Mastering API Automation with Rest Assured & POJOs

API Automation Essentials

Introduction

Master these four key areas to excel in API automation with Rest Assured and Java:

- Request Specification: Define base URIs, headers, and parameters once to keep tests DRY.
- Response Validation: Verify status codes, headers, and response data accurately.
- Authentication: Handle security with Basic, OAuth2, API keys, etc.
- POJOs & Serialization: Map API data to Java objects for clean, maintainable tests.

1. Request Specification & Parameterization

- Set base URIs, headers, query/path parameters, and reuse request logic with RequestSpecification.
- Keeps tests maintainable and DRY (Don't Repeat Yourself).

Example:

```
RequestSpecification req = given()
  .baseUri("https://api.example.com")
  .header("Authorization", "Bearer token");
```

2. Response Validation & Assertions

- Validate status codes, headers, response body fields, and use Hamcrest matchers.
- Ensures your API works as expected and catches regressions quickly.

Example:

```
.then()
  .statusCode(200)
  .body("name", equalTo("John"));
```

3. Authentication & Authorization Handling

- Implement authentication methods like Basic, Bearer, OAuth2, API keys.
- Essential for working with secured real-world APIs.

Example:

```
given()
  .auth()
  .oauth2("your_access_token")
  .get("/secure-endpoint");
```

4. What is a POJO?

- POJO = Plain Old Java Object - a simple Java class with private fields and public getters/setters.
- Used to represent structured data in your Java code.
- Keeps code type-safe, clean, and easy to maintain.

Example POJO class:

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

    // Getters and setters
    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. Why Use POJOs in API Testing?

- POJOs help map JSON or XML API responses to Java objects.
- Makes test code readable and maintainable.
- Avoids manual string parsing or handling maps.

Example JSON:

```
{
  "name": "Priya",
  "email": "priya@example.com"
}
```

6. What is Serialization?

- Serialization = Turning a Java object (POJO) into a data format (JSON or XML) for transmission.
- Used when you want to send data in an HTTP request.
- In Rest Assured, passing a POJO to `.body()` automatically serializes it to JSON.
- Saves time by avoiding manual JSON building.

Example:

```
User user = new User();
user.setName("Priya");
user.setEmail("priya@example.com");

given()
  .contentType("application/json")
  .body(user)  // Serialization: Java -> JSON
.when()
  .post("/users");
```

Behind the scenes:

```
{
  "name": "Priya",
  "email": "priya@example.com"
}
```

7. What is Deserialization?

- Deserialization = Turning JSON (or XML) from an HTTP response into a Java object (POJO).
- Used when you want to read or assert API response data.
- In Rest Assured, use `.extract().as(POJO.class)` to deserialize the response.

Example:

```
User responseUser = given()
  .get("/users/1")
  .then()
  .extract()
  .as(User.class);  // Deserialization: JSON -> Java
```

```
System.out.println(responseUser.getName());
```

Behind the scenes:

```
{
  "name": "Priya",
  "email": "priya@example.com"
}
```

8. Advanced: Nested Objects & Lists

- APIs often return complex JSON with nested objects or lists.
- Use nested POJO classes or Java Lists inside your POJO to handle these.

Example POJO with a list:

```
public class User {
    private String name;
    private String email;
    private List<String> roles;  // List of roles

    // Getters and setters
}
```

9. Under the Hood

- Rest Assured uses Jackson (default) or Gson libraries for conversion.
- These libraries handle serialization and deserialization details behind the scenes.

Interview Q&A: Rest Assured & POJO Essentials

1. What is RequestSpecification and why is it useful?

What is RequestSpecification and why is it useful?

- Allows setting base URIs, headers, and parameters once to reuse across requests, keeping tests DRY.

2. How do you set base URI and headers using RequestSpecification?

How do you set base URI and headers using RequestSpecification?

- Use `given().baseUri("https://api.example.com").header("Authorization", "Bearer token")`.

3. How to add query and path parameters?

How to add query and path parameters?

- Use `.queryParam("key", "value")` for query parameters and `.pathParam("id", 123)` for path parameters.

4. How do you validate status codes and response body fields?

How do you validate status codes and response body fields?

- Use `.then().statusCode(200).body("name", equalTo("John"))` with Hamcrest matchers.

5. How to validate response headers?

How to validate response headers?

- Use `.then().header("Content-Type", "application/json")`.

6. What authentication methods does Rest Assured support?

What authentication methods does Rest Assured support?

- Basic, Bearer tokens, OAuth2, API keys, and custom headers.

7. How do you pass OAuth2 tokens?

How do you pass OAuth2 tokens?

- Use `.auth().oauth2("your_access_token")`.

8. What is a POJO and why is it important?

What is a POJO and why is it important?

- Plain Old Java Object (POJO) represents structured API data in Java, making tests type-safe.

9. How to serialize a POJO in Rest Assured?

How to serialize a POJO in Rest Assured?

- Pass the POJO to `.body()` to automatically convert it to JSON.

10. How to deserialize a response to a POJO?

How to deserialize a response to a POJO?

- Use `.extract().as(YourPOJO.class)` to convert JSON response to Java object.

11. How to handle nested JSON in POJOs?

How to handle nested JSON in POJOs?

- Define nested classes or use `List` fields for arrays.

12. Which libraries handle JSON serialization/deserialization?

Which libraries handle JSON serialization/deserialization?

- Jackson (default) and Gson.

13. How to map differing JSON and Java field names?

How to map differing JSON and Java field names?

- Use `@JsonProperty("json_field_name")` annotation.

14. Why prefer POJOs over manual JSON parsing?

Why prefer POJOs over manual JSON parsing?

- POJOs offer type safety, better refactoring, and IDE support.

15. How to validate API responses using POJOs?

How to validate API responses using POJOs?

- Deserialize responses and assert fields in Java test code.

Conclusion

- Strong API tests rely on reusable requests, precise validations, proper authentication, and clear data mapping.
- Mastering these improves test quality and interview readiness.