# E2E Automation Framework Creation

**Author:** Jatin Sharma

Connect with Me: Linked in

**If you are preparing for Job Change and want to Prepare for SDET interviews Fast**
**Checkout :** SDET Mastery Course 🔥

# Day 1: Project Setup & Introduction

### Agenda

1. Introduction to API Testing
2. Setting Up Development Environment
3. Creating First Test

## API Testing

- API (Application Programming Interface) testing is a type of software testing that focuses on verifying and validating the functionality, reliability, performance, and security of APIs.
- APIs are intermediaries that allow different software systems to communicate with each other, typically handling data requests and responses.
- In API testing, instead of focusing on the user interface (UI), testers interact directly with the API using tools or code to ensure it works as expected.

---

## Key Objectives of API Testing

1. **Functionality Testing**:
    - Ensure that the API returns the correct responses for various input conditions.
    - Verify that all endpoints (GET, POST, PUT, DELETE) behave as expected.

2. **Reliability Testing**:
    - Test whether the API consistently provides the intended results without any errors.
3. **Performance Testing**:
    - Measure how the API performs under different loads (e.g., high traffic) and stress conditions.
    - Check response time and latency.
4. **Security Testing**:
    - Verify that sensitive data is handled securely (e.g., authentication, encryption).
    - Ensure the API is protected against attacks like SQL injection, XSS, or DDoS.
5. **Error Handling**:
    - Ensure appropriate error messages are returned for invalid requests.
    - Validate HTTP status codes (e.g., 200 for success, 404 for not found).

---

## Benefits of API Testing

- **Faster Testing**: APIs can be tested before the UI is complete, accelerating the testing process.
- **Comprehensive Coverage**: Ensures that business logic and data transfer between systems work as expected.
- **Early Bug Detection**: Allows issues in backend services to be caught earlier in the development lifecycle.
- **Automation-Friendly**: APIs are easily automatable with tools and scripts, making regression testing efficient.

---

## How to Perform API Testing

1. **Understand API Requirements**:
    - Review API documentation (e.g., Swagger, Postman Collection).
    - Understand the request structure, endpoints, authentication methods, and response formats.

2. **Set Up Test Environment**:
    - Ensure the API server and database are configured properly.
3. **Write Test Cases**:
    - Cover positive and negative scenarios.
    - Include tests for edge cases, boundary values, and performance thresholds.
4. **Execute Tests**:
    - Use tools like Postman, Rest Assured (Java), or tools like SoapUI and JMeter.
    - Send requests and verify responses.
5. **Validate**:
    - Check the correctness of the response (data, status codes, headers).
    - Validate against expected performance metrics.
6. **Report and Fix Bugs**:
    - Document issues and share them with developers for resolution.

---

## Common Tools for API Testing

- **Postman**: Simplifies manual and automated API testing.
- **Rest Assured**: A Java library for automating REST API tests.
- **SoapUI**: For testing SOAP and REST APIs.
- **JMeter**: For load and performance testing of APIs.
- **k6**: Focused on performance testing.
- **Newman**: CLI for running Postman collections.

---

### Project Setup in Eclipse

Create a Maven Based Project in Eclipse

and Add the dependency in your pom.xml

```
<!-- pom.xml --><dependencies> <dependency> <groupId>io.rest-assured</groupId
> <artifactId>rest-assured</artifactId> <version>5.3.0</version> </dependency
> <!-- https://mvnrepository.com/artifact/org.testng/testng --><dependency> <
groupId>org.testng</groupId> <artifactId>testng</artifactId> <version>7.10.2
</version> <scope>test</scope> </dependency> </dependencies>
```

## Project Structure for Our API Automation Framework

This is will be project Structure for our API Automation Framework

```
├── main/java/com/api | ├── config | ├── services | └── utils └── test/java/c
om/api └── tests
```

## Simple Rest Assured Test Script:

```
@Test public void login() { given() .baseUri("http://64.227.160.186:8080/api/
auth/") .header("Content-type","application/json") .body("{\"username\": \"ud
ay1234\", \"password\": \"uday12345\"}") .when() .log().all() .post("/login")
.then() .log().all() .statusCode(200) .time(lessThan(3000L)) }
```

Here we are making an POST Request to the endpoint `login` with the user credentials uday1234 and password uday12345 and verifying if the status code is 200 and response time is less than 3seconds.

## Now the Design Pattern:

Service Objects are a design pattern commonly used in software development, particularly in web applications, to encapsulate business logic and keep it separate from other layers of the application.

Key Benefits:

1. **Separation of Concerns** - Business logic is separate from model classes(POJO)
2. Reusability - Service objects can be used by different parts of application
3. Testability - Easier to unit test business logic
4. Maintainability - Changes to business logic are contained in one place

**NOTE: Model Classes are classes that represent the core data structure or entities in your application**

In your API Automation Framework the Request and response payload will be have their respective model classes.

| Aspect | Service Object Model (SOM) | Page Object Model (POM) |
|---|---|---|
| Purpose | Handles business logic and application rule | Handles UI elements and interactio |
| Layer | Business/Service Layer | UI/Presentation Layer |
| Usage | Core application functionality | Test automation framework |
| Main Focus | Data processing and business operations | Web element interactions and UI ve |
| Example Classes | Auth Service | LoginPage, StudentRegistrationPag DashboardPage |
| Methods Contain | Business logic methods | UI interaction methods (clickButtor |
| Dependencies | Database connections, other services | WebDriver, UI elements (buttons, fi |
| When to Use | When organizing business logic and operations | When creating UI test automation f |
| Scope | Backend operations | Frontend operations |
| Error Handling | Business logic exceptions | UI interaction exceptions |
| Testing Focus | Unit testing business logic | UI testing |

Would you like me to elaborate on any of these aspects or provide specific code examples for either model?

- Base Configuration Setup
- Creating BaseService
- HTTP Methods Implementation BaseService Implementation

```
public class BaseService { protected RequestSpecification requestSpec; public BaseService() { this.requestSpec = RestAssured.given() .baseUri(ConfigManager.getBaseUrl()) } }
```

Configuration Setup

```
public class ConfigManager { private static Properties props; static { loadProperties(); } private static void loadProperties() { props = new Properties(); try (InputStream input = ConfigManager.class.getClassLoader().getResourceAsStream("config.properties")) { if (input == null) { throw new RuntimeException("Unable to find config.properties"); } props.load(input); } catch (IOException e) { throw new RuntimeException("Failed to load properties file", e); } } }
```

## 📂 Code Structure

```
com.api ├── services │ ├── BaseService.java │ └── AuthService.java ├── models │ ├── request │ │ └── LoginRequest.java │ └── response │ └── LoginResponse.java └── tests └── AuthTest.java
```

## 💻 Implementation

### 1. Login Request Model

```
public class LoginRequest { private String username; private String password; // Constructor public LoginRequest(String username, String password) { this.username = username; this.password = password; } // Getters and Setters public String getUsername() { return username; } public void setUsername(String username) { this.username = username; } public String getPassword() { return password; } public void setPassword(String password) { this.password = password; } }
```

## 2. Login Response Model

```
public class LoginResponse { private String token; private String type; private String username; // Getters and Setters public String getToken() { return token; } public void setToken(String token) { this.token = token; } // ... other getters/setters }
```

## 3. Auth Service

```
public class AuthService extends BaseService{ public Response login(Object payload) { return postRequest("/api/auth/login", payload); } }
```

### Add the Dependency

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind --> <dependency> <groupId>com.fasterxml.jackson.core</groupId> <artifactId>jackson-databind</artifactId> <version>2.18.2</version> </dependency> <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core --> <dependency> <groupId>com.fasterxml.jackson.core</groupId> <artifactId>jackson-core</artifactId> <version>2.18.2</version> </dependency> <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind --> <dependency> <groupId>com.fasterxml.jackson.core</groupId> <artifactId>jackson-databind</artifactId> <version>2.18.2</version> </dependency>
```

## 4. Basic Test Example

```
public class AuthTest { private AuthService authService; @BeforeClass public void setup() { authService = new AuthService(); } @Test public void testSuccessfulLogin() { LoginRequest request = new LoginRequest("testuser", "testpass"); Response response = authService.login(request); assertEquals(200, response.getStatusCode()); LoginResponse loginResponse = response.as(LoginResponse.class); assertNotNull(loginResponse.getToken()); } }
```

## User Service

Modify the Base Service:

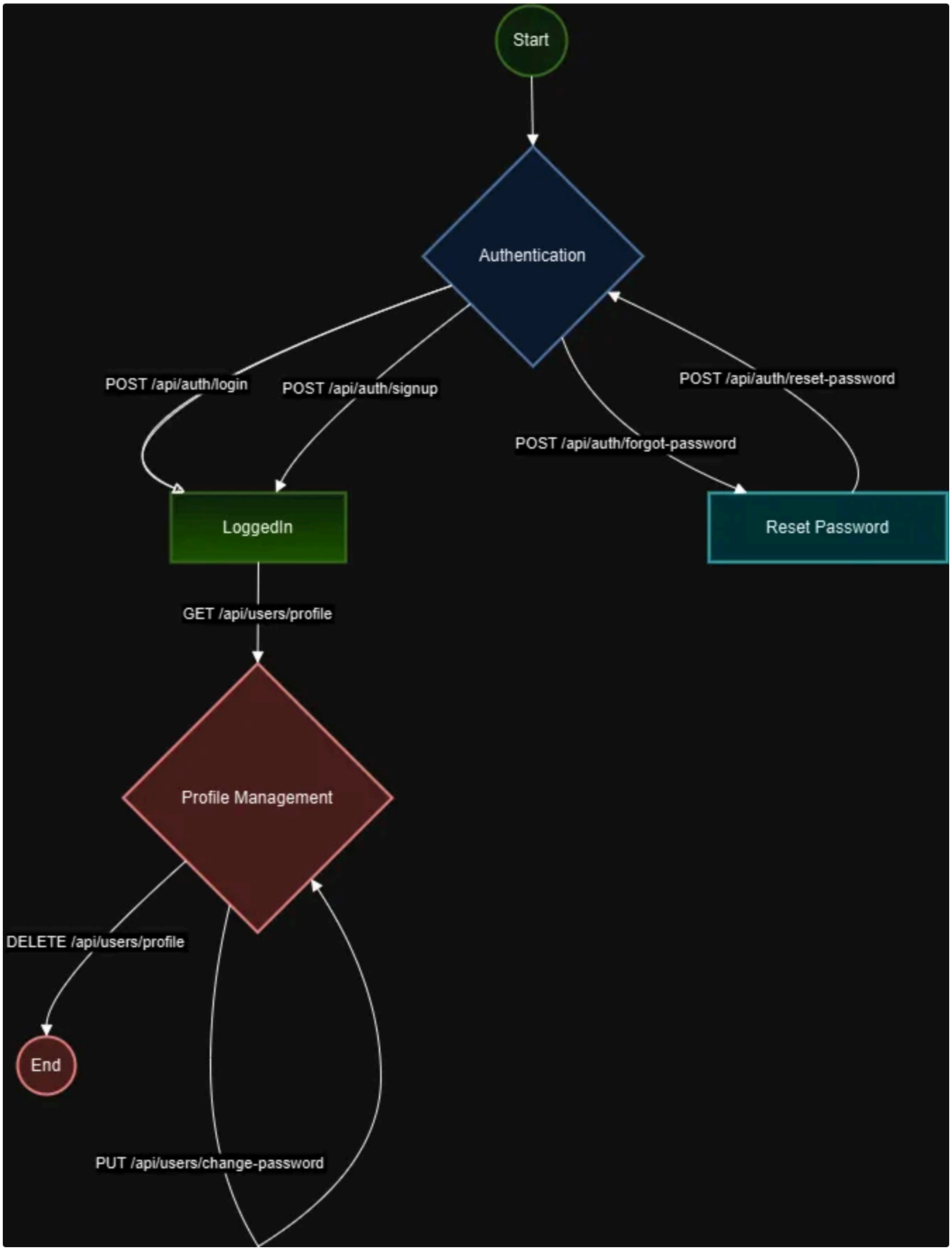
**public void** attachAuthToken(String token) {

requestSpec.header("Authorization", "Bearer "+ token);

}

# Day 2 Creating an E2E solution for User Management Service

User Management flow:


Start
Authentication
POST /api/auth/login
POST /api/auth/signup
POST /api/auth/reset-password
POST /api/auth/forgot-password
LoggedIn
Reset Password
GET /api/users/profile
Profile Management
DELETE /api/users/profile
End
PUT /api/users/change-password

## ➡️ Part 3 Coming Soon!

📱 Share Your Progress! I'd love to see what you build - connect with me on LinkedIn (@Jatin Shharma) and share your framework. Your success stories inspire our community!

# Day 3

## 📂 Project Updates

```
src ├── test ├── java/com/api/filters │ │ │── LoggingFilter.java │ └── liste
ners │ └── TestListener.java └── resources └── log4j2.xml
```

## 💻 Implementation

## Add the Dependency in Pom.xml

```
<dependencies> <!-- Log4j2 --><dependency> <groupId>org.apache.logging.log4j
</groupId> <artifactId>log4j-api</artifactId> <version>2.20.0</version> </dep
endency> <dependency> <groupId>org.apache.logging.log4j</groupId> <artifactId
>log4j-core</artifactId> <version>2.20.0</version> </dependency> </dependenci
es>
```

## 1. log4j2.xml Configuration

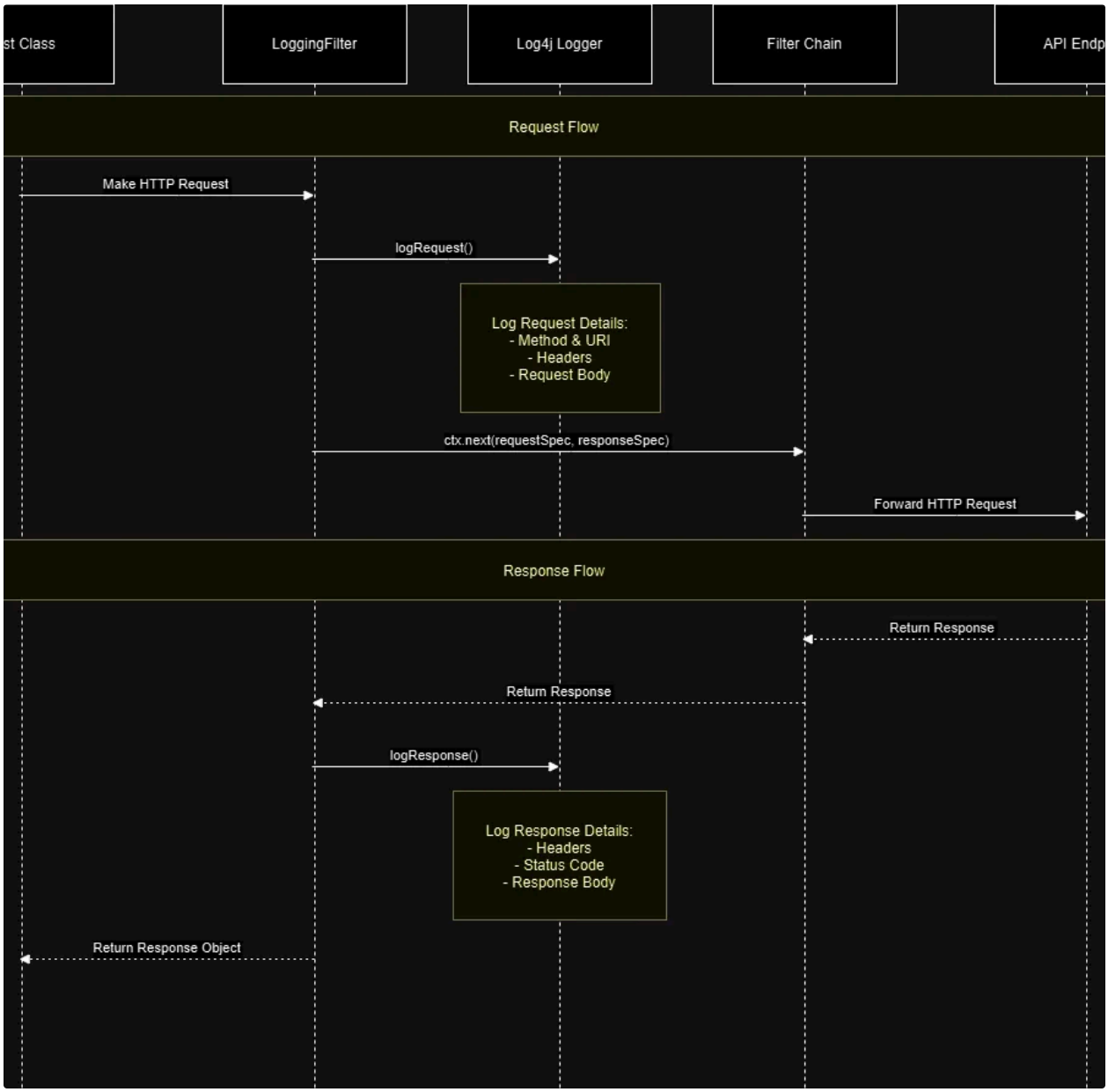
```
<?xml version="1.0" encoding="UTF-8"?> <Configuration status="WARN"> <Appende
rs> <Console name="Console" target="SYSTEM_OUT"> <PatternLayout pattern="%d{H
H:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/> </Console> <File name="Fil
e" fileName="logs/test.log"> <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-
5level %logger{36} - %msg%n"/> </File> </Appenders> <Loggers> <Root level="in
fo"> <AppenderRef ref="Console"/> <AppenderRef ref="File"/> </Root> </Loggers
> </Configuration>
```

## 2. Basic Test Listener

```
public class TestListener implements ITestListener { private static final Log
ger logger = LogManager.getLogger(TestListener.class); @Override public void
onTestStart(ITestResult result) { logger.info("Test Started: {}", result.getN
ame()); } @Override public void onTestSuccess(ITestResult result) { logger.in
fo("Test Passed: {}", result.getName()); } @Override public void onTestFailur
e(ITestResult result) { logger.error("Test Failed: {}", result.getName()); lo
gger.error("Exception: ", result.getThrowable()); } }
```

## 3. Logging Filter

```
public class LoggingFilter implements Filter { private static final Logger lo
gger = LogManager.getLogger(LoggingFilter.class); @Override public Response f
ilter(FilterableRequestSpecification requestSpec, FilterableResponseSpecifica
tion responseSpec, FilterContext ctx) { // Log request logRequest(requestSpe
c); // Get responseResponse response = ctx.next(requestSpec, responseSpec);
// Log response logResponse(response); return response; } private void logReq
uest(FilterableRequestSpecification requestSpec) { logger.info("Request: {}
{}", requestSpec.getMethod(), requestSpec.getURI()); logger.info("Headers:
{}", requestSpec.getHeaders()); } private void logResponse(Response response)
{ logger.info("Response Status: {}", response.getStatusCode()); logger.info
("Response Body: {}", response.getBody().asString()); } }
```




### Filters in Rest Assured Framework

Intercepting in software means catching or "interrupting" something while it's moving from point A to point B, similar to intercepting a pass in football. It allows you to perform actions before letting it continue its journey.

In the context of this logging filter code, intercepting means:

1. The **HTTP request is created by your application**
2. Before it reaches the server, the filter "catches" it (intercepts)

3. The filter can then:
    - *Examine the request (in this case, log its details)*
    - *Modify it if needed (though this example only logs)*
    - *Decide whether to let it continue or block it*

## 4. Update BaseService.java

```
public class BaseService { static { RestAssured.filters(new LoggingFilter());
} protected RequestSpecification requestSpec; protected BaseService() { this.
requestSpec = RestAssured.given() .contentType(ContentType.JSON); } }
```

## 5 Running the Test with TestNG.xml

### 1. TestNG Parallel Suite Configuration

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd"> <suite name="Para
llel Test Suite" parallel="methods" thread-count="4"> <listeners> <listener c
lass-name="com.api.listeners.TestListener"/> </listeners> <test name="API Tes
ts"> <groups> <run> <include name="auth"/> <include name="user"/> </run> </gr
oups> <classes> <class name="com.api.tests.AuthTest"/> <class name="com.api.t
ests.UserTest"/> </classes> </test> </suite>
```

## 📂 Project Structure