Laboratory Work 2

Building Users App - CRUD API Implementation

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 2.1: 50 + Lab 2.2: 50)

Bonus: 10 extra points available

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
- Implement exactly as described in the book (Chapter 1, Users App)

Lab 2.1: Implementing Users App - Part 1 (50 Points)

Objective

Objective

Build the Users App project as described in Chapter 1 of the textbook. Create a CRUD (Create, Read, Update, Delete) API for managing users with name and email. Use in-memory storage (Map) as specified in the book.

Background

The Users App is the first complete project in the textbook. It demonstrates core Spring Boot concepts: REST controllers, request mapping, HTTP methods, and JSON serialization. This lab implements the basic structure and GET operations.

Vocabulary

Vocabulary

- **CRUD:** Create, Read, Update, Delete operations
- **REST API:** Web service following REST principles
- **JSON:** JavaScript Object Notation, data interchange format
- **In-memory storage:** Data stored in application memory (not persistent)
- **HTTP Status Code:** Response code (200 OK, 404 Not Found, etc.)

Task 1: Create Users App Project (15 points)

Task

Steps:

1. Go to `https://start.spring.io`
2. Configure project (as in book):
    - Project: Gradle - Groovy
    - Language: Java
    - Spring Boot: 3.2.x
    - Group: `com.apress` (or `com.iitu.users`)
    - Artifact: `users`
    - Name: `users`
    - Package: `com.apress.users` (or `com.iitu.users`)
    - Packaging: JAR
    - Java: 17
3. Add Dependencies:
    - Spring Web
4. Generate, download, and import to IDE

Deliverable

- Project created and imported
- Screenshot of project structure

Task 2: Create User Model Class (10 points)

Task

Steps:

1. Create `User.java` in `src/main/java/com/apress/users/` (or your package):

```java
package com.apress.users;

public class User {
    private String email;
    private String name;

    // Default constructor
    public User() {}

    // Constructor with parameters
    public User(String email, String name) {
        this.email = email;
        this.name = name;
    }

    // Getters and Setters
    public String getEmail() {
        return email;
    }

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

    public String getName() {
        return name;
    }

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

2. Note: This matches the book's User class structure

Deliverable

- `User.java` source code
- Brief explanation of why email is used as key

Task 3: Create UsersController - GET Operations (25 points)

Task

Steps:

1. Create `UsersController.java` exactly as in the book:

```java
package com.apress.users;

import org.springframework.web.bind.annotation.*;
import java.util.*;

@RestController
@RequestMapping("/users")
public class UsersController {

    // In-memory storage using Map (email as key)
    private Map<String, User> users = new HashMap<>();

    // Initialize with sample data (as in book)
    public UsersController() {
        users.put("ximena@email.com",
                  new User("ximena@email.com", "Ximena"));
        users.put("norma@email.com",
                  new User("norma@email.com", "Norma"));
    }

    // GET /users - Get all users
    @GetMapping
    public Collection<User> getAllUsers() {
        return users.values();
    }

    // GET /users/{email} - Get user by email
    @GetMapping("/{email}")
    public User findUserByEmail(@PathVariable String email) {
        return users.get(email);
    }
}
```

2. Add comments explaining:
   - Why `Map<String, User>` is used
   - What `@GetMapping` does
   - What `@PathVariable` does
   - What `Collection<User>` returns

Deliverable

- `UsersController.java` with all GET methods
- Comments explaining key concepts
- Brief explanation of the API design

Grading Rubric

Component	Points
Task 1: Project setup	15
Task 2: User model	10
Task 3: GET operations	25
Code quality and comments	5
Git commits	5
Total	**50**

Lab 2.2: Completing Users App - POST and DELETE Operations (50 Points)

Objective

Objective

Complete the Users App by implementing POST (create) and DELETE operations. Test all CRUD operations and verify the API works as described in the book.

Background

A complete CRUD API requires all four operations. This lab completes the Users App by adding create and delete functionality. You'll also learn about HTTP status codes and proper API responses.

Vocabulary

Vocabulary

- **@PostMapping:** Annotation for HTTP POST requests
- **@DeleteMapping:** Annotation for HTTP DELETE requests
- **@RequestBody:** Annotation to bind request body to method parameter
- **HTTP 201 Created:** Status code for successful resource creation
- **HTTP 204 No Content:** Status code for successful deletion

Task 1: Implement POST Operation (20 points)

Task

Steps:

1. Add `save()` method to `UsersController`:

```
// POST /users - Create new user
@PostMapping
public User save(@RequestBody User user) {
    users.put(user.getEmail(), user);
    return user;
}
```

2. Add comments explaining:
   - What `@PostMapping` does
   - What `@RequestBody` does
   - Why we use `user.getEmail()` as the key
   - What happens if user with same email already exists
3. Test the endpoint:

```
curl -X POST http://localhost:8080/users \
  -H "Content-Type: application/json" \
  -d '{"email":"alice@email.com","name":"Alice"}'
```

Deliverable

- Updated `UsersController.java` with POST method
- Comments explaining POST operation
- Screenshot of curl command and response
- Screenshot showing new user in GET `/users` response

Task 2: Implement DELETE Operation (20 points)

Task

Steps:

1. Add `deleteByEmail()` method to `UsersController`:

```
// DELETE /users/{email} - Delete user by email
@DeleteMapping("/{email}")
public void deleteByEmail(@PathVariable String email) {
    users.remove(email);
}
```

2. Add comments explaining:
   - What `@DeleteMapping` does
   - Why method returns `void`
   - What happens if email doesn't exist
   - HTTP status code returned (default is 200 OK)
3. Test the endpoint:

```
curl -X DELETE http://localhost:8080/users/ximena@email.com
```

4. Verify deletion:

```
curl http://localhost:8080/users
```

Deliverable

- Updated `UsersController.java` with DELETE method
- Comments explaining DELETE operation
- Screenshot of DELETE request
- Screenshot showing user was removed from list

Task 3: Complete CRUD Testing (10 points)

Task

Steps:

1. Test all operations in sequence:
   a) GET all users (should show 2 initial users)
   b) POST new user (create `alice@email.com`)
   c) GET all users (should show 3 users)
   d) GET user by email (get `alice@email.com`)
   e) DELETE user (delete `norma@email.com`)
   f) GET all users (should show 2 users)
2. Document test results:
   - Screenshot of each operation
   - Brief summary of what each test verified

Deliverable

- Complete test sequence with screenshots
- Summary document explaining each test case
- Final `UsersController.java` with all CRUD operations

Grading Rubric

Component	Points
Task 1: POST operation	20
Task 2: DELETE operation	20
Task 3: Complete testing	10
Code quality, comments, structure	5
Git commits	5
Total	**50**

Bonus Task: Improve Users App (Optional - 10 Extra Points)

Bonus Task (Optional - 10 Extra Points)

Task 1: Add PUT Operation

Add PUT operation for updating users:

```
@PutMapping("/{email}")
public User updateUser(@PathVariable String email, @RequestBody User
    user) {
    if (users.containsKey(email)) {
        users.put(email, user);
        return user;
    }
    return null; // or throw exception
}
```

Task 2: Add Error Handling

- Return 404 if user not found in GET `/users/{email}`
- Return 400 if email missing in POST request
- Return appropriate status codes

Task 3: Add Validation

- Validate email format
- Validate name is not empty

Deliverable

- Updated controller with PUT method
- Error handling implementation
- Validation logic
- Test cases for error scenarios

Submission Instructions

1. Create folder structure:

```
Lab_02/
  users-app/
    [complete Spring Boot project]
  screenshots/
    [all test screenshots]
  README.txt
    [brief description of what was implemented]
```

2. Git commits:

```
git init
git add .
git commit -m "feat: add Users App project structure"
git commit -m "feat: implement User model and GET operations"
git commit -m "feat: implement POST and DELETE operations"
git commit -m "test: add CRUD operation tests"
```

3. Create `README.txt` with:
   - Project description
   - How to run the application
   - API endpoints documentation
   - Test instructions
4. Create `.txt` file with GitHub repository link (if using GitHub)
5. Upload to learning management system

Additional Notes

- Follow the book's implementation exactly
- Email is used as the unique identifier (key in Map)
- All operations work with in-memory storage (data lost on restart)
- JSON serialization is automatic with `@RestController`
- Use Postman, HTTPie, or curl for testing
- Check console for any errors or warnings

Reference: Book Implementation

This lab is based on Chapter 1, "Spring Boot Quick Start" from "Pro Spring Boot 3 with Kotlin, 3rd Edition". The Users App demonstrates:

1. Creating Spring Boot project with Spring Initializr
2. Building REST controller with `@RestController`
3. Implementing CRUD operations
4. Using in-memory storage (Map)
5. Testing REST API endpoints

Key Concepts from the Book:

- `@RestController` annotation
- `@RequestMapping` for base URL mapping
- HTTP method annotations (`@GetMapping`, `@PostMapping`, `@DeleteMapping`)
- `@PathVariable` for URL parameters

- `@RequestBody` for request body binding
- JSON automatic serialization/deserialization