Day 1 - Client Server Architecture

A diagram of a computer network

AI-generated content may be incorrect.

**What is client ?**

A client is

- Computer

- Mobile

- Host

That sends request to server fro any service, data using http Protocol (using URL) and receive a Response.

**What is server?**

A server is Remote Machine that receive request from client and server them with appropriate information as Response using HTTP/s

**What is the client Server model?**

Diagram of a computer network

AI-generated content may be incorrect.

**What Happens when we type google.com ?**

**A close-up of a white background

AI-generated content may be incorrect.**

**Assignment :**

**A close up of a white background

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Goto URL –** [**https://mxtoolbox.com/DNSLookup.aspx and type google.com**](https://mxtoolbox.com/DNSLookup.aspx%20and%20type%20google.com) **as Domain name**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Day 2 : WebAuthentication & Cookies**

**What is Authentication?**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Cookies:**

**Install extension in google chrome (editThisCookies)**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**JSON Basics :**

**A screenshot of a white box

AI-generated content may be incorrect.**

**JSON Data Types.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**JSON Stringify()**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**JSON.parse()**

**A screenshot of a computer

AI-generated content may be incorrect.**

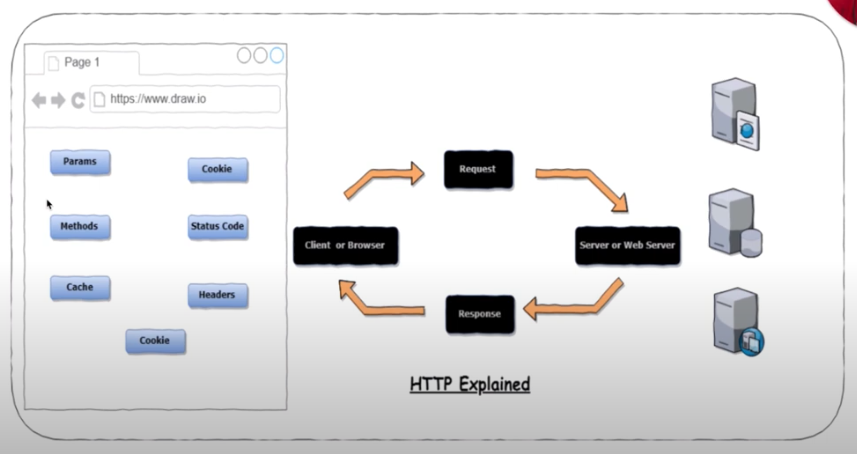
**Goto url -** [**https://objgen.com/json?demo=true**](https://objgen.com/json?demo=true) **and open developer tools - console , check all the JSON functions.**

[**https://jsoneditoronline.org**](https://jsoneditoronline.org)

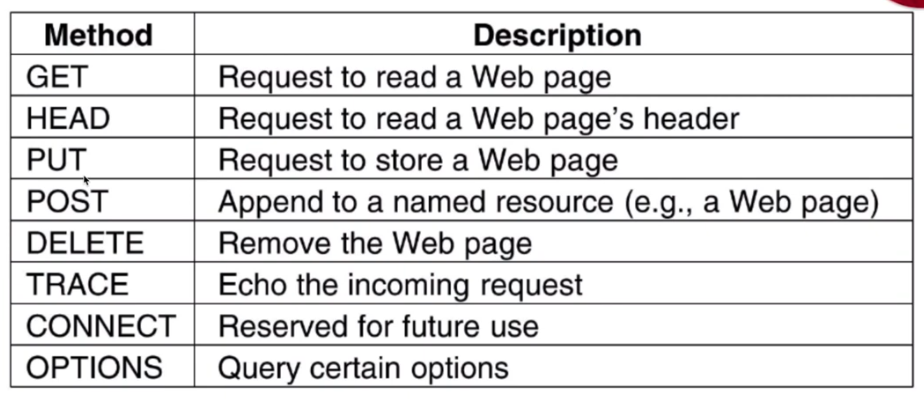
**HTTP Basics – HyperText transfer Protocol**

**A close-up of a computer screen

AI-generated content may be incorrect.**

****

**HTTP Methods**

****

**HTTP Status Codes**

**A list of messages on a white background

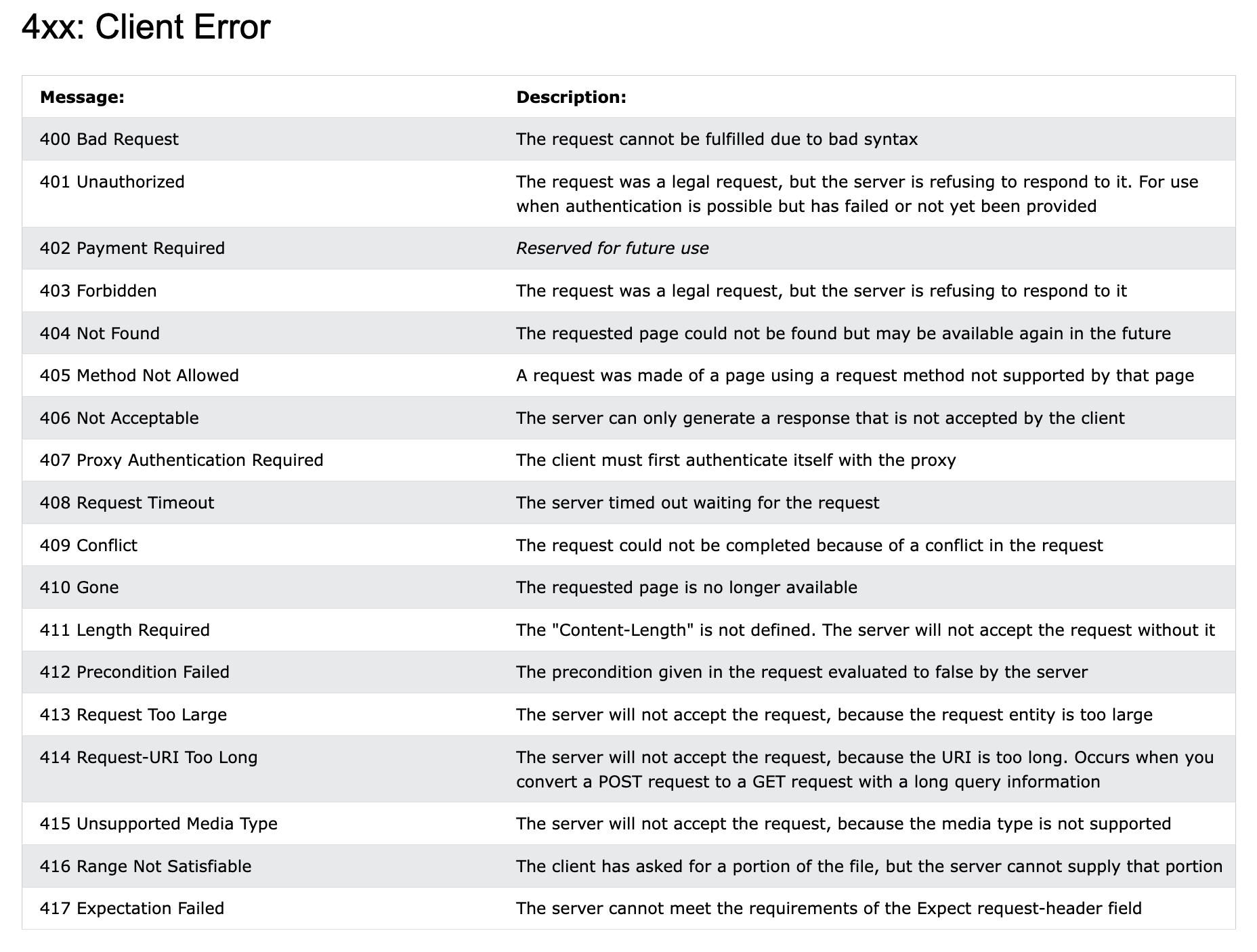
AI-generated content may be incorrect.**

[**https://en.wikipedia.org/wiki/List\_of\_HTTP\_status\_codes**](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

**A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer error message

AI-generated content may be incorrect.**

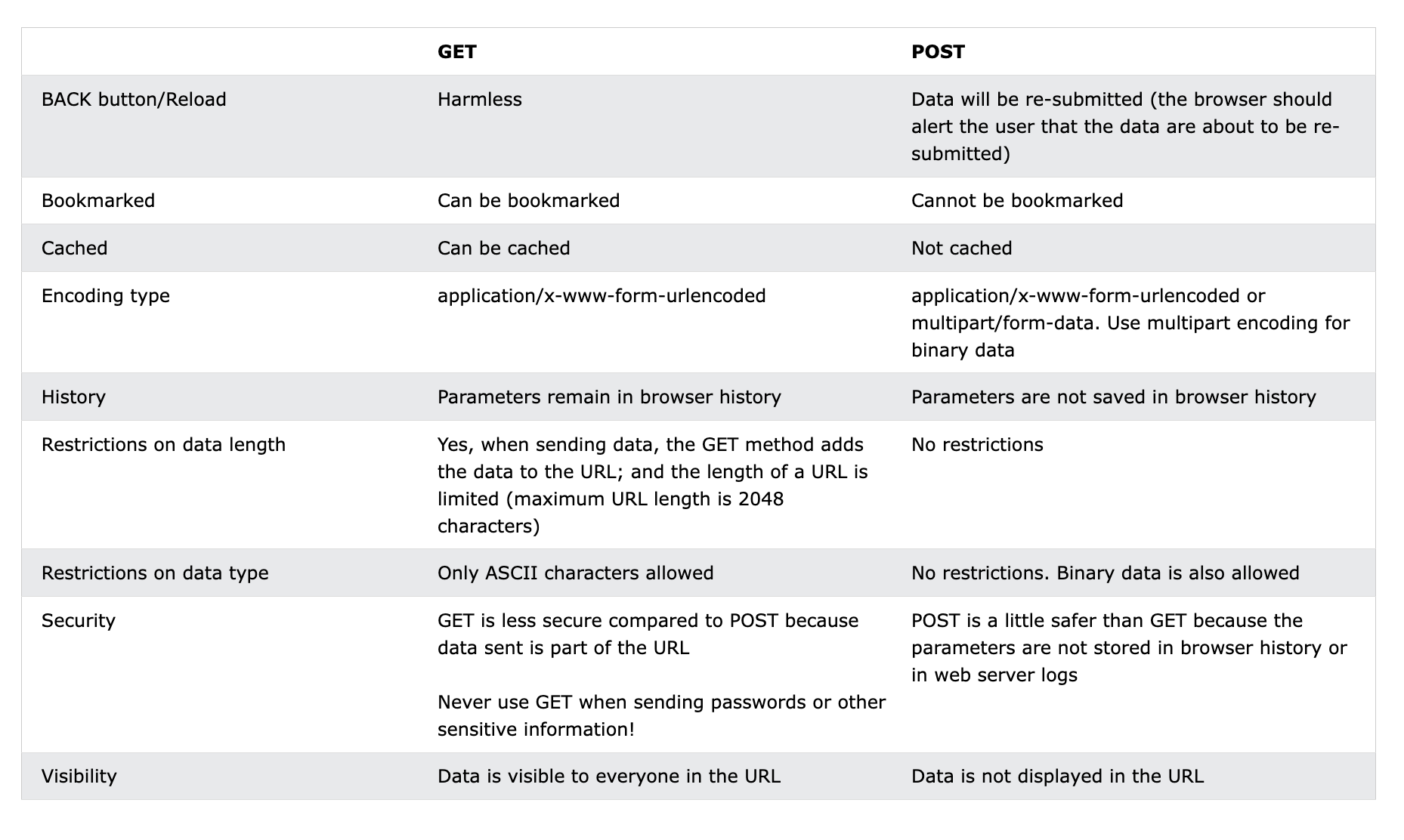
**HTTP Methods –**

**A screenshot of a web page

AI-generated content may be incorrect.**

**A screenshot of a web page

AI-generated content may be incorrect.**

****

**A close up of a text

AI-generated content may be incorrect.A white background with black text

AI-generated content may be incorrect.**

**A black text on a white background

AI-generated content may be incorrect.**

**A screenshot of a white background

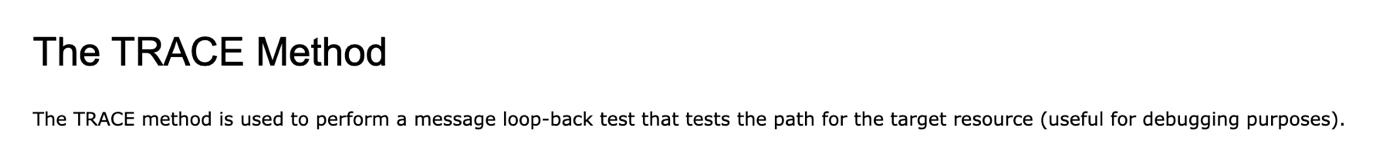
AI-generated content may be incorrect.**

**A black text on a white background

AI-generated content may be incorrect.A black text on a white background

AI-generated content may be incorrect.A white background with black text

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.**

**Web Services**

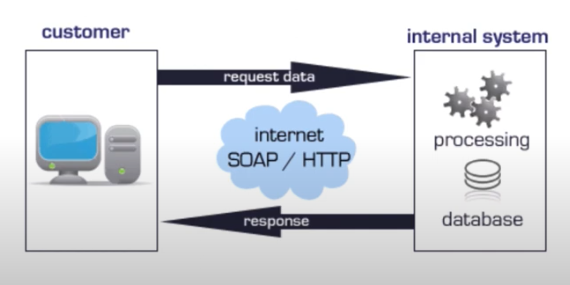
**What is Web Service?**

**A service offered by an electronic device to another electronic device, communicating with each other via the WWW (World Wide Web)**

**Or**

**They provide a common platform that allows multiple applications built on various programming languages to have the ability to commuinicate with each other.**

**The main component of a web service is the data which is transferred between client and the server and that is XML.**

****

Server Hosting Service

Client

**Components of a Web Service –**

The basic web services platform is xml + http. All the standard web services work using following components –

SOAP – simple Object Access Protocol

UDDI – Universal Description, Discovery and Integration

WSDL – Web Services Description Language

**A screenshot of a computer

AI-generated content may be incorrect.**

**How Does Web Service Work?**

A web service enables communication among various applications by using open standards such as HTML , XML , WSDL, and SOAP. A web service takes the help of –

* XML to tag the data
* SOAP to transfer a message
* WSDL to describe the availability of service.

**Benefits of Web Services**

**Different Web Services Protocols are**

SOAP

REST

WSDL

**What is an API and Types of API and API Testing?**

API is acronym for Application Programming Interface. It’s kind of intermediary that allows two applications to talk to each other.

They can be set of subroutine definitions , protocols

API can be web-based system, operating system, database system, computer hardware or software library.

A screenshot of a cartoon character

AI-generated content may be incorrect.

🔗 [REST Assured Complete Basic Example by Elias Nogueira](https://github.com/eliasnogueira/restassured-complete-basic-example)

<https://github.com/eliasnogueira/restassured-complete-basic-example.git>

This repo includes:

* Maven setup
* JUnit 5
* Allure reporting
* Test Data Factory
* Request/Response specs
* Contract, functional, and E2E tests
* GitHub Actions CI pipeline

1. Project Setup
   1. Create Maven project in your IDE (Intellij or Eclipse)
   2. Add Dependencies in pom.xml file

<dependencies>

    <dependency>

        <groupId>io.rest-assured</groupId>

        <artifactId>rest-assured</artifactId>

        <version>5.3.0</version>

        <scope>test</scope>

    </dependency>

    <dependency>

        <groupId>org.junit.jupiter</groupId>

        <artifactId>junit-jupiter</artifactId>

        <version>5.9.2</version>

    </dependency>

    <dependency>

        <groupId>com.fasterxml.jackson.core</groupId>

        <artifactId>jackson-databind</artifactId>

        <version>2.15.0</version>

    </dependency>

    <dependency>

        <groupId>io.qameta.allure</groupId>

        <artifactId>allure-rest-assured</artifactId>

        <version>2.21.0</version>

    </dependency>

</dependencies>

\*Note: Update dependency version as latest one.

1. Framework Structure

src/

├── main/

│ └── java/

│ └── config/ # Configuration classes

│ └── model/ # POJOs for request/response

│ └── utils/ # Utility classes

├── test/

│ └── java/

│ └── base/ # Base test setup

│ └── specs/ # Request/Response specs

│ └── tests/ # Test classes

│ └── datafactory/ # Fake data generation

└── resources/

└── api.properties # Configurable properties

└── schemas/ # JSON schemas for contract testing

1. Base Test Setup

Create BaseTest class to initialize common configurations

@BeforeAll

public static void setup() {

    RestAssured.baseURI = "https://api.example.com";

    RestAssured.basePath = "/v1";

}

1. Request & Response Specificattions.

Use RequestSpecBuider and ResponseSpecBuider

RequestSpecification requestSpec = new RequestSpecBuilder()

    .setContentType(ContentType.JSON)

    .addHeader("Authorization", "Bearer " + token)

    .build();

ResponseSpecification responseSpec = new ResponseSpecBuilder()

    .expectStatusCode(200)

    .expectContentType(ContentType.JSON)

    .build();

1. Test Scenario :
2. Health Check

given().spec(requestSpec)

.when().get("/health")

.then().spec(responseSpec);

1. CRUD Operations

* **GET**: Fetch data
* **POST**: Create resource
* **PUT/PATCH**: Update resource
* **DELETE**: Remove resource

1. Authentication

* Token-based (Bearer)
* Basic Auth
* OAuth2

1. Schema Validation

.then().body(matchesJsonSchemaInClassPath(“user-schema.json”));

1. Serialization/Deserialization

Use POJOs with Jackson:

User user = new User("John", "Doe");

given().body(user).post("/users");

1. Data-Driven Testing

Use JUnit 5 @ParameterizedTest or TestNG @DataProvider.

1. Contract Testing

Validate response structure using JSON Schema

1. End-to-End Testing

Simulate full user journeys across multiple endpoints

1. Reporting

Use Allure for beautiful test reports

Mvn clean test

Mvn allure:serve

1. CI/CD Integration

Integrate with GITHub Actions or Jenkins:

- name: Run API Tests

  run: mvn test -Dgroups="functional"

**Comprehensive Framework with CI/CD**

🔗 [ashchupliak/java-rest-assured-api-testing-framework](https://github.com/ashchupliak/java-rest-assured-api-testing-framework" \t "_blank)  
Features:

* REST Assured + TestNG + Maven
* Allure & Extent Reports
* Data-driven testing (Excel, JSON, DB)
* Docker support
* Jenkins & GitHub Actions CI/CD
* Retry logic, Slack notifications, and more

1

**Simplified Framework with Jenkins Integration**

🔗 [omprakashchavan01/RestAssuredTestNGFramework](https://github.com/omprakashchavan01/RestAssuredTestNGFramework)  
Features:

* REST Assured + TestNG + Maven
* Allure Reports
* Lombok, POJOs, Builder Pattern
* Jenkins integration
* **Step-by-Step Framework Setup**

**Project Initialization**

* Create a Maven project.
* Add dependencies in pom.xml:

<dependencies>

    <dependency>

        <groupId>io.rest-assured</groupId>

        <artifactId>rest-assured</artifactId>

        <version>5.3.0</version>

    </dependency>

    <dependency>

        <groupId>org.testng</groupId>

        <artifactId>testng</artifactId>

        <version>7.9.0</version>

    </dependency>

    <dependency>

        <groupId>io.qameta.allure</groupId>

        <artifactId>allure-testng</artifactId>

        <version>2.21.0</version>

    </dependency>

</dependencies>

**2. Framework Structure**

src/

├── main/java/com/framework/

│ ├── config/ # Environment configs

│ ├── core/ # BaseTest, APIClient

│ ├── utils/ # ExcelUtils, JsonUtils

│ ├── models/ # POJOs

│ ├── reporting/ # Allure/Extent integration

│ └── listeners/ # Retry, SlackNotifier

├── test/java/com/tests/

│ ├── api/ # API test classes

│ ├── integration/ # Workflow tests

│ ├── performance/ # Response time tests

│ └── smoke/ # Health checks

resources/

├── config/ # application.properties

├── schemas/ # JSON schemas

└── testng.xml # TestNG suite config

3. TestNG Integration

Use @Test, @BeforeSuite, @DataProvider annotations

@Test(dataProvider = "userData")

public void createUserTest(String name, String email) {

    User user = new User(name, email);

    given().body(user)

           .when().post("/users")

           .then().statusCode(201);

}

@DataProvider(name = "userData")

public Object[][] getUserData() {

    return new Object[][] {

        {"John", "john@example.com"},

        {"Jane", "jane@example.com"}

    };

}

4. CI/CD Integration –

Jenkin – Add a Jenkinfile

pipeline {

    agent any

    stages {

        stage('Build') {

            steps {

                sh 'mvn clean install'

            }

        }

        stage('Test') {

            steps {

                sh 'mvn test'

            }

        }

        stage('Report') {

            steps {

                allure includeProperties: false, jdk: '', results: [[path: 'allure-results']]

            }

        }

    }

}

**GitHub Actions**

* Add .github/workflows/ci-cd.yml:

name: API Test CI

on: [push, pull\_request]

jobs:

  test:

    runs-on: ubuntu-latest

    steps:

      - uses: actions/checkout@v3

      - name: Set up JDK

        uses: actions/setup-java@v3

        with:

          java-version: '17'

      - name: Run Tests

        run: mvn clean test

      - name: Generate Allure Report

        run: mvn allure:report

**Advanced Features**

* **Retry Logic**: Retry failed tests using RetryAnalyzer.
* **Slack Notifications**: Send test results to Slack.
* **Docker Support**: Run tests in containers.
* **Database Validation**: Compare API response with DB records.
* **File Upload/Download**: Test multipart endpoints.
* **Performance Testing**: Validate response times.

BDD Framework :

**Sample GitHub Repository**

**🔗**[**Prabhu98/Rest-Assured-cucumber\_testng\_java**](https://github.com/Prabhu98/Rest-Assured-cucumber_testng_java)

<https://github.com/Prabhu98/Rest-Assured-cucumber_testng_java.git>

This is a reusable BDD automation framework that integrates:

* **Cucumber (BDD)**
* **TestNG**
* **REST Assured**
* **Allure Reports**
* **Jenkins CI**
* **Page Object Model**
* **Mobile & Web support (Appium, Selenium)**
* **Step-by-Step BDD Framework Setup**
* **1. Project Setup**

Use Maven and add the following dependencies in pom.xml:

<dependencies>

    <dependency>

        <groupId>io.rest-assured</groupId>

        <artifactId>rest-assured</artifactId>

        <version>5.3.0</version>

    </dependency>

    <dependency>

        <groupId>io.cucumber</groupId>

        <artifactId>cucumber-java</artifactId>

        <version>7.11.2</version>

    </dependency>

    <dependency>

        <groupId>io.cucumber</groupId>

        <artifactId>cucumber-testng</artifactId>

        <version>7.11.2</version>

    </dependency>

    <dependency>

        <groupId>org.testng</groupId>

        <artifactId>testng</artifactId>

        <version>7.9.0</version>

    </dependency>

    <dependency>

        <groupId>io.qameta.allure</groupId>

        <artifactId>allure-testng</artifactId>

        <version>2.21.0</version>

    </dependency>

</dependencies>

Folder Structure

src/

├── main/java/

│ └── utils/ # Utility classes

│ └── config/ # Config readers

│ └── models/ # POJOs

├── test/java/

│ ├── stepDefinitions/ # Step implementations

│ ├── runners/ # TestNG runner

│ └── hooks/ # Before/After hooks

├── test/resources/

│ └── features/ # .feature files

│ └── schemas/ # JSON schemas

└── testng.xml # TestNG suite

**Feature File Example**

Feature: Book Management

  Background:

    Given I am an authorized user

  Scenario: Add and remove a book

    Given A list of books are available

    When I add a book to my reading list

    Then The book is added

    When I remove a book from my reading list

    Then The book is removed

4 Step Definitions

@Given("I am an authorized user")

public void iAmAnAuthorizedUser() {

    RestAssured.baseURI = "https://bookstore.toolsqa.com";

    RequestSpecification request = RestAssured.given();

    request.header("Content-Type", "application/json");

    response = request.body("{ \"userName\":\"test\", \"password\":\"Test@123\" }")

                      .post("/Account/v1/GenerateToken");

    token = JsonPath.from(response.asString()).get("token");

}

Test Runner (TestNG + cucumber)

@CucumberOptions(

    features = "src/test/resources/features",

    glue = {"stepDefinitions", "hooks"},

    plugin = {"pretty", "io.qameta.allure.cucumber7jvm.AllureCucumber7Jvm"}

)

public class TestRunner extends AbstractTestNGCucumberTests {}

**Allure Reporting**

Generate reports with:

Mvn clean test

Mvn allure:report

Mvn allure:serve

CI/CD Inttegration

Githuib actions

**YAML file**

name: BDD API Tests

on: [push]

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Java

uses: actions/setup-java@v3

with:

java-version: '17'

- name: Run Tests

run: mvn clean test

- name: Generate Allure Report

run: mvn allure:report

jenkin pipelines

pipeline {

    agent any

    stages {

        stage('Test') {

            steps {

                sh 'mvn clean test'

            }

        }

        stage('Allure Report') {

            steps {

                allure includeProperties: false, results: [[path: 'allure-results']]

            }

        }

    }

}

**Features You Can Test**

| **Feature** | **Supported** |
| --- | --- |
| CRUD Operations | ✅ |
| Auth (Basic, Bearer, OAuth) | ✅ |
| JSON Schema Validation | ✅ |
| Data-Driven Testing | ✅ |
| File Upload/Download | ✅ |
| Retry Logic | ✅ |
| Parallel Execution | ✅ |
| CI/CD Integration | ✅ |
| Allure Reporting | ✅ |
| DB Validation (Optional) | ✅ |

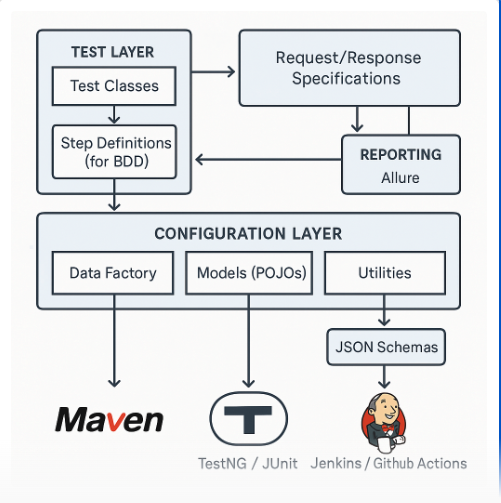
**Framework Architecture Diagram**

Here is the architecture diagram for the **REST Assured Java API Testing Framework** inspired by Elias Nogueira’s design:

![REST Assured Java Framework Architecture]

This diagram includes:

* **Test Layer** (TestNG or JUnit)
* **Step Definitions** (for BDD)
* **Request/Response Specs**
* **Configuration Layer**
* **Data Factory**
* **POJOs (Models)**
* **Utilities**
* **JSON Schemas**
* **Allure Reporting**
* **CI/CD Integration** (Jenkins, GitHub Actions)



**Custom Starter Template ZIP**

You can download your **REST Assured Java API Testing Framework Starter Template** here:

📦 [Download restassured-api-framework.zip](blob:https://m365.cloud.microsoft/31e4f502-4da9-4658-b42f-35702656e416)

This includes:

* Maven project structure
* Sample test class
* Request/response specifications
* Configuration and POJOs
* JSON schema
* Allure integration

**🧭 Framework Architecture Diagram**

Here is the architecture diagram for the framework based on Elias Nogueira’s design:

**Step 1: Import Project into IntelliJ IDEA**

1. **Open IntelliJ IDEA**
2. Click **File > Open**
3. Select the folder restassured-api-framework (from the ZIP you downloaded)
4. IntelliJ will detect it as a Maven project and import dependencies automatically
5. If prompted, click **Enable Auto-Import**

**🧪 Step 2: Add TestNG Support**

Update pom.xml:

<dependency>

  <groupId>org.testng</groupId>

  <artifactId>testng</artifactId>

  <version>7.9.0</version>

  <scope>test</scope>

</dependency>

Create testing.xml file

<suite name="API Test Suite">

  <test name="User API Tests">

    <classes>

      <class name="tests.UserApiTest"/>

    </classes>

  </test>

</suite>

Modify your test class to use TestNG

import org.testng.annotations.Test;

@Test

public class UserApiTest {

    public void testGetUsers() {

        // same test logic

    }

}

**Step 3: Add Cucumber BDD**

Add dependencies:

dependency>

  <groupId>io.cucumber</groupId>

  <artifactId>cucumber-java</artifactId>

  <version>7.11.2</version>

</dependency>

<dependency>

  <groupId>io.cucumber</groupId>

  <artifactId>cucumber-testng</artifactId>

  <version>7.11.2</version>

</dependency>

1. Create folders:

src/test/java/

├── stepdefinitions/

├── runners/

src/test/resources/

└── features/

1. Sample feature file (user.feature)

Feature: User API

  Scenario: Get all users

    When I send a GET request to "/users"

    Then the response status code should be 200

1. Step definitions

@When("I send a GET request to {string}")

public void sendGetRequest(String endpoint) {

    response = given().get(endpoint);

}

@Then("the response status code should be {int}")

public void verifyStatusCode(int statusCode) {

    response.then().statusCode(statusCode);

}

1. Runner class :

@cucumberOptions {

features = "src/test/resources/features",

  glue = "stepdefinitions",

  plugin = {"pretty", "io.qameta.allure.cucumber7jvm.AllureCucumber7Jvm"}

)

public class TestRunner extends AbstractTestNGCucumberTests {}

**Step 4: Add Docker Support**

Create a Dockerfile:

FROM maven:3.9.3-eclipse-temurin-17

WORKDIR /app

COPY . .

RUN mvn clean install

CMD ["mvn", "test"]

Build and run

docker build -t restassured-api-tests .

docker run restassured-api-tests

**Step 5: Add Database Validation (Optional)**

1. Add JDBC dependency:

<dependency>

  <groupId>mysql</groupId>

  <artifactId>mysql-connector-java</artifactId>

  <version>8.0.33</version>

</dependency>

1. Create a DB utility class:

public class DBUtils {

    public static Connection getConnection() throws SQLException {

        return DriverManager.getConnection("jdbc:mysql://localhost:3306/testdb", "user", "pass");

    }

}

1. Use in a test

ResultSet rs = DBUtils.getConnection().createStatement().executeQuery("SELECT \* FROM users");

<https://github.com/ashchupliak/java-rest-assured-api-testing-framework.git>