Test Automation Framework

Course Content

- What is a Test Automation Framework
- Types of Test Automation Frameworks
- Key Components of Test Automation Framework
- Steps to Build Test Automation Framework from Scratch

Building Test Automation Framework from Scratch

What is a Test Automation Framework?

- A **test automation framework** provides a structured approach to writing and managing automated test scripts.
- It integrates **tools, libraries**, and **best practices** to ensure consistency, scalability, and maintainability in test automation.

Types of Test Automation Frameworks

1. Linear (Record and Playback)

- Simplest framework, often used by beginners.
- Test scripts are recorded and executed in a sequential manner.
  - **Pros**: Easy to implement.
  - **Cons**: Hard to maintain, lacks scalability.

2. Modular Framework

- Test scripts are divided into **independent modules**.
- Reusable functions are created for each module.
  - **Pros**: Promotes reusability.
  - **Cons**: Requires upfront planning and coding.

3. Data-Driven Framework

- Test data is separated from the test scripts, often stored in files like **Excel, CSV, or databases**.
- Test scripts run for multiple data sets.
  - **Pros**: Allows testing with diverse data inputs.
  - **Cons**: Complex setup and maintenance.

Types of Test Automation Frameworks Contd.

4. Keyword-Driven Framework

- Uses **keywords** (specific actions) to describe test steps.
- Testers write high-level test cases using predefined keywords.
  - **Pros**: Non-technical testers can write tests.
  - **Cons**: Initial implementation can be time-intensive.

5. Hybrid Framework

- Combines features of **data-driven**, **keyword-driven**, and **modular frameworks**.
- Promotes flexibility and scalability.
  - **Pros**: Adaptable to complex testing needs.
  - **Cons**: Requires advanced planning and expertise.

Types of Test Automation Frameworks Contd.

Framework	Description	Advantages	Disadvantages	Use Case
Linear Framework	Sequentially records and plays back test steps.	Easy to implement and execute.	Not maintainable or scalable.	Quick tests for small applications.
Modular Framework	Divides the application into independent modules, with reusable functions for each module.	Promotes reusability.	Requires planning and setup effort.	Applications with repetitive workflows.
Data-Driven Framework	Separates test logic and data, allowing testing with multiple data sets.	Tests with diverse inputs.	Complex initial setup.	Applications requiring extensive input tests.
Keyword-Driven Framework	Uses keywords for actions, allowing non-technical testers to write tests.	Simplifies test case writing for non-coders.	Time-intensive initial keyword design.	Projects with non-technical team members.
Hybrid Framework	Combines features of multiple frameworks for flexibility and scalability.	Highly adaptable and scalable.	Requires expertise and planning.	Complex projects with diverse testing needs.

Key Components of a Test Automation Framework

1. Test Scripts

- Contains test logic written in a programming language (e.g., Java, Python).

2. Test Data Management

- External files for test data (e.g., Excel, JSON).

3. Reporting Mechanism

- Tools like **Extent Reports**, **Allure**, or custom solutions for detailed test reports.

4. Logging

- Logs generated during execution for debugging (e.g., using **Log4j**, **Slf4j**).

5. Configuration Files

- Centralized properties for URLs, credentials, and environment-specific configurations.

6. Utilities and Libraries

- Reusable components for handling common tasks like API requests, database operations, etc.

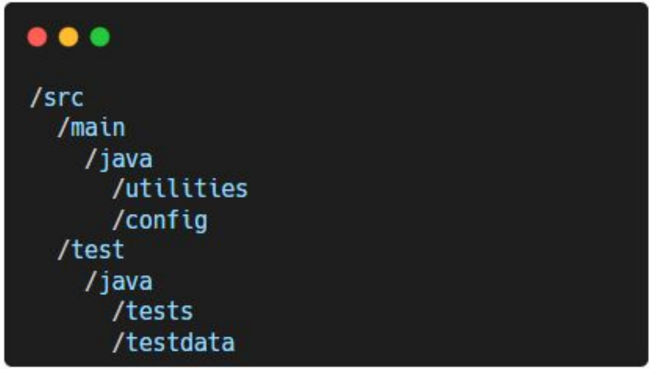
Steps to Build a Hybrid Framework from Scratch

Step 1: Define Goals

- Determine the testing scope (e.g., API, UI, or both).
- Identify the tools and libraries (e.g., **REST Assured**, **TestNG**, **Postman**, **Extent Reports**).

Step 2: Set Up the Project Structure

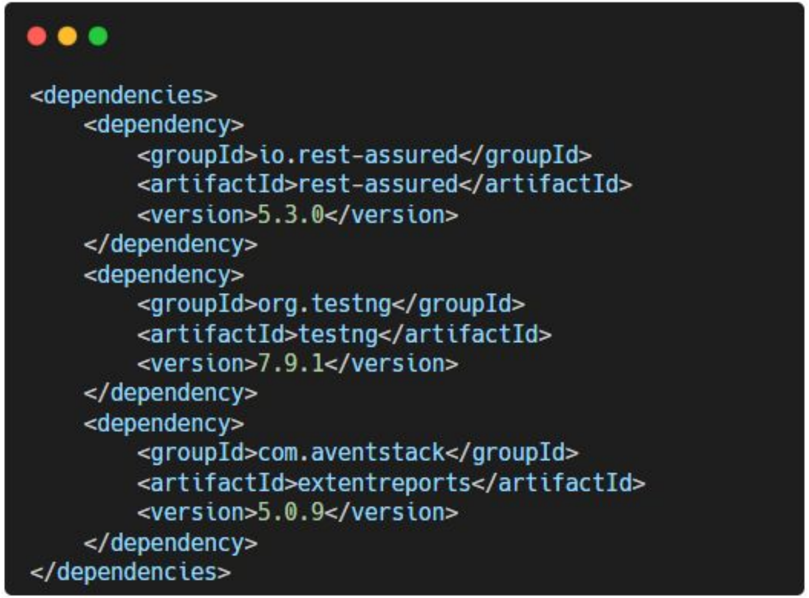
- Follow a standard folder structure:



```
/src
  /main
    /java
      /utilities
      /config
  /test
    /java
      /tests
      /testdata
```

Step 3: Configure Build Tool

- Use **Maven** or **Gradle** for dependency management.
- Example Maven dependencies:



```
<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>7.9.1</version>
    </dependency>
    <dependency>
        <groupId>com.aventstack</groupId>
        <artifactId>extentreports</artifactId>
        <version>5.0.9</version>
    </dependency>
</dependencies>
```

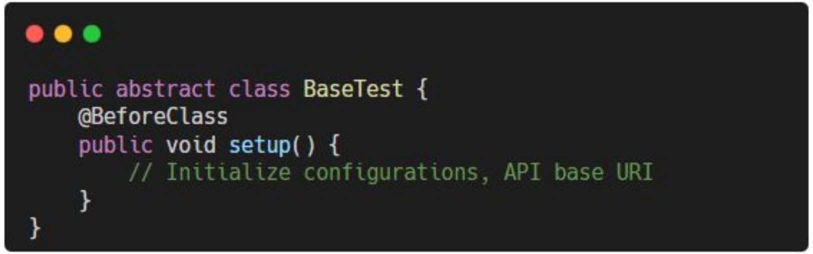
Steps to Build a Hybrid Framework from Scratch

Step 4: Create Utility Classes

- Example utilities:
  - **APIUtils**: For making API requests.
  - **ExcelUtils**: For reading test data.
  - **ConfigReader**: For reading properties files.

Step 5: Write Base Test Class

- Abstract class containing common setups like configuration loading or authentication setup.
- Example:



```
public abstract class BaseTest {
    @BeforeClass
    public void setup() {
        // Initialize configurations, API base URI
    }
}
```

Step 6: Add Reporting and Logging

- Integrate **Extent Reports** for reporting.
- Use **Log4j** or **Slf4j** for logs.

Step 7: Write Test Cases

- Use TestNG or JUnit for organizing and executing test cases.

Step 8: Parameterize Test Data

- Load test data from external sources like Excel or JSON.

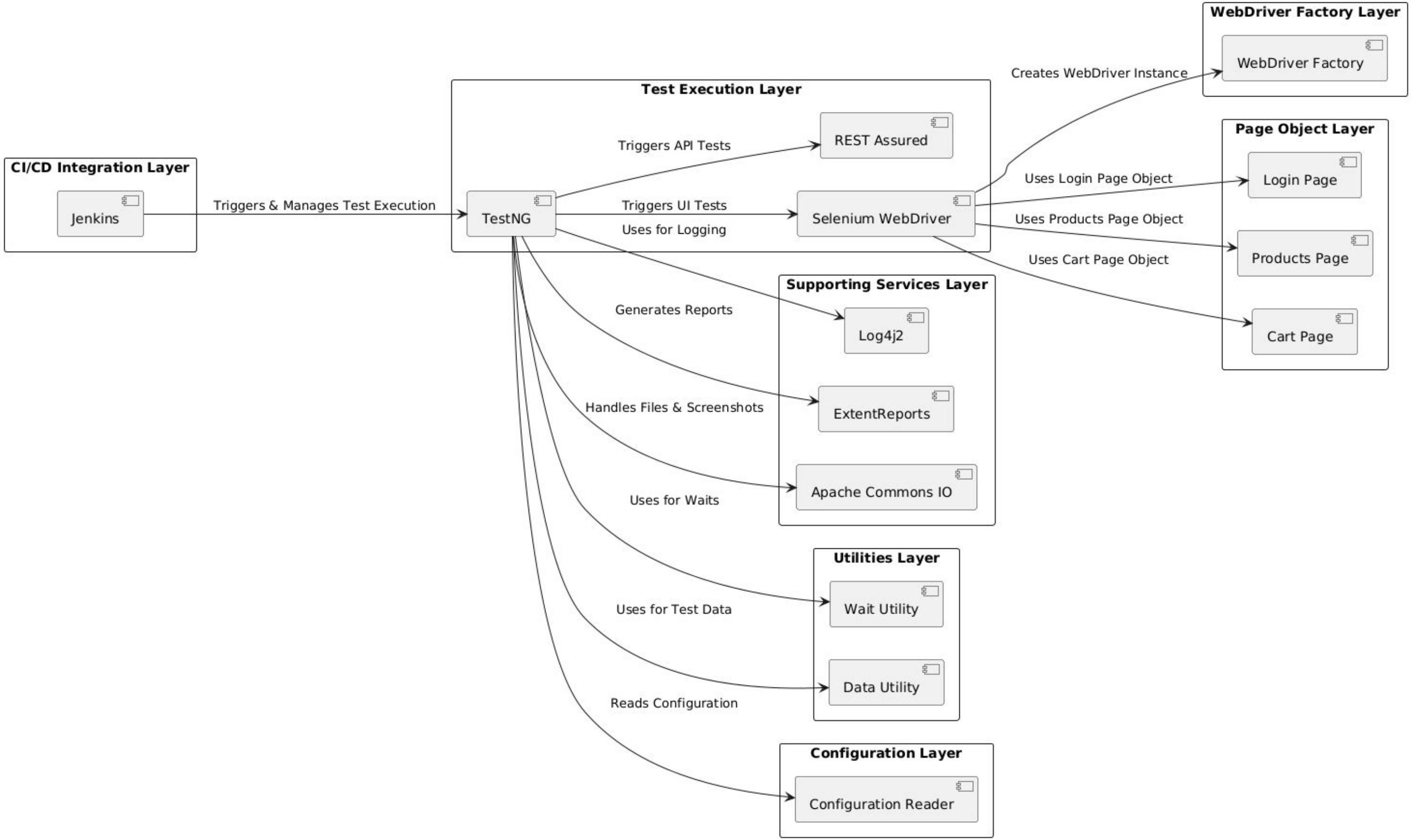
Step 9: Integrate with CI/CD

- Use tools like **Jenkins**, **GitHub Actions**, or **GitLab CI** to automate execution.

Step 10: Maintenance

- Regularly review and update the framework to accommodate changes in the API or testing requirements.

Hybrid Framework Component Diagram


CI/CD Integration Layer
Jenkins
Triggers & Manages Test Execution
Test Execution Layer
TestNG
Triggers API Tests
REST Assured
Triggers UI Tests
Selenium WebDriver
Uses for Logging
WebDriver Factory Layer
WebDriver Factory
Creates WebDriver Instance
Page Object Layer
Uses Login Page Object
Login Page
Uses Products Page Object
Products Page
Uses Cart Page Object
Cart Page
Supporting Services Layer
Generates Reports
Log4j2
ExtentReports
Handles Files & Screenshots
Apache Commons IO
Uses for Waits
Utilities Layer
Wait Utility
Uses for Test Data
Data Utility
Reads Configuration
Configuration Layer
Configuration Reader