EXPERIMENT NO. 02

Title: Spring Boot with REST API

Outcome: Students will be able to develop Spring Boot with REST API

Theory:

1) Prerequisites

- Eclipse IDE for Java Developers (with **Maven** & **Spring Tools** is helpful).
- JDK installed (Java 11+ recommended).
- Internet to download Maven dependencies.
- (Optional) Postman or curl to test the API.

---

2) Create the project in Eclipse

1. File → **New** → **Other…** → **Maven** → **Maven Project** → Next.
2. Check *Create a simple project (skip archetype selection)* = **unchecked** → Next.
3. Choose *Archetype*: org.springframework.boot:spring-boot-starter-parent (or use Spring Initializr from Eclipse: File → New → Spring Starter Project).
4. Fill Group Id e.g. com.example, Artifact Id e.g. bookapi. Finish.

(If using Spring Initializr option, choose Java, Maven, Spring Boot version, and add dependencies: Web, JPA, H2.)

---

3) pom.xml (Maven) — minimal required deps

Replace or update your pom.xml with the following:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>bookapi</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.0</version> <!-- or latest stable -->
    <relativePath/>
  </parent>

  <properties>
    <java.version>17</java.version>
  </properties>
```

```
<dependencies>
  <!-- Web -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <!-- JPA -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <!-- H2 in-memory DB -->
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>

  <!-- Devtools (optional for hot reload) -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>

  <!-- Test -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
```

```
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
 </build>
</project>
```

If your Eclipse created the project from Spring Initializr most of this is generated already.

---

4) Project package structure (recommended)

```
src/main/java/com/example/bookapi
 ├─ BookApiApplication.java   (main class)
 ├─ controller/
 │   └─ BookController.java
 ├─ model/
 │   └─ Book.java
 ├─ repository/
 │   └─ BookRepository.java
 └─ service/
     └─ BookService.java
```

---

5) Main application class

src/main/java/com/example/bookapi/BookApiApplication.java:

```
package com.example.bookapi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BookApiApplication {
 public static void main(String[] args) {
  SpringApplication.run(BookApiApplication.class, args);
 }
}
```

---

6) Entity (model)

src/main/java/com/example/bookapi/model/Book.java:

```
package com.example.bookapi.model;

import jakarta.persistence.*;

@Entity
public class Book {
```

```
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String title;
  private String author;
  private Double price;

  // constructors
  public Book() {}
  public Book(String title, String author, Double price) {
    this.title = title; this.author = author; this.price = price;
  }

  // getters & setters
  public Long getId() { return id; }
  public void setId(Long id) { this.id = id; }
  public String getTitle() { return title; }
  public void setTitle(String title) { this.title = title; }
  public String getAuthor() { return author; }
  public void setAuthor(String author) { this.author = author; }
  public Double getPrice() { return price; }
  public void setPrice(Double price) { this.price = price; }
}
```

Note: Using jakarta.persistence.* for newer Spring Boot. If older, use javax.persistence.*.

---

7) Repository (Spring Data JPA)

src/main/java/com/example/bookapi/repository/BookRepository.java:

```
package com.example.bookapi.repository;

import com.example.bookapi.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface BookRepository extends JpaRepository<Book, Long> { }
```

---

8) Service layer (optional but recommended)

src/main/java/com/example/bookapi/service/BookService.java:

```
package com.example.bookapi.service;
```

```
import com.example.bookapi.model.Book;
import com.example.bookapi.repository.BookRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class BookService {
  private final BookRepository repo;

  public BookService(BookRepository repo) {
    this.repo = repo;
  }

  public List<Book> findAll() { return repo.findAll(); }
  public Optional<Book> findById(Long id) { return repo.findById(id); }
  public Book save(Book b) { return repo.save(b); }
  public void deleteById(Long id) { repo.deleteById(id); }
}
```

---

9) Controller — REST endpoints

src/main/java/com/example/bookapi/controller/BookController.java:

```
package com.example.bookapi.controller;

import com.example.bookapi.model.Book;
import com.example.bookapi.service.BookService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping("/api/books")
public class BookController {
  private final BookService service;

  public BookController(BookService service) { this.service = service; }

  @GetMapping
```

```
  public List<Book> getAll() { return service.findAll(); }

  @GetMapping("/{id}")
  public ResponseEntity<Book> getById(@PathVariable Long id) {
    return service.findById(id)
      .map(ResponseEntity::ok)
      .orElse(ResponseEntity.notFound().build());
  }

  @PostMapping
  public ResponseEntity<Book> create(@RequestBody Book book) {
    Book saved = service.save(book);
    return ResponseEntity.created(URI.create("/api/books/" + saved.getId()))
                .body(saved);
  }

  @PutMapping("/{id}")
  public ResponseEntity<Book> update(@PathVariable Long id, @RequestBody Book
book) {
    return service.findById(id).map(existing -> {
      existing.setTitle(book.getTitle());
      existing.setAuthor(book.getAuthor());
      existing.setPrice(book.getPrice());
      Book updated = service.save(existing);
      return ResponseEntity.ok(updated);
    }).orElse(ResponseEntity.notFound().build());
  }

  @DeleteMapping("/{id}")
  public ResponseEntity<Void> delete(@PathVariable Long id) {
    return service.findById(id).map(b -> {
      service.deleteById(id);
      return ResponseEntity.noContent().<Void>build();
    }).orElse(ResponseEntity.notFound().build());
  }
}
```

---

10) application.properties for H2 & JPA

src/main/resources/application.properties:

```
spring.datasource.url=jdbc:h2:mem:bookdb
spring.datasource.driverClassName=org.h2.Driver
```

```
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

# H2 console (browser)
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

---

11) Run the app in Eclipse

- Right-click BookApiApplication.java → **Run As** → **Spring Boot App** (or Java Application).
- In console you should see Tomcat started on port(s): 8080 ... and *Started BookApiApplication*.

---

12) Test the REST endpoints

Use curl or Postman. Examples:

- Create a book:

```
curl -X POST http://localhost:8080/api/books \
  -H "Content-Type: application/json" \
  -d '{"title":"Clean Code","author":"Robert C. Martin","price":29.99}'
```

- Get all books:

```
curl http://localhost:8080/api/books
```

- Get by id:

```
curl http://localhost:8080/api/books/1
```

- Update:

```
curl -X PUT http://localhost:8080/api/books/1 \
 -H "Content-Type: application/json" \
 -d '{"title":"Clean Code (Updated)","author":"Robert C. Martin","price":34.99}'
```

- Delete:

```
curl -X DELETE http://localhost:8080/api/books/1
```

- H2 console in browser: http://localhost:8080/h2-console JDBC URL: jdbc:h2:mem:bookdb, user sa, no password.

---

13) Build a runnable jar

From terminal (project root):

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

Questions:

1. What are the key advantages of using **Spring Boot** for building RESTful web services compared to traditional Spring MVC configuration
2. Explain the role of the following annotations in a Spring Boot REST API: @RestController, @RequestMapping, @GetMapping, and @Autowired.

Conclusion : Thus, I have implemented Spring Boot with REST API