git add .
git commit -m "feat: add Lab 1.1 Spring Boot project setup"
git commit -m "feat: add Lab 1.2 REST controller implementation"
```

3. Create `.txt` file with GitHub repository link (if using GitHub)
4. Upload to learning management system

Additional Notes

- Remember: context-path is `/api`, so all URLs start with `/api`
- JSON responses are automatic with `@RestController`
- Use Postman or similar tool if curl is not available
- Check console for any error messages
- Verify all endpoints return expected responses