REST Assured Framework Skeleton

Here's a basic **REST Assured framework skeleton/architecture** for API automation using **Java + TestNG + Maven**. It follows a modular and scalable structure suitable for real-time projects.

✅ 1. Project Structure

```
css

rest-assured-framework/
│
├── src/
│   ├── main/
│   │   └── java/
│   │       └── utils/
│   │           ├── ConfigReader.java
│   │           └── ExcelReader.java (optional)
│   │
│   └── test/
│       ├── java/
│       │   ├── base/
│       │   │   └── BaseTest.java
│       │   │
│       │   ├── tests/
│       │   │   └── UserApiTest.java
│       │   │
│       │   ├── endpoints/
│       │   │   └── UserEndpoints.java
│       │   │
│       │   ├── payloads/
│       │   │   └── UserPayload.java
│       │   │
│       │   └── utils/
│       │       └── TestDataUtil.java
│       │
│       └── resources/
│           └── config.properties
│
├── testng.xml
├── pom.xml
└── README.md
```

✅ 2. Core Components Explained

🔹 **`BaseTest.java`**

Handles:

- Base URI
- Request specification setup
- Hooks (`@BeforeSuite`, `@AfterSuite`, etc.)

```
java

public class BaseTest {
    @BeforeSuite
    public void setup() {
```

```
        RestAssured.baseURI = ConfigReader.getProperty("baseURL");
    }
}
```

🔹 **UserEndpoints.java**

Handles the API calls (GET/POST/PUT/DELETE).

```
java

public class UserEndpoints {

    public static Response createUser(UserPayload payload) {
        return given()
                .contentType(ContentType.JSON)
                .body(payload)
                .when()
                .post("/users");
    }

    public static Response getUser(String username) {
        return given()
                .pathParam("username", username)
                .when()
                .get("/users/{username}");
    }

    // Add updateUser, deleteUser etc.
}
```

🔹 **UserPayload.java**

POJO class to map request body.

```
java

public class UserPayload {
    private String name;
    private String job;

    // Getters and Setters
}
```

🔹 **UserApiTest.java**

Test class using TestNG annotations.

```
java

public class UserApiTest extends BaseTest {

    @Test
    public void testCreateUser() {
        UserPayload user = new UserPayload();
        user.setName("John");
        user.setJob("Engineer");

        Response response = UserEndpoints.createUser(user);
        response.then().statusCode(201);
```

```
    }
}
```

🔹 **ConfigReader.java**

Reads config values.

```
java

public class ConfigReader {
    private static Properties prop;

    static {
        try (FileInputStream fis = new FileInputStream("src/test/resources/config.properties"))
{
            prop = new Properties();
            prop.load(fis);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static String getProperty(String key) {
        return prop.getProperty(key);
    }
}
```

🔹 **config.properties**

Stores API URLs, credentials, environment flags.

```
properties

baseURL=https://reqres.in/api
```

✅ 3. `pom.xml` Dependencies

```
xml

<dependencies>
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>5.4.0</version>
    </dependency>

    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.9.0</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.17.1</version>
    </dependency>
</dependencies>
```

✅ 4. `testng.xml`

```
xml

<suite name="API Automation Suite">
    <test name="User API Tests">
        <classes>
            <class name="tests.UserApiTest"/>
        </classes>
    </test>
</suite>
```

✅ Optional Enhancements

- ✅ **Logging filters** (Request/Response)
- ✅ **Extent Reports / Allure**
- ✅ **Data-driven testing** from Excel/CSV/JSON
- ✅ **Retry logic, test listeners**

Would you like a downloadable version or GitHub template of this framework?

In **REST Assured**, the `given()` method is used to **set up preconditions** before sending an API request. It is the starting point of a BDD-style syntax and allows you to define:

✅ Purpose of `given()`

`given()` is used to configure:

- Headers
- Cookies
- Authentication
- Request body (payload)
- Path or query parameters
- Content type
- Request logging

✅ Basic Syntax

```
java

given()
    .contentType(ContentType.JSON)
    .header("Authorization", "Bearer token")
    .body(requestBody)
.when()
    .post("/api/endpoint")
.then()
    .statusCode(200);
```

✅ Example 1: Set JSON payload and content type

```
java

given()
    .contentType(ContentType.JSON)
    .body("{ \"name\": \"John\", \"job\": \"Engineer\" }")
.when()
    .post("/users")
.then()
    .statusCode(201);
```

✅ Example 2: Set headers and query parameters

```
java

given()
    .header("Accept", "application/json")
    .queryParam("page", 2)
.when()
    .get("/users")
```

```
.then()
    .statusCode(200);
```

✅ Example 3: Add authentication

```
java

given()
    .auth().basic("username", "password")
.when()
    .get("/secure-data")
.then()
    .statusCode(200);
```

✅ Example 4: Log request details

```
java

given()
    .log().all()
    .contentType(ContentType.JSON)
    .body(userPayload)
.when()
    .post("/users")
.then()
    .log().all()
    .statusCode(201);
```

✅ Summary Table

`given()` Configuration	Purpose
`.contentType()`	Set request content type
`.header()` / `.headers()`	Add single or multiple headers
`.body()`	Add JSON, XML, or form body
`.queryParam()`	Add query parameters
`.pathParam()`	Add path parameters
`.auth()`	Authentication (basic, OAuth, etc.)
`.cookie()`	Send cookies
`.log().all()`	Log full request

Let me know if you want a **real API practice project** using `given()` with REST Assured.

✅ `when()` – Call the API in REST Assured

The **`when()`** method in REST Assured is used to **initiate the actual HTTP request** after the preconditions have been set using **`given()`**.

🔹 Purpose of `when()`

- It **triggers the request** (GET, POST, PUT, DELETE, PATCH).
- Comes **after** **`given()`** (which sets up headers, body, etc.).
- Works with HTTP method functions like:
  - **`.get()`**
  - **`.post()`**
  - **`.put()`**
  - **`.delete()`**
  - **`.patch()`**

✅ Syntax

```
java

given()
    // preconditions like headers, body
.when()
    .get("/endpoint")  // <--- API call made here
.then()
    // assertions or response validations
```

✅ Example 1: GET request

```
java

given()
    .queryParam("page", 2)
.when()
    .get("/users")
.then()
    .statusCode(200);
```

✅ Example 2: POST request

```
java

given()
    .contentType(ContentType.JSON)
    .body("{ \"name\": \"John\", \"job\": \"Engineer\" }")
.when()
    .post("/users")
```

```
.then()
    .statusCode(201);
```

✅ Example 3: PUT request

```
java

given()
    .contentType(ContentType.JSON)
    .body("{ \"name\": \"John\", \"job\": \"Manager\" }")
.when()
    .put("/users/2")
.then()
    .statusCode(200);
```

✅ Example 4: DELETE request

```
java

given()
.when()
    .delete("/users/2")
.then()
    .statusCode(204);
```

✅ Summary Table

HTTP Method	REST Assured Usage	Purpose
GET	`.get("/path")`	Retrieve data
POST	`.post("/path")`	Create new resource
PUT	`.put("/path")`	Update entire resource
PATCH	`.patch("/path")`	Update part of a resource
DELETE	`.delete("/path")`	Remove a resource

Let me know if you'd like a hands-on mini project using these REST calls with `given()` and `when()` in a real REST Assured test.

✅ `then()` – Validate the Response in REST Assured

The **`then()`** method in REST Assured is used to **assert, validate, or verify** the response returned from an API call made using **`when()`**.

🔹 Purpose of `then()`

- Check **status codes**
- Validate **response body values**
- Assert **headers**
- Use **response matchers**
- Log the response

✅ Basic Structure

```
java

given()
    // Pre-conditions
.when()
    .get("/api")
.then()
    .statusCode(200)
    .body("key", equalTo("value"))
    .header("Content-Type", equalTo("application/json"));
```

✅ Common Assertions Using `then()`

Validation Type	Method Used	Example
Status Code	**`.statusCode()`**	**`statusCode(200)`**
Response Body	**`.body()`**	**`body("name", equalTo("John"))`**
Response Header	**`.header()`**	**`header("Content-Type", "application/json")`**
Logging	**`.log().all()`**	Logs full response
JSON Path Expressions	**`.body("data.id", equalTo(1))`**	Validate nested JSON fields

✅ Example 1: Validate status code

```
java

given()
.when()
    .get("/users/2")
.then()
    .statusCode(200);
```

✅ Example 2: Validate JSON response body

```
java


given()
.when()
    .get("/users/2")
.then()
    .statusCode(200)
    .body("data.first_name", equalTo("Janet"));
```

✅ Example 3: Validate response header

```
java


given()
.when()
    .get("/users/2")
.then()
    .header("Content-Type", equalTo("application/json; charset=utf-8"));
```

✅ Example 4: Log response

```
java


given()
.when()
    .get("/users/2")
.then()
    .log().all();
```

✅ Example with Full Flow

```
java


given()
    .contentType(ContentType.JSON)
.when()
    .get("/users/2")
.then()
    .statusCode(200)
    .body("data.email", equalTo("janet.weaver@reqres.in"))
    .log().body();
```

✅ Use with Hamcrest Matchers (Optional but Powerful)

```
java


import static org.hamcrest.Matchers.*;

.body("data.id", equalTo(2))
.body("support.url", containsString("https"))
```

Would you like to set up a reusable validation method or assert multiple responses together in a framework setup?

✅ Understanding BDD Style Syntax in REST Assured

REST Assured supports **BDD (Behavior-Driven Development)** syntax using the chain:
👉 `given()` → `when()` → `then()`
This structure makes your test cases **readable, structured, and expressive**.

🔹 What is BDD Style?

BDD is a development approach that encourages writing test cases in a way **non-technical stakeholders** can also understand.
In REST Assured, BDD syntax reflects this natural language:

BDD Keyword	REST Assured Function	Description
`Given`	`given()`	Set up the preconditions
`When`	`when()`	Perform the actual API action
`Then`	`then()`	Validate the response

✅ Example in REST Assured

```
java

given()
    .baseUri("https://reqres.in/api")
    .header("Content-Type", "application/json")
    .body("{ \"name\": \"John\", \"job\": \"Engineer\" }")
.when()
    .post("/users")
.then()
    .statusCode(201)
    .body("name", equalTo("John"))
    .log().all();
```

🔍 Explanation of the Above Code

```
java

given()  // Precondition setup
    .baseUri("https://reqres.in/api")   // Base URL
    .header("Content-Type", "application/json")  // Request Header
    .body("{ \"name\": \"John\", \"job\": \"Engineer\" }")  // Request Payload

.when()  // Action
    .post("/users")  // API call

.then()  // Validation
    .statusCode(201)  // Response Code
    .body("name", equalTo("John"))  // JSON body validation
    .log().all();  // Logs full response
```

✅ Benefits of BDD Syntax in REST Assured

- 🔸 **Readable**: Easy to understand and follow the test steps.

- 🔸 **Maintainable**: You can separate setup, execution, and assertions clearly.
- 🔸 **Scalable**: Fits well into larger test frameworks using TestNG, JUnit, or Cucumber.

✅ BDD vs Non-BDD in REST Assured

BDD Style (Recommended)	Non-BDD Style (Also Valid)
`given().when().then()`	`RestAssured.given().get().then()`
More expressive and natural	More compact but less readable
Used in teams favoring behavior-driven approach	Used in quick or internal testing

---

🔧 Example with Cucumber + REST Assured (Advanced BDD)

```
gherkin

Scenario: Create a new user
  Given I have the user payload
  When I send a POST request to "/users"
  Then I should receive status code 201
  And the response should contain name "John"
```

This can be mapped in step definitions with `given()`, `when()`, `then()`.

Would you like a working BDD-style mini project with Cucumber + REST Assured next?