Building and Testing RESTful Web Services in Spring Boot

1. HTTP Request and Response

What is HTTP?

HTTP (Hypertext Transfer Protocol) is a protocol used for communication between clients (like browsers or mobile apps) and web servers.

HTTP Request Format

A typical HTTP request includes:

GET /hello.txt HTTP/1.1

Host: www.example.com

User-Agent: curl/7.16.3

Accept-Language: en

- **GET**: HTTP Method (Request Type)
- **/hello.txt**: Resource being requested
- **HTTP/1.1**: HTTP version
- **Host**: Server domain
- **User-Agent**: Client browser/tool info

HTTP Response Format

HTTP/1.1 200 OK

Content-Type: text/plain

Content-Length: 51

Hello World! My payload includes a trailing CRLF.

- **HTTP/1.1**: Version
- **200 OK**: Status code
- **Content-Type**: Format of response (text/html, application/json, etc.)
- **Body**: Actual response content

2. Need and Benefits of RESTful Web Services

What is REST?

REST (Representational State Transfer) is a lightweight web service architecture based on HTTP.

Key Benefits:

- **Lightweight**: Uses standard HTTP methods like GET, POST, etc.
- **Scalable**: Suitable for large applications
- **Stateless**: Every request is independent
- **Maintainable**: Clear separation between client and server
- **Universal**: Works with browsers, mobile apps, and APIs

3. GET Method: Hello World Example

Code Example:

```
@RestController
public class HelloController {

   private static final Logger LOGGER = LoggerFactory.getLogger(HelloController.class);

   @GetMapping("/hello")
   public String sayHello() {
      LOGGER.info("START - sayHello()");
      String message = "Hello World!!";
      LOGGER.info("END - sayHello()");
      return message;
   }
}
```

Test:

- **URL**: http://localhost:8083/hello
- **Tools**: Browser / Postman
- **Response**: Hello World!!

4. Country Web Service (Single Country)

Code:

```
@RequestMapping("/country")
public Country getCountryIndia() {
```

```
    ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
    return (Country) context.getBean("in");
}
```

Sample XML:

```
<bean id="in" class="com.example.Country">
    <property name="code" value="IN"/>
    <property name="name" value="India"/>
</bean>
```

5. Get All Countries

```
@GetMapping("/countries")
public List<Country> getAllCountries() {
    ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
    return (List<Country>) context.getBean("countryList");
}
```

URL: http://localhost:8083/countries

Response:

```
[
  { "code": "IN", "name": "India" },
  { "code": "US", "name": "United States" },
  { "code": "JP", "name": "Japan" },
  { "code": "DE", "name": "Germany" }
]
```

6. Get Country by Code

```
@GetMapping("/countries/{code}")
public Country getCountry(@PathVariable String code) throws CountryNotFoundException {
    List<Country> countries = getAllCountries();
    return countries.stream()
        .filter(c -> c.getCode().equalsIgnoreCase(code))
        .findFirst()
```

```
        .orElseThrow(() -> new CountryNotFoundException());
}
```

7. Exception Handling for Invalid Country Code

Exception Class:

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Country not found")
public class CountryNotFoundException extends Exception {
}
```

Sample Request: http://localhost:8083/country/az
Response:

```
{
  "status": 404,
  "error": "Not Found",
  "message": "Country not found"
}
```

8. Testing with MockMVC

Test Class Setup:

```
@SpringBootTest
@AutoConfigureMockMvc
public class SpringLearnApplicationTests {

    @Autowired
    private CountryController countryController;

    @Autowired
    private MockMvc mvc;

    @Test
    public void contextLoads() {
        assertNotNull(countryController);
```

```
    }

    @Test
    public void testGetCountry() throws Exception {
        ResultActions actions = mvc.perform(get("/country"));
        actions.andExpect(status().isOk());
        actions.andExpect(jsonPath("$.code").exists());
        actions.andExpect(jsonPath("$.code").value("IN"));
        actions.andExpect(jsonPath("$.name").value("India"));
    }

    @Test
    public void testGetCountryException() throws Exception {
        mvc.perform(get("/countries/zz"))
            .andExpect(status().isNotFound())
            .andExpect(status().reason("Country not found"));
    }
}
```

Maven Command to Run Tests:

```
mvn clean test
```

9. Testing with Browser and Postman

- Open **Chrome DevTools → Network Tab**
- Make a request to the REST endpoint
- View **Request Headers** and **Response Headers**
- In **Postman** → Click on "Headers" tab after calling the API
- Observe key headers like:
    - Content-Type
    - User-Agent
    - Accept
    - Response Status Code