Lab Exercise 4- — Create a Simple API using Spring Boot

Objective:

This lab will take you from **installing prerequisites** to **building, running, and testing** a simple Spring Boot REST API that exposes CRUD-like endpoints for a User resource. No previous Spring experience required.

---

1. Goal

Build a minimal Spring Boot REST API that supports:

- GET /api/users → list users
- GET /api/users/{id} → get user by id
- POST /api/users → create user
- PUT /api/users/{id} → update user
- DELETE /api/users/{id} → delete user

This lab uses **in-memory storage (HashMap)** so you don't need a database.

---

2. Prerequisites (install these first)

- Eclipse / Spring Tools Suite (STS)
- VS Code + Java extensions

Install the IDE and open it after setup.

---

3. Create a Spring Boot project

Option A — Spring Initializr (recommended)

Open https://start.spring.io/ in browser (or use IDE wizard):

- Project: **Maven Project**
- Language: **Java**
- Spring Boot: **latest 3.x** (e.g. 3.2.x)
- Group: com.example
- Artifact: simpleapi
- Dependencies:
    - **Spring Web**
    - (optional) Spring Boot DevTools

Click **Generate** → unzip the downloaded project → open it in your IDE.

---

4. Project structure (what you should see)

```
simpleapi/
 ├─ src/main/java/com/example/simpleapi/
 │    ├─ SimpleapiApplication.java
 │    └─ (we will add model, controller, service)
 ├─ src/main/resources/
 │    └─ application.properties
 └─ pom.xml
```

---

5. Implement the API (add code)

Create these files under src/main/java/com/example/simpleapi/

5.1 Model — User.java

```
package com.example.simpleapi.model;

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

    public User() {}

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

    // getters & 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; }
}
```

5.2 Service — UserService.java

```
package com.example.simpleapi.service;

import com.example.simpleapi.model.User;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.concurrent.atomic.AtomicLong;

@Service
public class UserService {
    private final Map<Long, User> store = new HashMap<>();
    private final AtomicLong idGen = new AtomicLong(1);

    // seed a couple users
    public UserService() {
        save(new User(null, "Alice", "alice@example.com"));
        save(new User(null, "Bob", "bob@example.com"));
    }

    public List<User> findAll() {
        return new ArrayList<>(store.values());
    }

    public User findById(Long id) {
        return store.get(id);
    }

    public User save(User user) {
        if (user.getId() == null) {
            user.setId(idGen.getAndIncrement());
        }
        store.put(user.getId(), user);
        return user;
    }

    public boolean delete(Long id) {
        return store.remove(id) != null;
    }

    public boolean exists(Long id) {
        return store.containsKey(id);
    }
}
```

5.3 Controller — UserController.java

```
package com.example.simpleapi.controller;

import com.example.simpleapi.model.User;
import com.example.simpleapi.service.UserService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.URI;
import java.util.List;


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

    private final UserService svc;
    public UserController(UserService svc) { this.svc = svc; }

```

```
  @GetMapping
  public List<User> list() {
    return svc.findAll();
  }

  @GetMapping("/{id}")
  public ResponseEntity<User> get(@PathVariable Long id) {
    User u = svc.findById(id);
    return (u == null) ? ResponseEntity.notFound().build() : ResponseEntity.ok(u);
  }

  @PostMapping
  public ResponseEntity<User> create(@RequestBody User user) {
    User created = svc.save(user);
    return ResponseEntity.created(URI.create("/api/users/" +
created.getId())).body(created);
  }

  @PutMapping("/{id}")
  public ResponseEntity<User> update(@PathVariable Long id, @RequestBody User
user) {
    if (!svc.exists(id)) return ResponseEntity.notFound().build();
    user.setId(id);
    return ResponseEntity.ok(svc.save(user));
  }

  @DeleteMapping("/{id}")
  public ResponseEntity<Void> delete(@PathVariable Long id) {
    if (!svc.exists(id)) return ResponseEntity.notFound().build();
    svc.delete(id);
    return ResponseEntity.noContent().build();
  }
}
```

---

6. Configure application.properties (optional)

src/main/resources/application.properties

```
server.port=8080
spring.main.banner-mode=off
```

---

7. Build and run

From IDE

- Run SimpleapiApplication main class (Run as Spring Boot App).

From command line (Maven)

```
mvn clean package
java -jar target/simpleapi-0.0.1-SNAPSHOT.jar
```

You should see Spring Boot start and listen on port 8080.

---

8. Test the API (curl / Postman)

8.1 List users

```
http://localhost:8080/api/users
```

8.2 Get user by id

```
http://localhost:8080/api/users/1
```

8.3 Create user

```
POST http://localhost:8080/api/users

{"name":"Hitesh","email":"hitesh@test.com"}
```

8.4 Update user

```
PUT http://localhost:8080/api/users/1

{"name":"Alice Updated","email":"alice2@example.com"}
```

8.5 Delete user

```
DELETE http://localhost:8080/api/users/1
```