**Project Overview**

This project automates the extraction of car washing service details and validates form interactions on local services websites such as Justdial. The solution is developed using Selenium WebDriver, TestNG, and Cucumber for test orchestration and behavior-driven development. The automation scope includes service discovery based on filters, error message validation during form submissions, and extraction of menu items across dynamic categories. Results are reported using structured logs and detailed test reports.

**Problem Statement**

**Discover Car Washing Services**

Objective: Display nearby car washing services filtered by the following criteria:

1. Rating greater than 4 stars
2. More than 20 customer votes
3. Sorted in descending order by rating

**Additional Functionalities:**

1. Access the "Free Listing" registration form and submit it with invalid input (e.g., incorrect phone number), capturing and displaying the corresponding warning message.
2. Navigate from the "Fitness" section to "Gym" and extract all sub-menu items into a list.

**Detailed Description**

**Requirements**

1. Access a verified directory website (e.g., justdial.com) and extract information for five local car washing services satisfying:
   * Rating > 4
   * Customer votes > 20
   * Sorted from highest to lowest rating
2. Automate the submission of the “Free Listing” form using one invalid field and capture the resulting error.
3. Traverse the “Fitness” → “Gym” category hierarchy and collect all sub-menu items for display.

**Automation Scope**

* Handle JavaScript alerts and dynamic search interactions
* Automate form submissions using diverse input elements
* Capture and verify warning or error messages
* Extract and store sub-menu items in structured collections
* Enable alternate navigation paths to target pages
* Implement smoothreturn-to-home page transitions

**Frameworks and Tools**

**Selenium WebDriver**

Core automation tool for browser-based testing.

**TestNG**

* Supports parallel execution and advanced configuration via testng.xml
* Integrates custom listeners such as CustomListener and retry logic using RetryTransformer
* Manage suite lifecycle events, including setup and teardown

**Cucumber**

* Facilitates behavior-driven development (BDD) using Gherkin syntax
* Feature files define user behaviors and acceptance criteria
* Java-based step definitions serve as implementation logic
* Integrated with TestNG to support hybrid execution

**Allure Reporting**

* Provides graphical reports including test results, error traces, and screenshots
* Displays step-level details, status, metadata, and attachments for better diagnostics
* Automatically integrates with Cucumber and TestNG test execution

**Maven**

* Manages project dependencies, lifecycle, and build configuration
* Streamlines test executions and packaging

**Apache POI**

* Used for reading and writing Excel files to handle input/output data

**Page Object Model (POM) and Page Factory**

* Promotes modularity and reusability by encapsulating web element interactions
* Page Factory simplifies element initialization and reduces code overhead

**Steps to Run the Project**

**Browser Configuration**

**Set the browser driver in config.properties:**

* 1 for Chrome
* 2 for Edge

**Execution Flow (via TestNG)**

When the testng.xml file is executed, TestNG performs the following steps:

1. Suite Initialization: The test suite named in the XML is launched, initiating the test lifecycle.

2. Listener Setup: Custom listeners like CustomListener and RetryTransformer are activated for logging, reporting, and retrying failed tests.

3. Test Block Execution: The <test> block begins execution using multiple parallel threads (e.g., up to five), as defined in the XML configuration.

4. Class & Method Discovery: TestNG loads test classes (e.g., TestNGTestRunner) and identifies executable test methods and lifecycle hooks (@BeforeSuite, @AfterClass, etc.).

5. Test Execution: Setup methods run first, followed by actual test methods and/or Cucumber scenarios. Tests are executed in parallel if enabled.

6. Failure Handling: Retry logic is applied for failed tests via RetryTransformer, ensuring re-execution based on the configured retry policy.

7. Reporting: Throughout the run, logs are generated, screenshots are captured on failure, and reports are compiled. Allure integration adds visual reports for enhanced insights.

8. Completion: Once all tests and cleanup routines finish, TestNG finalizes the suite and generates its final reports.

**Prerequisites**

* Java JDK
* Eclipse IDE or compatible Java editor
* Selenium WebDriver
* TestNG
* Cucumber
* Maven
* Apache POI
* Allure Report setup

**File Structure**

| Folder | Description |
| --- | --- |
| src/test/java | Page Object Models and utility classes, test scripts, step definitions, listeners, hooks, runners |
| src/test/resources | Feature files, test datasets (Excel, JSON, XML), configuration files |
| TestNG.xml | Suite-level configuration for TestNG execution |

**Conclusion**

This automation suite demonstrates a comprehensive testing solution that validates dynamic content extraction, form submission behavior, and structured menu hierarchy interactions. By incorporating Selenium, TestNG, Cucumber, and Allure Reporting, the project showcases real-world web automation practices and test management strategies. The modular architecture and usage of industry-standard frameworks make it scalable and maintainable for future enhancements.