The framework follows a modular approach, adhering to the Page Object Model (POM) for better maintainability and scalability. Each page of the application has a corresponding page object class, encapsulating the locators and methods that interact with UI elements. The framework is scalable, maintainable, and supports complex test scenarios across multiple browsers and environments.

This section provides a detailed explanation of the automation framework, its components, and step-by-step instructions on how to set up and execute the tests.

1. Framework Overview

The automation framework is built using Playwright for browser automation and TestNG for test execution and management. It follows the Page Object Model (POM) design pattern to enhance maintainability and readability and also separates test logic from page-specific logic. The framework is structured to support both functional and non-functional testing, including performance, scalability, and responsiveness tests.

Tools and Libraries

1. Playwright: A powerful browser automation library that supports multiple browsers (Chromium, Firefox, WebKit, Edge).
2. TestNG: A testing framework inspired by JUnit and NUnit, used for test configuration, parallel execution, and reporting.
3. Log4j: For logging within the framework.
4. Apache POI: For handling Microsoft Office documents, useful for data-driven testing.
5. Commons IO and Commons Lang3: Utility libraries for common tasks.
6. ExtentReports: For generating detailed and interactive test reports.
7. Maven: For project management and dependency management. Allows seamless integration with CI/CD pipelines.

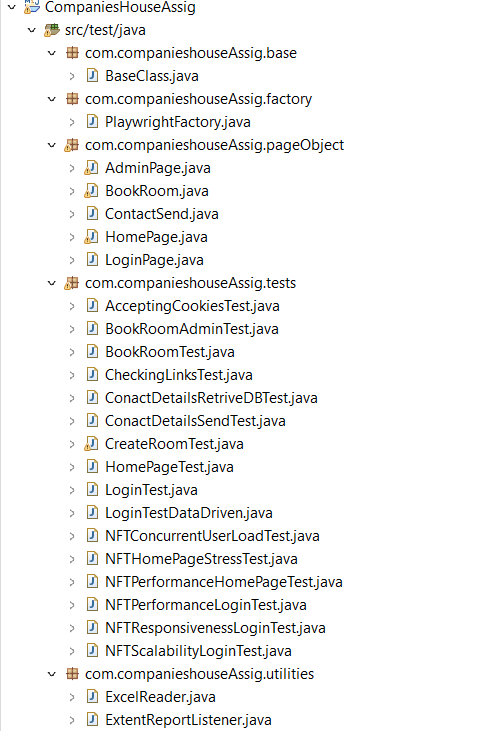
Prerequisites

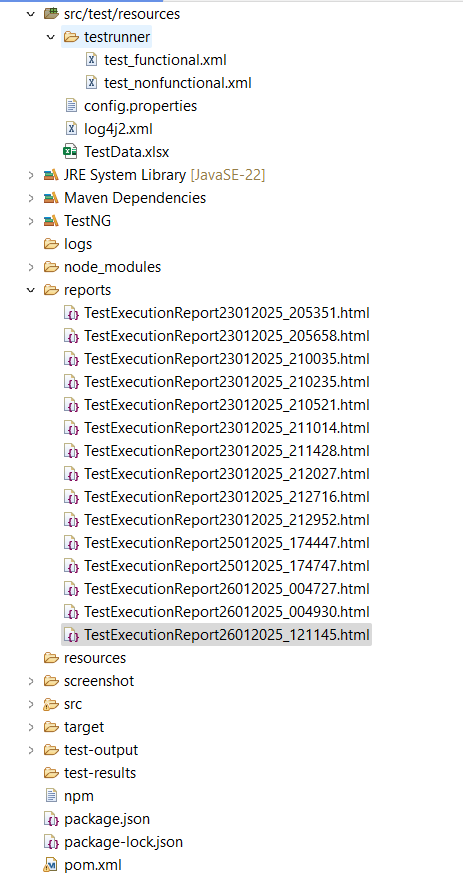
Before executing the tests, ensure the following tools and dependencies are installed:

1. Java Development Kit (JDK): Version 22 or higher.
   * Download and install from [Oracle JDK](https://www.oracle.com/java/technologies/javase-downloads.html).
   * Set up the JAVA\_HOME environment variable.
2. Maven: For dependency management and project building.
   * Download and install from [Maven](https://maven.apache.org/download.cgi).
   * Set up the MAVEN\_HOME environment variable.
3. IDE: IntelliJ IDEA or Eclipse for code editing and execution.
   * Download and install from [IntelliJ IDEA](https://www.jetbrains.com/idea/) or [Eclipse](https://www.eclipse.org/downloads/).
4. Node.js: Playwright requires Node.js for browser binaries.
   * Download and install from [Node.js](https://nodejs.org/).
5. Git: For cloning the repository (if applicable).
   * Download and install from [Git](https://git-scm.com/).

2. Folder Structure

* src/test/java: Contains the main Java source code.
  + com.companieshouseAssig.base: Base classes and utilities.
    - BaseClass.java: Initializes the browser and sets up the test environment. Contains common setup and teardown methods for initializing the browser, loading properties, and creating page object instances.
  + com.companieshouseAssig.factory: Factory classes for creating instances.
    - PlaywrightFactory.java: Manages browser and page instances. Handles the Playwright browser initialization and configuration.
  + com.companieshouseAssig.pageObject: Page classes following the POM pattern.
    - AdminPage.java, BookRoom.java, ContactSend.java, HomePage.java, LoginPage.java: Page classes for different parts of the application.
  + com.companieshouseAssig.tests: Contains test classes.
    - Various test classes like AcceptingCookiesTest.java, BookRoomTest.java, etc.
  + com.companieshouseAssig.utilities: Utility classes.
    - ExtentReportListener.java: Listener for generating ExtentReports.
    - ExcelReader.java: Data driven testing achieved using this files
* src/test/resources: Contains configuration files and resources.
  + testrunner: TestNG XML files for test execution.
    - test\_functional.xml, test\_nonfunctional.xml: Configuration files for running functional and non-functional tests.
  + config.properties: Configuration properties for the framework. Parameterization is achieved in the framework by using the config.properties file to externalize and manage environment-specific configurations. This approach allows for flexibility in defining key parameters, such as browser settings and URLs, without modifying the test code.
  + log4j2.xml: Log4j configuration file.
  + TestData.xlsx for data driven testing with username and password.
* screenshots/
  + Saves screenshots captured during test execution.
* reports/
* The test execution results will be generated in the reports folder, organized by date and time.





3. Key Components

1. BaseClass.java: Sets up the test environment before each test class execution and tears it down afterward. It initializes the browser and page objects.
2. PlaywrightFactory.java: Manages the creation and lifecycle of Playwright browser instances. It supports multiple browsers and handles browser context and page creation.
3. Page Object Classes: Each page of the application has a corresponding class that encapsulates the page's elements and actions.
4. Test Classes: Each test class corresponds to a specific functionality or scenario. They extend BaseClass and use the page objects to interact with the application.
5. ExtentReportListener.java: A TestNG listener that generates detailed test reports using ExtentReports.
6. ExcelReader.java: To implement data-driven testing using Excel in your Playwright automation framework with Java, you need a utility class (ExcelReader.java) to read data from Excel files.

4. Configuration

* Maven: The project is managed using Maven, with dependencies and plugins defined in the pom.xml file.
* TestNG XML Files: Used to configure and execute test suites. They define which tests to run and their parameters.
* Logging: Configured using Log4j for detailed logging during test execution.
* Reports: ExtentReports is used to generate comprehensive test execution reports.

Example Workflow

1. Setup: The BaseClass initializes the browser and navigates to the application URL.
2. Test Execution: Test classes use page objects to interact with the application and perform assertions.
3. Reporting: ExtentReports captures the test execution details and generates a report.
4. Teardown: The browser is closed, and resources are cleaned up after test execution.
5. Executing Tests
6. Running Functional Tests:
   * Open the project in your IDE (IntelliJ IDEA or Eclipse).
   * Navigate to src/test/resources/testrunner/test\_functional.xml.
   * Right-click the file and select Run or Debug.
7. Running Non-Functional Tests:
   * Navigate to src/test/resources/testrunner/test\_nonfunctional.xml.
   * Right-click the file and select Run or Debug.
8. TestNG Execution:
   * Alternatively, you can directly execute tests from the TestNG XML file:
9. Check Test Reports:
   * After execution, navigate to the reports/ folder to view the test execution report generated by ExtentReports.
   * Similarly TestNG reports can be viewed under test-output folder with index.html
10. Inspect Logs:
    * Check the logs for debugging and details in the log4j2.xml configured output (e.g., console or a file).
11. Specific test case

* To run specific test suites, use the TestNG XML files located in src/test/resources/testrunner:

1. Open the project in IntelliJ IDEA or Eclipse.
2. Right-click on the test\_functional.xml or test\_nonfunctional.xml file.
3. Select Run or Debug.

While executing, ensure that non-functional test cases are executed separately. Do not mix them with functional test cases, as this might cause system changes and result in test case failures.

6.Viewing Reports

After test execution, the reports are generated in the :

* ExtentReports: Located in reports/Test ExecutionReportDateand time formate.html.

Eg. TestExecutionReport26012025\_121145.html

* TestNG Reports: Located in test-output/index.html.

Open these files in a browser to view detailed test results.

7. Extending the Framework

1. Adding New Test Cases:
   * Create a new test class in the com.companieshouseAssig.tests package.
   * Instantiate the necessary page objects from the BaseClass.
   * Write your test methods annotated with @Test.
2. Adding New Pages:
   * Create a new page object class in the com.companieshouseAssig.pageObject package.
   * Define the UI elements using Playwright locators.
   * Add methods to interact with these elements.
3. Updating Configuration:
   * Modify config.properties for environment-specific settings like base URL, browser type, etc.
   * Example:
4. Improving Reporting:
   * Customize ExtentReportListener.java to include additional metrics or visualizations.
5. Data Driven Testing :

* Current Implementation: Already have a utility class (ExcelReader.java) to read data from Excel files.
* Adding More Data: Add more rows or columns to your existing Excel file (testdata.xlsx).
* Update the @DataProvider method to handle the new data.

8. Troubleshooting Tips

1. Dependency Issues:
2. Browser Not Launching:
   * Verify Playwright is correctly installed by running:
3. Test Failures:
   * Check the screenshots/ folder for captured screenshots.
   * Inspect logs to identify root causes.
4. Java Version Compatibility:
   * Ensure you are using the specified Java version in pom.xml.