

# Selenium Java Automation Testing Project Documentation

## Capstone Project – JpetStore

---

### 1. Project Overview

**Project Name:** Capstone Project – JpetStore Automation

**Problem Statement:**

JPetStore is an online pet store platform that allows users to register, log in, search for pets, add them to the cart, and filter pet categories. Users can manage their cart by editing or removing selected pets before proceeding to checkout. The checkout process involves confirming pet selection, entering shipping details, and making a payment. Given the critical nature of these functionalities, any failure in the system can impact user experience and lead to potential revenue loss. Implementing an automated testing solution ensures seamless performance, functionality, and reliability for JPetStore users.

**Objective:**

The objective of this project is to implement automated testing for the JPetStore website using Selenium and TestNG. This automation will cover UI testing to ensure critical functionalities like registration, login, product search, cart management, and checkout work seamlessly. The framework will be built using Page Object Model (POM) or Page Factory, ensuring maintainability

**Tools & Technologies Used:**

- **Programming Language:** Java
- **Testing Framework:** TestNG, Junit
- **BDD tool:** Cucumber
- **Automation Tool:** Selenium WebDriver
- **Build Tool:** Maven
- **Reporting:** Extent Reports
- **IDE:** Eclipse
- **Data Management:** Excel file (for test data) and properties file (for configuration)

### 2. Project Objective

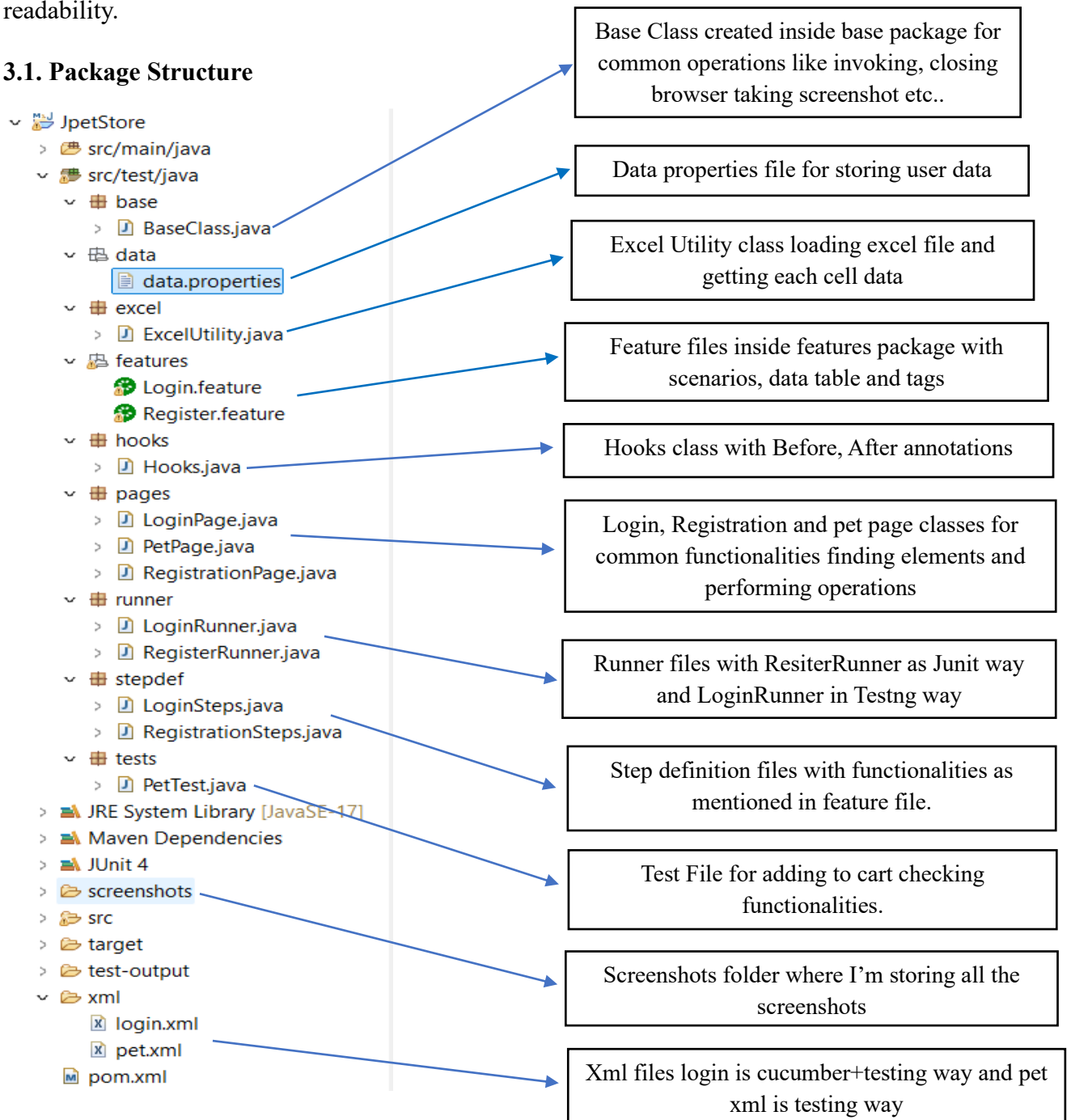
The objective of this project is to automate the testing of key functionalities of the JpetStore website. The project covers the following test cases:

- User Registration
- User Login
- Adding item to Cart
- Validating Successful Purchase

### 3. Project Structure

The project is organized into multiple packages and classes for better maintainability and readability.

#### 3.1. Package Structure



## 4. Key Components

### 4.1. Base Class

- The BaseClass.java handles the WebDriver initialization and browser configurations.

#### 1. initExtentReport()

- Initializes the **Extent Reports** framework for test reporting.
- Saves the report inside target/ExtentReports/ with a timestamped filename.
- Sets system information such as OS and default browser.

#### 2. invokeBrowser()

- Reads the browser type from data.properties and launches **Chrome, Firefox, or Edge**.
- Maximizes the browser window and sets an implicit wait of **10 seconds**.
- Loads the URL from the properties file.

#### 3. invokeBrowser(String browser, String url)

- Accepts **browser type and URL** as parameters and launches the specified browser.
- Ensures that the **Extent Report** is initialized before starting the test.
- Maximizes the window and applies implicit waits for better stability.

#### 4. loadProperties()

- Loads **application configurations** from the data.properties file.
- Uses FileInputStream to read property values such as **browser type and URL**.
- Throws an error if the file cannot be loaded.

#### 5. loadExcelData()

- Loads test data from an **Excel file** (data.xlsx).
- Uses ExcelUtility.loadExcelData() to initialize Excel reading.
- Sets the default Excel sheet to **"sign"** for accessing test data.

#### 6. screenshot()

- Captures a **screenshot** of the current webpage.
- Uses TakesScreenshot to store the screenshot in the ./screenshots/ folder.
- Returns the file path of the captured image.

#### 7. closeBrowser()

- Closes the browser instance if it is open.

- Calls driver.quit() to terminate the WebDriver session.

## 8. flushExtentReport()

- Finalizes and **saves the Extent Report** after test execution.
- Ensures all logs and test results are properly written to the report.

## 5. Utility Classes – Excel Utility.java

### 1. loadExcelData(String filePath)

- Loads an Excel file from the specified file path into memory.
- Uses FileInputStream to read the file and initializes an XSSFWorkbook.
- Prints an error if the file cannot be loaded.

### 2. setSheet(String sheetName)

- Sets the active sheet based on the provided sheet name.
- Throws an error if the sheet does not exist in the workbook.

### 3. getCellData(int rowNum, int colNum)

- Retrieves cell data as a String from the specified row and column.
- Uses DataFormatter to format different types of cell values into String format.
- Returns "Cell not found" if the cell is empty or missing.

## 6. Features

### 6.1. Login. feature

#### 1. @invalid1 - Login with Invalid Data Table Details

- Navigates to the login page.
- Attempts login using invalid data table credentials (wrong username/password).
- Verifies that an error message is displayed.

#### 2. @valid1 - Login with Valid Data Table Details

- Navigates to the login page.
- Logs in using valid data table credentials.
- Verifies that the user is successfully logged in.

#### 3. @invalid2 - Login with Invalid Excel Details

- Navigates to the login page.
- Attempts login using invalid Excel data.
- Verifies that an error message is displayed.

#### 4. @valid2 - Login with Valid Properties Details

- Navigates to the login page.
- Logs in using valid properties file credentials.
- Verifies that the user is successfully logged in.

## 6.2. Register. Feature

#### 1. @invalid1 - Register with Invalid Data Table Details

- Navigates to the registration page.
- Attempts registration using invalid data table details (incorrect/missing fields).
- Verifies that an error message appears for missing details.

#### 2. @valid1 - Register with Valid Data Table Details

- Navigates to the registration page.
- Registers using valid data table details.
- Verifies that the user is successfully registered.

#### 3. @invalid2 - Register with Invalid Excel Details

- Navigates to the registration page.
- Attempts registration using invalid Excel data.
- Verifies that an error message appears for missing details.

#### 4. @valid2 - Register with Valid Properties Details

- Navigates to the registration page.
- Registers using valid properties file details.
- Verifies that the user is successfully registered.

## 7. hooks

**Class:** Hooks.java

**Description:** The Hooks class initializes and cleans up test execution using **Cucumber**

**Hooks (@Before and @After).** It extends BaseClass to reuse common functionalities.

Methods:

1. setUp() (@Before Hook)
  - Initializes the Extent Report (initExtentReport()).
  - Launches the browser (invokeBrowser()).
2. tearDown() (@After Hook)
  - Closes the browser (closeBrowser()).
  - Flushes the Extent Report (flushExtentReport()).

## 8. Pages

**8.1 Class:** LoginPage.java

**Description:** This class interacts with the **Login Page** using the **Page Object Model (POM)** and includes:

- **Explicit waits** for stable interactions.
- **Exception handling** for missing elements.
- **Logging & Reporting** using test.pass() and test.fail().
- **Screenshots** for test evidence.

**Methods:**

1. **openLoginPage()**
  - Clicks on "**Sign In**" link to navigate to the Login page.
  - Logs success or failure.
2. **login(String user, String pass)**
  - Enters **username** and **password**.
  - Clears password field before entering data.
  - Logs success or failure.

### 3. **submit()**

- Clicks the **Login** button.
- Captures a **screenshot** of the login attempt.

### 4. **isErrorMessageDisplayed()**

- Checks if an error message is displayed after login failure.

### 5. **isLoginSuccessful()**

- Verifies successful login by checking **current URL**.

### 6. **captureScreenshot(String testName)**

- Captures and attaches a screenshot to the report.

#### **Code Snippet:**

```
LoginPage lp = new LoginPage(driver);  
  
lp.clickLoginPage();  
  
lp.enterUsername(username);  
  
lp.enterPassword(password);  
  
lp.clickLoginBtn();  
  
ExtentReport.createTest("Login Test").log(Status.PASS, "Login Successful");
```

#### **8.2 Class: RegisterPage. java**

**Description:** This class interacts with the **Registration Page** using the **Page Object Model (POM)** and includes:

- **Explicit waits** for stable interactions.
- **Exception handling** for missing elements.
- **Logging & Reporting** using test.pass() and test.fail().
- **Screenshots** for test evidence.

## **Methods:**

### **1. openRegistrationPage()**

- Navigates to the Registration Page by clicking Sign In → Register Now!
- Logs success or failure.

### **2. fillRegistrationForm(...)**

- Fills in the registration form fields.
- Selects optional checkboxes (enableMyList, enableMyBanner).
- **Logs success or failure.**

### **3. submitForm()**

- Clicks the Submit button.
- Captures a screenshot of the registration attempt.

### **4. isErrorDisplayed()**

- Checks if an error message is displayed after registration failure.

### **5. isRegistrationSuccessful()**

- Verifies successful registration by checking current URL.

### **6. captureScreenshot(String testName)**

- Captures and attaches a screenshot to the report.

## **Code Snippet:**

```
RegisterPage rp = new RegisterPage(driver);  
rp.clickRegisterPage();  
rp.enterFirstName(firstName);  
rp.enterLastName(lastName);  
rp.enterEmail(email);
```



```
rp.enterPassword(password);  
rp.confirmPassword(password);  
rp.clickRegisterBtn();  
ExtentReport.createTest("Register Test").log(Status.PASS, "Registration Successful");
```

### 8.3 Class: petPage. java

**Description:** This class interacts with the **Product Search & Add to Cart** functionalities in the **Pet Store** using **POM**.

It includes:

- **Explicit waits** for stable UI interactions.
- **Exception handling** for missing elements.
- **Logging & Screenshot Capture** using BaseClass.screenshot().

#### Methods:

##### 1. searchPet(String petName)

- Enters the pet name in the search box.
- Clicks the search button.

##### 2. selectPet()

- Selects the first pet from the search results.

##### 3. addToCart()

- Clicks the "Add to Cart" button.

##### 4. isPetAddedToCart()

- Verifies that the pet is successfully added to the cart by checking if the cart item is displayed.
- Captures a screenshot using BaseClass.screenshot().

### Code Snippet:

```
PetPage pp = new PetPage(driver);  
pp.searchPet(petName);  
pp.selectPet();  
pp.addToCart();  
  
ExtentReport.createTest("Pet Search & Add to Cart Test").log(Status.PASS, "Pet added to  
cart successfully");
```

## 9. Runner

### 9.1 Class: LoginRunner.java

**Description:** This class is a **Cucumber TestNG Runner** that executes the **Login.feature** scenarios using **Cucumber + TestNG**.

#### Configuration Details (@CucumberOptions)

1. **features = "src/test/java/features/Login.feature"**
  - Specifies the feature file location.
2. **glue = {"stepdef", "hooks"}**
  - stepdef → Contains Step Definitions for Login.
  - hooks → Handles test setup & teardown.
3. **tags = "@valid1 or @invalid1 or @valid2 or @invalid2"**
  - Runs scenarios tagged as @valid1, @invalid1, @valid2, or @invalid2.
4. **plugin = {"pretty", "html:target/cucumber-reports.html",  
"json:target/cucumber.json"}**
  - Generates reports in:
    - Pretty Console Output
    - HTML Format (target/cucumber-reports.html)
    - JSON Format (target/cucumber.json)

## 9.2 Class: RegisterRunner.java

**Description:** This class is a **Cucumber JUnit Runner** that executes the **Register.feature** scenarios using **Cucumber + JUnit.Configuration Details (@CucumberOptions)**

### Configuration Details (@CucumberOptions)

1. **features = "src/test/java/features/Register.feature"**
  - Specifies the feature file location for user registration tests.
2. **glue = {"stepdef", "hooks"}**
  - stepdef → Contains Step Definitions for Registration.
  - hooks → Handles test setup & teardown.
3. **tags = "@valid1 or @invalid1 or @valid2 or @invalid2"**
  - Runs scenarios tagged as @valid1, @invalid1, @valid2, or @invalid2.
4. **plugin = {"pretty", "html:target/cucumber-reports.html", "json:target/cucumber.json"}**
  - Generates reports in:
    - Pretty Console Output
    - HTML Format (target/cucumber-reports.html)
    - JSON Format (target/cucumber.json)

## 10. Step Definitions

### 10.1 Class: LoginSteps.java

**Description:** This class defines the **Cucumber step definitions** for **login scenarios**, handling **valid and invalid credentials** from different sources (**DataTable, Excel, Properties file**).

### Methods & Functionality

1. **@Given("User navigates to the login page")**
  - Initializes Extent Report test case: "User Login Test".
  - Opens the application URL from valid.properties.

- Calls `loginPage.openLoginPage()` to navigate to the login page.
  - Marks the step as passed if successful.
2. **@When("User enters invalid data table login details")**
    - Retrieves invalid credentials from Cucumber DataTable (`dataTable.asMaps()`).
    - Iterates through each row of credentials and performs login.
    - Submits the form and marks the step as passed.
  3. **@When("User enters valid data table login details")**
    - Same as above, but for valid credentials.
  4. **@When("User enters invalid excel login details")**
    - Fetches credentials from an Excel file using `getCellData(row, column)`.
    - Calls `loginPage.login(username, password)`.
    - Submits the form and marks the step as passed.
  5. **@When("User enters valid properties login details")**
    - Retrieves login credentials from `valid.properties`.
    - Performs login and submits the form.
    - Marks the step as passed.
  6. **@Then("Error message should be displayed")**
    - Verifies whether the error message is displayed using `Assert.assertTrue(loginPage.isErrorMessageDisplayed())`.
    - On failure: Captures a screenshot and attaches it to Extent Report.
  7. **@Then("User should be logged in successfully")**
    - Verifies successful login using `Assert.assertTrue(loginPage.isLoginSuccessful())`.
    - On failure: Captures a screenshot and attaches it to Extent Report.

## 10.2 Class: RegisterSteps. java

**Description:** Defines step definitions for the **User Registration** feature in the **BDD Cucumber + Maven + TestNG + Java + Selenium WebDriver** automation framework. It handles both **valid and invalid** registration scenarios using **DataTable, Excel, and Properties file** for test data.

## Methods & Functionality:

### 1. **userNavigatesToRegistrationPage()**

- Initializes Extent Report logging for registration test.
- Navigates to the registration page using the URL from the properties file.
- Calls `openRegistrationPage()` from `RegistrationPage.java`.

### 2. **userEntersInvalidDetails(DataTable dataTable)**

- Reads invalid registration data from a Cucumber DataTable (List of Maps).
- Calls `fillRegistrationForm()` to enter details into the registration form.
- Submits the form and logs the event.

### 3. **userEntersValidDetails(DataTable dataTable)**

- Reads valid registration data from a Cucumber DataTable.
- Calls `fillRegistrationForm()` to enter details into the registration form.
- Submits the form and logs the event.

### 4. **userEntersInvalidDetails() (Excel-based Data-Driven Testing)**

- Fetches invalid user details from Excel using `getCellData(row, column)`.
- Calls `fillRegistrationForm()` with the fetched data.
- Submits the form and logs the event.

### 5. **userEntersValidDetails() (Properties-based Data-Driven Testing)**

- Fetches valid user details from a properties file (`valid.properties`).
- Calls `fillRegistrationForm()` with the fetched data.
- Submits the form and logs the event.

### 6. **verifyErrorMessage1() (Validation for Invalid Registration)**

- Uses `Assert.assertTrue()` to validate that the error message appears.
- Captures a screenshot and logs a failure if the error message is missing.

### 7. **verifySuccessfulRegistration() (Validation for Successful Registration)**

- Uses `Assert.assertTrue()` to check if the registration was successful.
- Captures a screenshot and logs a failure if registration fails.

## 11. Tests

### 11.1 Class: PetTest.java

**Description:** This class contains TestNG test cases for searching a pet and adding it to the cart in the online pet store. It retrieves test data from both an **Excel sheet** and a **properties file**, ensuring data-driven testing.

#### Methods & Functionality:

##### 1. setUp(String browser, String url)

- BeforeMethod annotation: Initializes the WebDriver before each test.
- Invokes browser and navigates to the provided URL.
- Instantiates the PetPage object.

##### 2. getPetDataFromExcel() (*Data Provider*)

- Fetches pet search data from an Excel file using Apache POI.
- Returns the pet name from the 18th column (index 17) of the Excel sheet.

##### 3. getPetDataFromProperties() (*Data Provider*)

- Retrieves pet search data from a properties file (searchProduct property).

##### 4. testSearchAndAddToCart\_ExcelData(String petName) (*Test Case - Disabled*)

- Uses Excel data for searching a pet.
- Calls searchPet(), selectPet(), and addToCart().
- Asserts if the pet is successfully added to the cart.
- Handles NoSuchElementException and captures a screenshot on failure.
- Extent Reporting is used for logging.

##### 5. testSearchAndAddToCart\_PropertiesFile(String petName) (*Test Case - Enabled*)

- Uses properties file data for searching a pet.
- Performs the same steps as testSearchAndAddToCart\_ExcelData().
- Handles exceptions and logs results in Extent Reports.

##### 6. close() (*AfterMethod annotation*)

- Closes the browser after each test execution.
- Flushes the Extent Reports to store logs.

#### Code Snippet:

```

PetPage pp = new PetPage(driver);

pp.searchPet(petName);

pp.selectPet();

pp.addToCart();

Assert.assertTrue(pp.isPetAddedToCart(), "Pet was not added to the cart!");

ExtentReport.createTest("Pet Store Test").log(Status.PASS, "Pet successfully added to cart");
("Pet Search & Add to Cart Test").log(Status.PASS, "Pet added to cart successfully");

```

## 12. Tests

### 12.1 File: login. xml

**Description:** This TestNG XML file defines a parallel test execution suite for running Cucumber-based tests.

#### Configuration breakdown:

- **Parallel Execution:** Runs tests concurrently (parallel="tests", thread-count="2").
- **Suite Name:** "Automation Suite" (handles multiple tests).
- **Test Defined:** "Cucumber Tests" (executes runner.LoginRunner).
- **Enhancement:** Add more test classes (runner.RegisterRunner) for parallel execution.

### 12.2 File: pet. xml

**Description:** Runs **PetTest** in **Edge & Firefox parallelly** using TestNG, with **thread-count=2** for faster execution.

#### Configuration Breakdown:

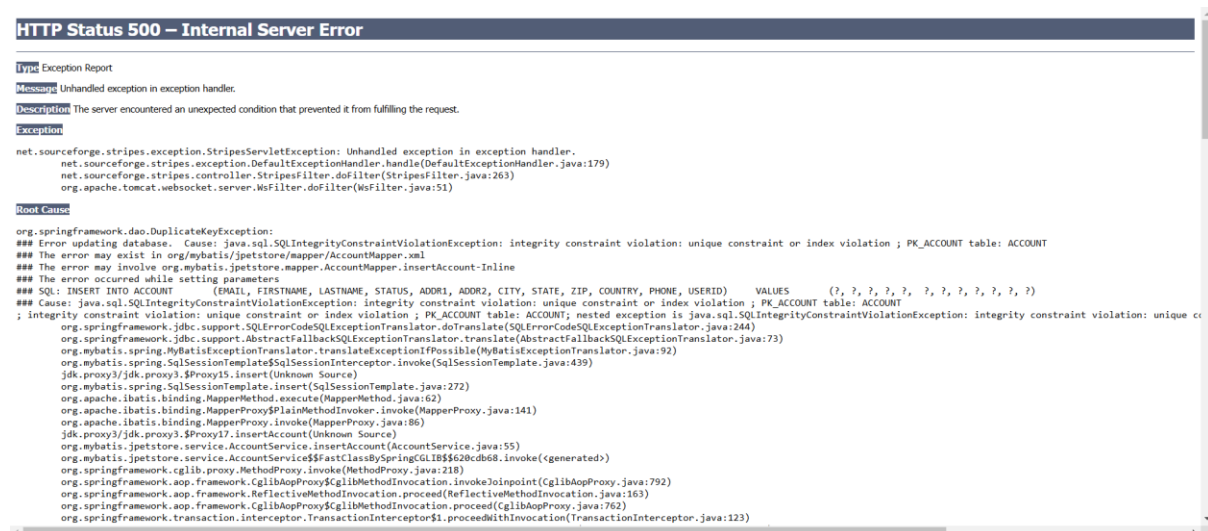
- **Suite Name:** "PetStoreSuite" (manages multiple browser tests).
- **Parallel Execution:** Tests run simultaneously (parallel="tests").
- **Edge\_Test:** Runs PetTest on Edge (browser="edge").
- **Firefox\_Test:** Runs PetTest on Firefox (browser="firefox").
- **URL Parameter:** "https://petstore.octoperf.com/actions/Catalog.action" used for both tests.

## 13. Reports & ScreenShots

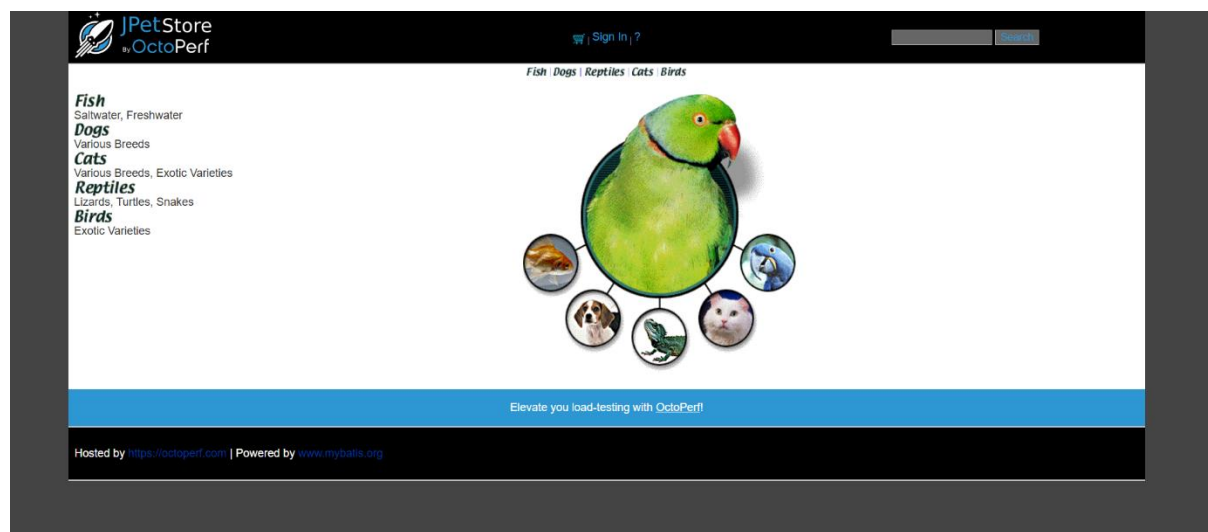
### 13.1. Screenshots

#### 13.1.1. Registration Page:

**Register Page Screenshot for invalid data:** Captures successful and failed Register attempt, displaying error messages for invalid credentials.




**Register Page Screenshot for valid data:** Captures successful and successfully Registered, redirected to home page without any warnings.





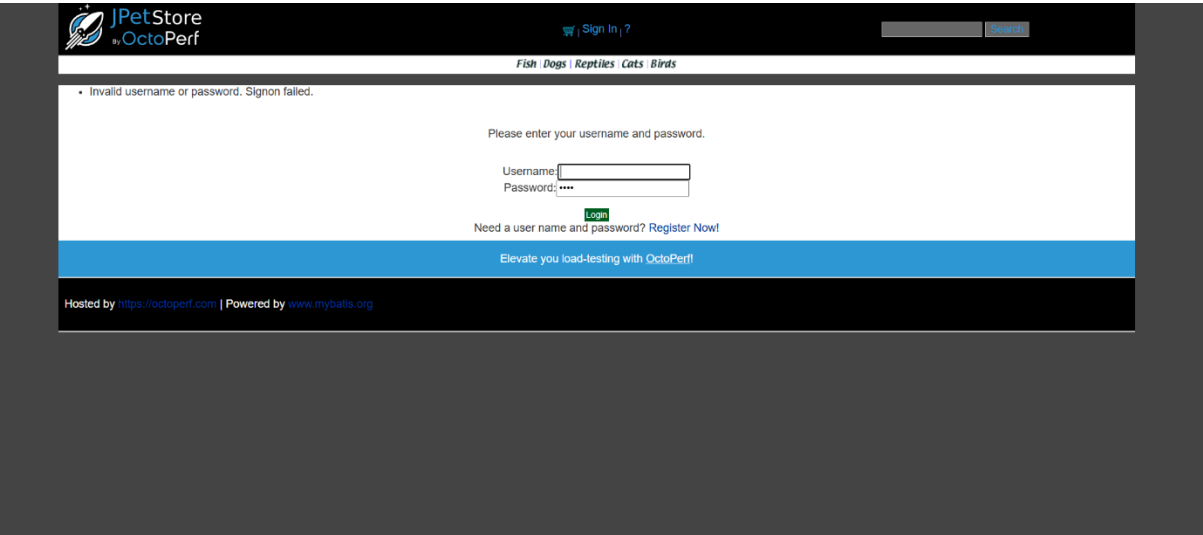
## Extent Report:

Tests		User Registration Test	
User Registration Test	12:57:48 am / 00:00:03.767	Pass	
User Registration Test	12:57:56 am / 00:00:03.331	Pass	
User Registration Test	12:58:01 am / 00:00:05.845	Pass	
User Registration Test	12:58:10 am / 00:00:03.884	Pass	

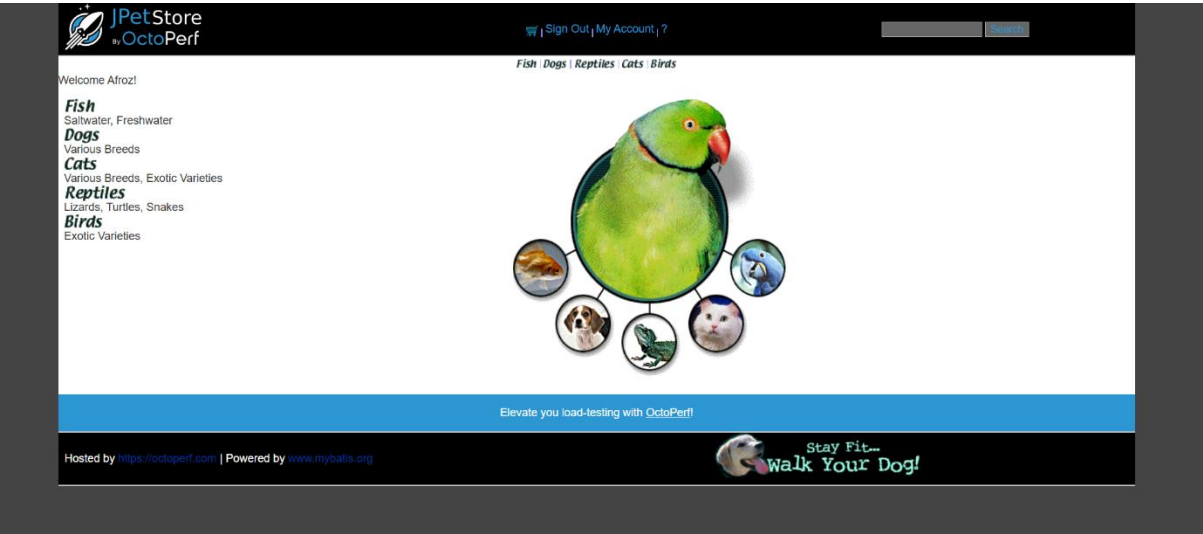
User Registration Test		
09.23.2025 12:57:48 am 09.23.2025 12:57:52 am 00:00:03.767 #test-id=1		
		
STATUS	TIMESTAMP	DETAILS
Info	12:57:48 am	Navigating to Registration Page
Pass	12:57:50 am	Navigated to Registration Page
Pass	12:57:50 am	User successfully navigated to registration page
Info	12:57:50 am	Entering invalid registration details using DataTable
Pass	12:57:51 am	Filled Registration Form successfully
Pass	12:57:52 am	Submitted Registration Form
Pass	12:57:52 am	Invalid registration form submitted
Pass	12:57:52 am	Error message displayed
Pass	12:57:52 am	Error message displayed successfully

### 13.1.2. Login Page:


**Login Page Screenshot for invalid data:** Captures successful and failed login attempt, displaying error messages for invalid credentials.



**Login Page Screenshot for valid data:** Captures successful and successfully logged in, entered to the home page.

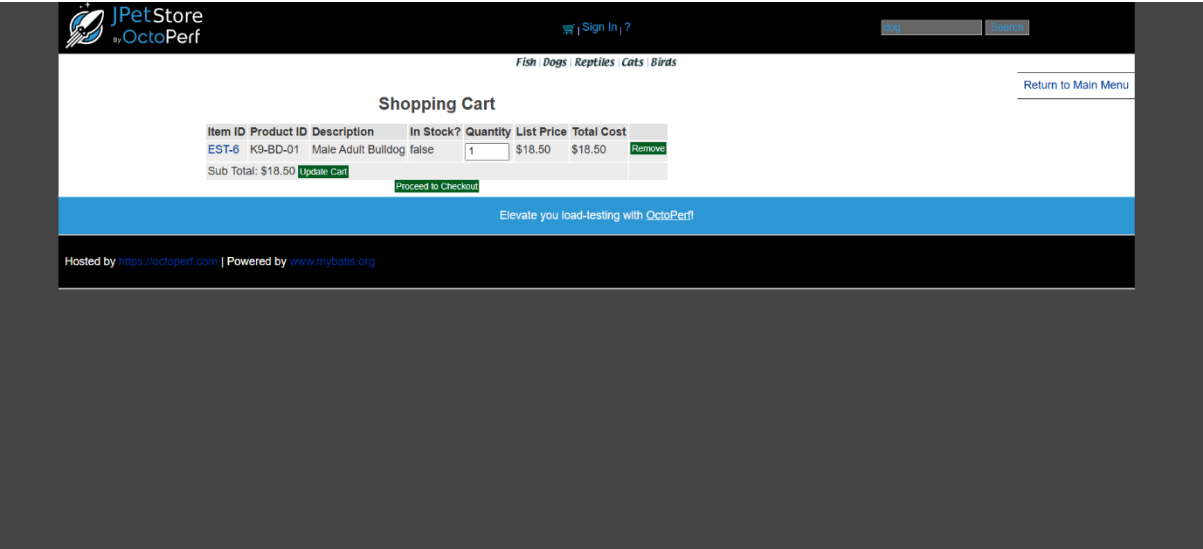


**Extent Report:**

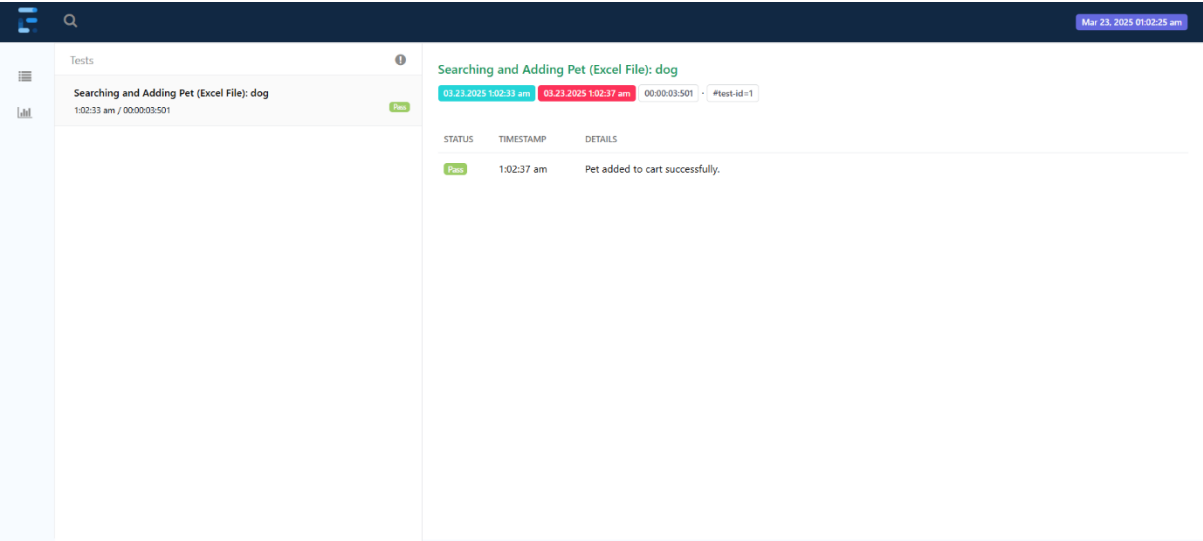
Tests			User Login Test		
User Login Test			03.23.2025 1:13:44 am	03.23.2025 1:13:46 am	00:00:02:649 #test-id=1
1:13:44 am / 00:00:02:649					
User Login Test					
1:13:51 am / 00:00:02:074					
User Login Test					
1:13:56 am / 00:00:03:968					
User Login Test					
1:14:02 am / 00:00:02:220					
			STATUS	TIMESTAMP	DETAILS
			Info	1:13:44 am	Navigating to Login Page
			Pass	1:13:45 am	Opened Login Page
			Pass	1:13:45 am	User successfully navigated to login page
			Info	1:13:45 am	Entering invalid login details using DataTable
			Pass	1:13:45 am	Entered credentials: afroz
			Pass	1:13:45 am	Entered credentials: afroz
			Pass	1:13:46 am	Clicked on Login button
			Pass	1:13:46 am	Invalid login form submitted
			Pass	1:13:46 am	Error message displayed
			Pass	1:13:46 am	Error message displayed successfully

13.1.3. Cart Page:

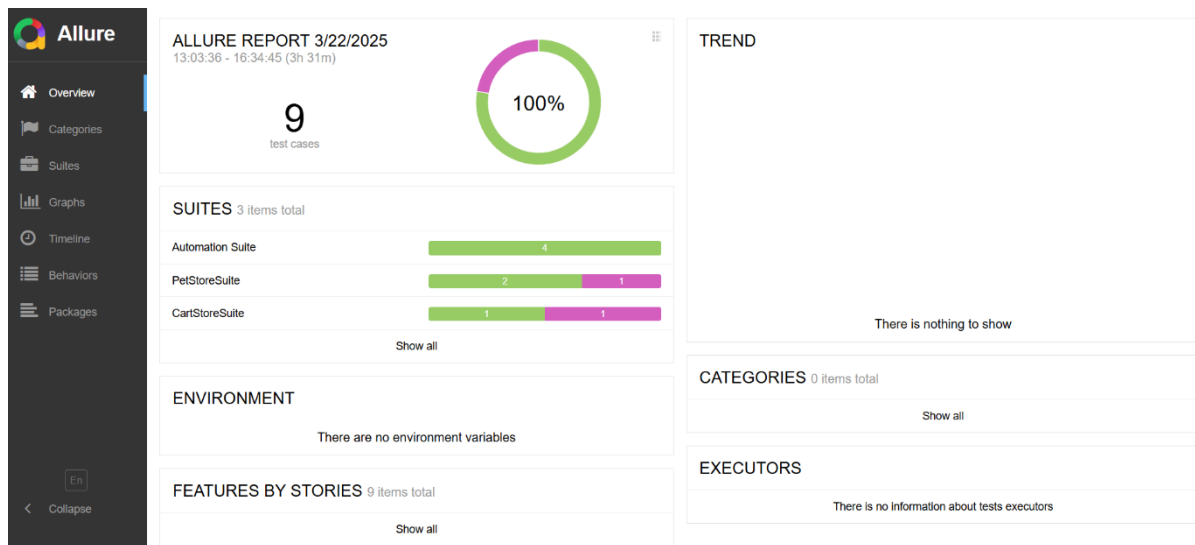
**Cart validation:** After adding any item to cart the cart is updated like this.



Extent Report:



## Allure Report:



## 14. Challenges & Solutions

### 1. Handling Dynamic Elements & Synchronization Issues

- **Challenge:** Some elements (e.g., search results, cart updates) load dynamically, causing `NoSuchElementException` due to delays in visibility.
- **Solution:** Implemented **WebDriverWait (Explicit Wait)** in Page Object Model (POM) to ensure elements are interactable before performing actions.

### 2. Ensuring Consistent Parallel Execution Across Browsers

- **Challenge:** Running tests in **Edge** and **Firefox** in parallel caused test failures due to stale elements and timing issues.
- **Solution:** Adjusted **TestNG** thread count in **XML**, optimized wait times, and ensured each test starts with a fresh browser session to prevent conflicts.

## 15. Conclusion

This Selenium-based Java automation project successfully performs end-to-end testing of the Pet Store website using JUnit, TestNG, and Cucumber. It efficiently integrates BDD Cucumber for scenario-based testing, TestNG for structured execution and reporting, and Selenium WebDriver for UI automation. The project follows a modular Page Object Model (POM), ensuring easy maintenance and scalability. Additionally, parallel execution, data-driven testing, exception handling, and Extent Reports enhance test reliability, debugging efficiency, and detailed validation of test cases.