

Lab Exercise 12- Efficient Data Serialization & Deserialization Using Spring Boot

1. Lab Title

Efficient Data Serialization and Deserialization using a Spring Boot REST API

2. Objective

In this lab, students will:

- Understand serialization/deserialization in REST APIs
 - Implement **JSON**, **XML**, and **Binary** serialization using Spring Boot
 - Expose REST endpoints that serialize and deserialize objects
 - Measure performance (time taken + size differences)
 - Compare API responses in different serialization formats
 - Test everything using Postman
-

3. Prerequisites

- JDK 17+
- Maven 3+
- Spring Boot 3.x
- Postman
- Knowledge of REST APIs

4. What You Will Build

A Spring Boot API exposing endpoints:

| Format | Serialize Endpoint | Deserialize Endpoint |
|------------------|------------------------|--------------------------|
| JSON | /api/v1/json/serialize | /api/v1/json/deserialize |
| XML | /api/v1/xml/serialize | /api/v1/xml/deserialize |
| Binary | /api/v1/bin/serialize | /api/v1/bin/deserialize |
| Performance Test | /api/v1/performance | — |

5. Create Project Structure

```
serialization-api/
├── model/User.java
├── service/SerializationService.java
├── controller/SerializationController.java
├── controller/PerformanceController.java
├── config/JacksonConfig.java
└── SerializationApiApplication.java
└── pom.xml
```

6. Add Dependencies in pom.xml

```
<dependencies>

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

    <!-- XML (JAXB) for Spring Boot -->
    <dependency>
        <groupId>org.glassfish.jaxb</groupId>
        <artifactId>jaxb-runtime</artifactId>
        <version>4.0.2</version>
    </dependency>

</dependencies>
```

7. Step 1 – Create the Model Class

File: User.java

```
package com.example.serializationapi.model;

import jakarta.xml.bind.annotation.XmlRootElement;
import java.io.Serializable;

@XmlRootElement
public class User implements Serializable {

    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; }
}
```

8. Step 2 – Create the Serialization Service

File: **SerializationService.java**

```
package com.example.serializationapi.service;

import com.example.serializationapi.model.User;
import com.fasterxml.jackson.databind.ObjectMapper;
import jakarta.xml.bind.*;
import org.springframework.stereotype.Service;
```

```
import java.io.*;  
  
@Service  
public class SerializationService {  
  
    private final ObjectMapper mapper = new ObjectMapper();  
  
    // ----- JSON -----  
    public void serializeJson(User user, String path) throws Exception {  
        mapper.writerWithDefaultPrettyPrinter().writeValue(new File(path), user);  
    }  
  
    public User deserializeJson(String path) throws Exception {  
        return mapper.readValue(new File(path), User.class);  
    }  
  
    // ----- XML -----  
    public void serializeXml(User user, String path) throws Exception {  
        JAXBContext ctx = JAXBContext.newInstance(User.class);  
        Marshaller m = ctx.createMarshaller();  
        m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);  
        m.marshal(user, new File(path));  
    }  
  
    public User deserializeXml(String path) throws Exception {  
        JAXBContext ctx = JAXBContext.newInstance(User.class);  
        Unmarshaller um = ctx.createUnmarshaller();  
        return (User) um.unmarshal(new File(path));  
    }  
  
    // ----- BINARY -----  
    public void serializeBinary(User user, String path) throws Exception {
```

```

ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(path));
oos.writeObject(user);
oos.close();
}

public User deserializeBinary(String path) throws Exception {
    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(path));
    User user = (User) ois.readObject();
    ois.close();
    return user;
}
}

```

9. Step 3 – Create the API Controller

File: **SerializationController.java**

```

package com.example.serializationapi.controller;

import com.example.serializationapi.model.User;
import com.example.serializationapi.service.SerializationService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/v1")
public class SerializationController {

    private final SerializationService service;

    public SerializationController(SerializationService service) {
        this.service = service;
    }
}

```

```
}

// ----- JSON -----
@PostMapping("/json/serialize")
public String serializeJson(@RequestBody User user) throws Exception {
    service.serializeJson(user, "user.json");
    return "JSON serialized to user.json";
}

@GetMapping("/json/deserialize")
public User deserializeJson() throws Exception {
    return service.deserializeJson("user.json");
}

// ----- XML -----
@PostMapping("/xml/serialize")
public String serializeXml(@RequestBody User user) throws Exception {
    service.serializeXml(user, "user.xml");
    return "XML serialized to user.xml";
}

@GetMapping("/xml/deserialize")
public User deserializeXml() throws Exception {
    return service.deserializeXml("user.xml");
}

// ----- BINARY -----
@PostMapping("/bin/serialize")
public String serializeBinary(@RequestBody User user) throws Exception {
    service.serializeBinary(user, "user.bin");
    return "Binary serialized to user.bin";
}
```

```
@GetMapping("/bin/deserialize")
public User deserializeBinary() throws Exception {
    return service.deserializeBinary("user.bin");
}
```

10. Step 4 – Performance Test Controller

File: PerformanceController.java

```
package com.example.serializationapi.controller;

import com.example.serializationapi.model.User;
import com.example.serializationapi.service.SerializationService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/v1/performance")
public class PerformanceController {

    private final SerializationService service;
    private final User sampleUser = new User(1L, "John Doe", "john@example.com");

    public PerformanceController(SerializationService service) {
        this.service = service;
    }

    @GetMapping
    public String testPerformance() throws Exception {
        long start, end;
```

```
// JSON
start = System.nanoTime();
service.serializeJson(sampleUser, "user.json");
service.deserializeJson("user.json");
end = System.nanoTime();
long jsonTime = end - start;

// XML
start = System.nanoTime();
service.serializeXml(sampleUser, "user.xml");
service.deserializeXml("user.xml");
end = System.nanoTime();
long xmlTime = end - start;

// Binary
start = System.nanoTime();
service.serializeBinary(sampleUser, "user.bin");
service.deserializeBinary("user.bin");
end = System.nanoTime();
long binTime = end - start;

return "JSON: " + jsonTime + " ns\n" +
       "XML: " + xmlTime + " ns\n" +
       "Binary: " + binTime + " ns\n";
}
```

11. Step 5 – application.properties

```
server.port=8080  
spring.mvc.converters.preferred-json-mapper=jackson
```

12. Step 6 – Run the Application

```
mvn spring-boot:run
```

13. Step 7 – Postman Test Cases

A. JSON Serialization

1. Serialize JSON

POST

```
http://localhost:8080/api/v1/json/serialize
```

Body:

```
{  
  "id": 100,  
  "name": "Alice",  
  "email": "alice@test.com"  
}
```

2. Deserialize JSON

GET

```
http://localhost:8080/api/v1/json/deserialize
```

B. XML Serialization

1. Serialize

```
POST http://localhost:8080/api/v1/xml/serialize
```

Body:

```
{  
    "id": 100,  
    "name": "Alice",  
    "email": "alice@test.com"  
}
```

2. Deserialize

```
GET http://localhost:8080/api/v1/xml/deserialize
```

C. Binary Serialization

Serialize Binary

```
POST http://localhost:8080/api/v1/bin/serialize
```

Deserialize Binary

```
GET http://localhost:8080/api/v1/bin/deserialize
```

D. Performance Comparison

```
GET http://localhost:8080/api/v1/performance
```