Table of Contents

1. Framework Overview

Key Features

- ✓ **Page Object Model:** Clean separation of test logic and page interactions
- ✓ **TestNG Annotations:** @Test, @BeforeMethod, @AfterMethod, @DataProvider
- ✓ **Test Grouping:** Smoke, Regression, Negative test categories
- ✓ **Parallel Execution:** Multi-threaded test execution
- ✓ **Allure Reports:** Rich HTML reports with screenshots
- ✓ **Cross-Browser:** Chrome, Firefox, Edge support
- ✓ **Data-Driven:** External test data support
- ✓ **CI/CD Ready:** Jenkins pipeline integration

2. Project Architecture

```
selenium-testng-framework/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── pages/                    # Page Object classes
│   │   │   │   ├── BasePage.java         # Base page with common methods
│   │   │   │   ├── LoginPage.java        # Login page interactions
│   │   │   │   ├── ProductsPage.java     # Products page interactions
│   │   │   │   ├── CartPage.java         # Cart page interactions
│   │   │   │   └── CheckoutPage.java     # Checkout page interactions
│   │   │   ├── factory/
│   │   │   │   └── DriverFactory.java    # WebDriver initialization
│   │   │   └── utils/
│   │   │       └── ConfigReader.java     # Configuration file reader
│   │   └── resources/
│   │       ├── config.properties         # Test configuration
│   │       └── log4j2.xml                # Logging configuration
│   └── test/
│       └── java/
│           ├── tests/                    # Test classes
│           │   ├── BaseTest.java         # Base test setup/teardown
│           │   ├── LoginTest.java        # Login test cases
│           │   ├── ProductsTest.java     # Products test cases
│           │   ├── CartTest.java         # Cart test cases
│           │   └── CheckoutTest.java     # Checkout test cases
│           └── listeners/
│               └── AllureListener.java   # Allure report listener
├── testng.xml                            # TestNG suite configuration
├── pom.xml                               # Maven dependencies
├── Jenkinsfile                           # CI/CD pipeline
└── README.md                             # Project documentation
```

3. Page Objects

BasePage.java

src/main/java/pages/BasePage.java

```
public class BasePage {
    protected WebDriver driver;
    protected WebDriverWait wait;
    protected Logger logger = LogManager.getLogger(this.getClass());

    public BasePage(WebDriver driver) {
        this.driver = driver;
        this.wait = new WebDriverWait(driver, Duration.ofSeconds(10));
        PageFactory.initElements(driver, this);
    }

    protected void click(WebElement element) {
        wait.until(ExpectedConditions.elementToBeClickable(element));
        element.click();
    }

    protected void type(WebElement element, String text) {
        wait.until(ExpectedConditions.visibilityOf(element));
        element.clear();
        element.sendKeys(text);
    }

    protected String getText(WebElement element) {
        wait.until(ExpectedConditions.visibilityOf(element));
        return element.getText();
    }

    protected boolean isDisplayed(WebElement element) {
        try {
            return element.isDisplayed();
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}
```

LoginPage.java

src/main/java/pages/LoginPage.java

```
public class LoginPage extends BasePage {

    @FindBy(id = "user-name")
    private WebElement usernameInput;

    @FindBy(id = "password")
```

```
    private WebElement passwordInput;

    @FindBy(id = "login-button")
    private WebElement loginButton;

    @FindBy(css = "[data-test='error']")
    private WebElement errorMessage;

    public LoginPage(WebDriver driver) {
        super(driver);
    }

    public void enterUsername(String username) {
        type(usernameInput, username);
        logger.info("Entered username: " + username);
    }

    public void enterPassword(String password) {
        type(passwordInput, password);
        logger.info("Entered password");
    }

    public ProductsPage clickLoginButton() {
        click(loginButton);
        logger.info("Clicked login button");
        return new ProductsPage(driver);
    }

    public ProductsPage login(String username, String password) {
        enterUsername(username);
        enterPassword(password);
        return clickLoginButton();
    }

    public String getErrorMessage() {
        return getText(errorMessage);
    }
}
```

ProductsPage.java

src/main/java/pages/ProductsPage.java

```
public class ProductsPage extends BasePage {

    @FindBy(css = ".title")
    private WebElement pageTitle;

    @FindBy(css = ".inventory_item")
```

```
    private List<WebElement> productItems;

    @FindBy(css = ".shopping_cart_badge")
    private WebElement cartBadge;

    @FindBy(css = ".shopping_cart_link")
    private WebElement cartIcon;

    @FindBy(css = ".product_sort_container")
    private WebElement sortDropdown;

    public ProductsPage(WebDriver driver) {
        super(driver);
    }

    public String getPageTitle() {
        return getText(pageTitle);
    }

    public boolean isPageLoaded() {
        return isDisplayed(pageTitle) &&
               getPageTitle().equals("Products");
    }

    public int getProductCount() {
        return productItems.size();
    }

    public void addProductToCart(String productName) {
        String buttonId = "add-to-cart-" +
            productName.toLowerCase().replace(" ", "-");
        WebElement addButton = driver.findElement(By.id(buttonId));
        click(addButton);
        logger.info("Added to cart: " + productName);
    }

    public int getCartBadgeCount() {
        if (isCartBadgeVisible()) {
            return Integer.parseInt(cartBadge.getText());
        }
        return 0;
    }

    public boolean isCartBadgeVisible() {
        return isDisplayed(cartBadge);
    }

    public CartPage clickCartIcon() {
        click(cartIcon);
        return new CartPage(driver);
    }
```

```
    public void sortProducts(String option) {
        Select select = new Select(sortDropdown);
        select.selectByVisibleText(option);
        logger.info("Sorted by: " + option);
    }
}
```

4. Login Tests

6 test cases in this category

Test #1 Valid User Login

Critical Smoke Regression

Description

Verify that a valid user can successfully log in to the SauceDemo application using correct credentials and is redirected to the Products page.

TestNG Implementation

src/test/java/tests/LoginTest.java

```
@Epic("Authentication")
@Feature("Login")
@Story("Valid Login")
@Severity(SeverityLevel.CRITICAL)
@Test(groups = {"smoke", "regression"})
@Description("Verify valid user can login successfully")
public void testValidLogin() {
    logger.info("Starting valid login test");

    loginPage.enterUsername(config.getProperty("valid.username"));
    loginPage.enterPassword(config.getProperty("valid.password"));
    loginPage.clickLoginButton();

    Assert.assertEquals(productsPage.getPageTitle(), "Products",
        "User should be redirected to Products page");
    logger.info("Valid login test completed successfully");
}
```

Execution Flow

1. Browser launches via DriverFactory
2. Navigate to base URL from config.properties
3. LoginPage.enterUsername() - locates and enters username
4. LoginPage.enterPassword() - locates and enters password
5. LoginPage.clickLoginButton() - clicks login button
6. ProductsPage.getPageTitle() - retrieves page title
7. Assert equals 'Products'
8. Screenshot captured on success (AllureListener)

Expected Result

✓ User is logged in and redirected to Products page showing 'Products' as the page title.

Test #2 Invalid Credentials Error

Critical | Smoke | Regression | Negative

Description

Verify that attempting to login with invalid credentials displays an appropriate error message.

TestNG Implementation

src/test/java/tests/LoginTest.java

```
@Epic("Authentication")
@Feature("Login")
@Story("Invalid Login")
@Severity(SeverityLevel.CRITICAL)
@Test(groups = {"smoke", "regression", "negative"})
@Description("Verify error message for invalid credentials")
public void testInvalidCredentials() {
    logger.info("Starting invalid credentials test");

    loginPage.enterUsername("invalid_user");
    loginPage.enterPassword("wrong_password");
    loginPage.clickLoginButton();
```

```
    String errorMessage = loginPage.getErrorMessage();
    Assert.assertTrue(errorMessage.contains("Username and password do
not match"),
        "Error message should indicate invalid credentials");
    logger.info("Invalid credentials test completed");
}
```

Execution Flow

1. Browser launches via DriverFactory
2. Navigate to login page
3. Enter invalid username
4. Enter invalid password
5. Click login button
6. LoginPage.getErrorMessage() - retrieves error text
7. Assert error contains expected message
8. Screenshot captured

Expected Result

✓ Error message is displayed indicating 'Username and password do not match any user in this service'.

Test #3 Locked Out User Error

High Regression Negative

Description

Verify that a locked out user cannot login and receives an appropriate error message.

TestNG Implementation

src/test/java/tests/LoginTest.java

```
@Epic("Authentication")
@Feature("Login")
```

```
@Story("Locked User")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression", "negative"})
@Description("Verify locked out user cannot login")
public void testLockedOutUser() {
    logger.info("Starting locked out user test");

    loginPage.enterUsername("locked_out_user");
    loginPage.enterPassword("secret_sauce");
    loginPage.clickLoginButton();

    String errorMessage = loginPage.getErrorMessage();
    Assert.assertTrue(errorMessage.contains("locked out"),
        "Error message should indicate user is locked out");
}
```

Execution Flow

1. Navigate to login page
2. Enter 'locked_out_user' username
3. Enter valid password
4. Click login button
5. Verify error message displays
6. Assert message contains 'locked out'

Expected Result

✓ Error message displays 'Sorry, this user has been locked out'.

Test #4 Empty Username Validation

Medium | Regression | Negative

Description

Verify that submitting the login form with an empty username displays a validation error.

TestNG Implementation

src/test/java/tests/LoginTest.java

```
@Epic("Authentication")
@Feature("Login")
@Story("Field Validation")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression", "negative"})
@Description("Verify empty username validation")
public void testEmptyUsername() {
    logger.info("Starting empty username validation test");

    loginPage.enterPassword("secret_sauce");
    loginPage.clickLoginButton();

    String errorMessage = loginPage.getErrorMessage();
    Assert.assertTrue(errorMessage.contains("Username is required"),
        "Error should indicate username is required");
}
```

Execution Flow

1. Navigate to login page
2. Leave username field empty
3. Enter password
4. Click login button
5. Verify validation error appears
6. Assert message contains 'Username is required'

Expected Result

✓ Validation error message displays 'Username is required'.

Test #5 Empty Password Validation

Medium | Regression | Negative

Description

Verify that submitting the login form with an empty password displays a validation error.

TestNG Implementation

src/test/java/tests/LoginTest.java

```
@Epic("Authentication")
@Feature("Login")
@Story("Field Validation")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression", "negative"})
@Description("Verify empty password validation")
public void testEmptyPassword() {
    logger.info("Starting empty password validation test");

    loginPage.enterUsername("standard_user");
    loginPage.clickLoginButton();

    String errorMessage = loginPage.getErrorMessage();
    Assert.assertTrue(errorMessage.contains("Password is required"),
        "Error should indicate password is required");
}
```

Execution Flow

1. Navigate to login page
2. Enter username
3. Leave password field empty
4. Click login button
5. Verify validation error appears
6. Assert message contains 'Password is required'

Expected Result

✓ Validation error message displays 'Password is required'.

Test #6 Performance Glitch User Login

Low Regression

Description

Verify that the performance_glitch_user can login successfully despite delayed response times.

TestNG Implementation

src/test/java/tests/LoginTest.java

```
@Epic("Authentication")
@Feature("Login")
@Story("Performance User")
@Severity(SeverityLevel.MINOR)
@Test(groups = {"regression"})
@Description("Verify performance glitch user can login")
public void testPerformanceGlitchUser() {
    logger.info("Starting performance glitch user test");

    loginPage.enterUsername("performance_glitch_user");
    loginPage.enterPassword("secret_sauce");
    loginPage.clickLoginButton();

    // Wait for slow response
    WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));
    wait.until(ExpectedConditions.titleContains("Products"));

    Assert.assertEquals(productsPage.getPageTitle(), "Products");
}
```

Execution Flow

1. Navigate to login page
2. Enter performance_glitch_user credentials
3. Click login button
4. Wait up to 10 seconds for response
5. Verify products page loads
6. Assert page title

Expected Result

✓ User successfully logs in despite delayed response, products page displays.

5. Products Tests

7 test cases in this category

Test #7 Products Page Displayed After Login

Critical Smoke Regression

Description

Verify that the products page is displayed after successful login with all expected elements visible.

TestNG Implementation

src/test/java/tests/ProductsTest.java

```
@Epic("Products")
@Feature("Product Display")
@Story("Products Page Load")
@Severity(SeverityLevel.CRITICAL)
@Test(groups = {"smoke", "regression"})
@Description("Verify products page displays after login")
public void testProductsPageDisplayed() {
    logger.info("Verifying products page display");

    // Login first
    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    Assert.assertTrue(productsPage.isPageLoaded(),
        "Products page should be loaded");
    Assert.assertTrue(productsPage.isProductListVisible(),
        "Product list should be visible");
    Assert.assertTrue(productsPage.isCartIconVisible(),
        "Cart icon should be visible");
}
```

Execution Flow

1. Login with valid credentials
2. Wait for products page to load
3. Verify page title is 'Products'

4. Verify product list container visible
5. Verify cart icon visible
6. Verify header elements present

Expected Result

✓ Products page displays with product list, cart icon, and all header elements visible.

Test #8 Verify 6 Products Listed

High Smoke Regression

Description

Verify that exactly 6 products are displayed on the products page.

TestNG Implementation

src/test/java/tests/ProductsTest.java

```
@Epic("Products")
@Feature("Product Display")
@Story("Product Count")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"smoke", "regression"})
@Description("Verify 6 products are listed")
public void testSixProductsListed() {
    logger.info("Verifying product count");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    int productCount = productsPage.getProductCount();
    Assert.assertEquals(productCount, 6,
        "There should be exactly 6 products");
}
```

Execution Flow

1. Login with valid credentials

2. Navigate to products page
3. Count all product items
4. Assert count equals 6

Expected Result

✓ Exactly 6 products are displayed on the products page.

Test #9 Add Single Product to Cart

Critical | Smoke | Regression

Description

Verify that a single product can be added to the cart and the cart badge updates.

TestNG Implementation

src/test/java/tests/ProductsTest.java

```
@Epic("Products")
@Feature("Add to Cart")
@Story("Single Product")
@Severity(SeverityLevel.CRITICAL)
@Test(groups = {"smoke", "regression"})
@Description("Verify adding single product to cart")
public void testAddSingleProductToCart() {
    logger.info("Adding single product to cart");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");

    Assert.assertEquals(productsPage.getCartBadgeCount(), 1,
        "Cart badge should show 1");
    Assert.assertTrue(productsPage.isRemoveButtonVisible("Sauce Labs
Backpack"),
        "Remove button should be visible");
}
```

Execution Flow

1. Login with valid credentials
2. Navigate to products page
3. Click 'Add to cart' for product
4. Verify button changes to 'Remove'
5. Verify cart badge shows '1'

Expected Result

✓ Product is added to cart, button changes to 'Remove', cart badge shows 1.

Test #10 Add Multiple Products to Cart

High Regression

Description

Verify that multiple products can be added to the cart and the cart badge updates correctly.

TestNG Implementation

src/test/java/tests/ProductsTest.java

```
@Epic("Products")
@Feature("Add to Cart")
@Story("Multiple Products")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression"})
@Description("Verify adding multiple products to cart")
public void testAddMultipleProductsToCart() {
    logger.info("Adding multiple products to cart");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    productsPage.addProductToCart("Sauce Labs Bike Light");
    productsPage.addProductToCart("Sauce Labs Bolt T-Shirt");

```

```
    Assert.assertEquals(productsPage.getCartBadgeCount(), 3,
        "Cart badge should show 3");
}
```

Execution Flow

1. Login with valid credentials
2. Add first product to cart
3. Verify badge shows 1
4. Add second product to cart
5. Verify badge shows 2
6. Add third product to cart
7. Verify badge shows 3

Expected Result

✓ All products are added, cart badge shows correct count of 3.

Test #11 Remove Product from Products Page

High Regression

Description

Verify that a product can be removed from the cart via the products page.

TestNG Implementation

src/test/java/tests/ProductsTest.java

```
@Epic("Products")
@Feature("Remove from Cart")
@Story("Remove Product")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression"})
@Description("Verify removing product from products page")
public void testRemoveProductFromProductsPage() {
    logger.info("Removing product from products page");

    loginPage.login(config.getProperty("valid.username"),
```

```
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    Assert.assertEquals(productsPage.getCartBadgeCount(), 1);

    productsPage.removeProductFromCart("Sauce Labs Backpack");
    Assert.assertFalse(productsPage.isCartBadgeVisible(),
        "Cart badge should not be visible");
}
```

Execution Flow

1. Login and add product to cart
2. Verify cart badge shows 1
3. Click 'Remove' button on product
4. Verify button changes back to 'Add to cart'
5. Verify cart badge disappears

Expected Result

✓ Product is removed, button reverts to 'Add to cart', cart badge disappears.

Test #12 Sort Products by Price

Medium Regression

Description

Verify that products can be sorted by price (low to high and high to low).

TestNG Implementation

src/test/java/tests/ProductsTest.java

```
@Epic("Products")
@Feature("Product Sorting")
@Story("Sort by Price")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression"})
@Description("Verify sorting products by price")
public void testSortProductsByPrice() {
```

```
    logger.info("Testing product sorting by price");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    // Sort low to high
    productsPage.sortProducts("Price (low to high)");
    List<Double> pricesLowHigh = productsPage.getAllProductPrices();
    Assert.assertTrue(isSortedAscending(pricesLowHigh),
        "Prices should be sorted low to high");

    // Sort high to low
    productsPage.sortProducts("Price (high to low)");
    List<Double> pricesHighLow = productsPage.getAllProductPrices();
    Assert.assertTrue(isSortedDescending(pricesHighLow),
        "Prices should be sorted high to low");
}

private boolean isSortedAscending(List<Double> list) {
    for (int i = 0; i < list.size() - 1; i++) {
        if (list.get(i) > list.get(i + 1)) return false;
    }
    return true;
}
```

Execution Flow

1. Login with valid credentials
2. Click sort dropdown
3. Select 'Price (low to high)'
4. Get all product prices
5. Verify prices in ascending order
6. Select 'Price (high to low)'
7. Verify prices in descending order

Expected Result

✓ Products are correctly sorted by price in selected order.

Test #13 Cart Badge Count Updates

High Regression

Description

Verify that the cart badge count updates correctly when adding and removing products.

TestNG Implementation

src/test/java/tests/ProductsTest.java

```
@Epic("Products")
@Feature("Cart Badge")
@Story("Badge Count")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression"})
@Description("Verify cart badge count updates correctly")
public void testCartBadgeCountUpdates() {
    logger.info("Testing cart badge count updates");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    // Add products
    productsPage.addProductToCart("Sauce Labs Backpack");
    Assert.assertEquals(productsPage.getCartBadgeCount(), 1);

    productsPage.addProductToCart("Sauce Labs Bike Light");
    Assert.assertEquals(productsPage.getCartBadgeCount(), 2);

    // Remove one
    productsPage.removeProductFromCart("Sauce Labs Backpack");
    Assert.assertEquals(productsPage.getCartBadgeCount(), 1);

    // Remove last
    productsPage.removeProductFromCart("Sauce Labs Bike Light");
    Assert.assertFalse(productsPage.isCartBadgeVisible());
}
```

Execution Flow

1. Login with valid credentials
2. Add first product, verify badge shows 1
3. Add second product, verify badge shows 2
4. Remove first product, verify badge shows 1

5. Remove second product, verify badge disappears

Expected Result

✓ Cart badge count accurately reflects the number of items in cart.

6. Cart Tests

3 test cases in this category

Test #14 View Cart Items

Critical Smoke Regression

Description

Verify that clicking the cart icon displays all added items with correct details.

TestNG Implementation

src/test/java/tests/CartTest.java

```
@Epic("Shopping Cart")
@Feature("View Cart")
@Story("Cart Contents")
@Severity(SeverityLevel.CRITICAL)
@Test(groups = {"smoke", "regression"})
@Description("Verify viewing cart items")
public void testViewCartItems() {
    logger.info("Testing view cart items");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    productsPage.addProductToCart("Sauce Labs Bike Light");

    CartPage cartPage = productsPage.clickCartIcon();

    Assert.assertEquals(cartPage.getItemCount(), 2,
        "Cart should have 2 items");
    Assert.assertTrue(cartPage.isProductInCart("Sauce Labs
Backpack"));
```

```
    Assert.assertTrue(cartPage.isProductInCart("Sauce Labs Bike
Light"));
}
```

Execution Flow

1. Login and add products to cart
2. Click cart icon in header
3. Verify cart page loads
4. Count items in cart
5. Verify each product name visible
6. Verify product prices correct

Expected Result

✓ Cart page displays all added items with correct names, quantities, and prices.

Test #15 Proceed to Checkout from Cart

Critical Smoke Regression

Description

Verify that clicking the checkout button from cart navigates to the checkout page.

TestNG Implementation

src/test/java/tests/CartTest.java

```
@Epic("Shopping Cart")
@Feature("Checkout")
@Story("Proceed to Checkout")
@Severity(SeverityLevel.CRITICAL)
@Test(groups = {"smoke", "regression"})
@Description("Verify proceeding to checkout from cart")
public void testProceedToCheckout() {
    logger.info("Testing proceed to checkout");
```

```
    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    CartPage cartPage = productsPage.clickCartIcon();

    CheckoutPage checkoutPage = cartPage.clickCheckout();

    Assert.assertTrue(checkoutPage.isCheckoutInfoPageDisplayed(),
        "Checkout information page should be displayed");
}
```

Execution Flow

1. Login and add product to cart
2. Navigate to cart page
3. Click 'Checkout' button
4. Verify checkout info page loads
5. Verify input fields visible

Expected Result

✓ Checkout information page displays with first name, last name, and postal code fields.

Test #16 Continue Shopping from Cart

Medium Regression

Description

Verify that clicking 'Continue Shopping' returns to the products page.

TestNG Implementation

src/test/java/tests/CartTest.java

```
@Epic("Shopping Cart")
@Feature("Navigation")
@Story("Continue Shopping")
```

```
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression"})
@Description("Verify continue shopping from cart")
public void testContinueShopping() {
    logger.info("Testing continue shopping");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    CartPage cartPage = productsPage.clickCartIcon();

    ProductsPage returnedProductsPage =
cartPage.clickContinueShopping();

    Assert.assertTrue(returnedProductsPage.isPageLoaded(),
        "Should return to products page");
    Assert.assertEquals(returnedProductsPage.getCartBadgeCount(), 1,
        "Cart should still have 1 item");
}
```

Execution Flow

1. Login and add product to cart
2. Navigate to cart page
3. Click 'Continue Shopping' button
4. Verify products page loads
5. Verify cart badge still shows item count

Expected Result

✓ User returns to products page with cart items preserved.

7. Checkout Tests

4 test cases in this category

Test #17 Complete Checkout with Valid Info

Critical Smoke Regression

Description

Verify that a complete checkout flow with valid information results in order confirmation.

TestNG Implementation

src/test/java/tests/CheckoutTest.java

```
@Epic("Checkout")
@Feature("Complete Purchase")
@Story("Successful Checkout")
@Severity(SeverityLevel.CRITICAL)
@Test(groups = {"smoke", "regression"})
@Description("Verify complete checkout with valid information")
public void testCompleteCheckout() {
    logger.info("Testing complete checkout flow");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    CartPage cartPage = productsPage.clickCartIcon();
    CheckoutPage checkoutPage = cartPage.clickCheckout();

    checkoutPage.enterFirstName("John");
    checkoutPage.enterLastName("Doe");
    checkoutPage.enterPostalCode("12345");
    checkoutPage.clickContinue();

    // Verify overview page
    Assert.assertTrue(checkoutPage.isOverviewPageDisplayed());
    Assert.assertTrue(checkoutPage.isProductVisible("Sauce Labs
Backpack"));

    // Complete purchase
    checkoutPage.clickFinish();

    Assert.assertTrue(checkoutPage.isOrderConfirmed(),
        "Order confirmation should be displayed");
    Assert.assertEquals(checkoutPage.getConfirmationMessage(),
        "Thank you for your order!");
}
```

Execution Flow

1. Login and add product to cart

2. Navigate to cart and click checkout
3. Enter first name, last name, postal code
4. Click continue to overview
5. Verify product in order summary
6. Click finish to complete order
7. Verify confirmation message

Expected Result

✓ Order is placed successfully and confirmation message 'Thank you for your order!' is displayed.

Test #18 Checkout Missing First Name Error

High | Regression | Negative

Description

Verify that attempting checkout without first name displays a validation error.

TestNG Implementation

src/test/java/tests/CheckoutTest.java

```
@Epic("Checkout")
@Feature("Form Validation")
@Story("Missing First Name")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression", "negative"})
@Description("Verify error when first name is missing")
public void testMissingFirstName() {
    logger.info("Testing missing first name validation");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    CartPage cartPage = productsPage.clickCartIcon();
    CheckoutPage checkoutPage = cartPage.clickCheckout();

```

```
    checkoutPage.enterLastName("Doe");
    checkoutPage.enterPostalCode("12345");
    checkoutPage.clickContinue();

    Assert.assertTrue(checkoutPage.getErrorMessage()
        .contains("First Name is required"));
}
```

Execution Flow

1. Login and add product to cart
2. Navigate to checkout info page
3. Leave first name empty
4. Enter last name and postal code
5. Click continue
6. Verify error message displays

Expected Result

✓ Error message 'Error: First Name is required' is displayed.

Test #19 Checkout Missing Postal Code Error

High | Regression | Negative

Description

Verify that attempting checkout without postal code displays a validation error.

TestNG Implementation

src/test/java/tests/CheckoutTest.java

```
@Epic("Checkout")
@Feature("Form Validation")
@Story("Missing Postal Code")
@Severity(SeverityLevel.NORMAL)
@Test(groups = {"regression", "negative"})
@Description("Verify error when postal code is missing")
```

```
public void testMissingPostalCode() {
    logger.info("Testing missing postal code validation");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    CartPage cartPage = productsPage.clickCartIcon();
    CheckoutPage checkoutPage = cartPage.clickCheckout();

    checkoutPage.enterFirstName("John");
    checkoutPage.enterLastName("Doe");
    checkoutPage.clickContinue();

    Assert.assertTrue(checkoutPage.getErrorMessage()
        .contains("Postal Code is required"));
}
```

Execution Flow

1. Login and add product to cart
2. Navigate to checkout info page
3. Enter first name and last name
4. Leave postal code empty
5. Click continue
6. Verify error message displays

Expected Result

✓ Error message 'Error: Postal Code is required' is displayed.

Test #20 Order Confirmation Message

Critical | Smoke | Regression

Description

Verify that after completing checkout, the order confirmation page displays with correct messaging.

TestNG Implementation

src/test/java/tests/CheckoutTest.java

```
@Epic("Checkout")
@Feature("Order Confirmation")
@Story("Confirmation Page")
@Severity(SeverityLevel.CRITICAL)
@Test(groups = {"smoke", "regression"})
@Description("Verify order confirmation page")
public void testOrderConfirmationPage() {
    logger.info("Testing order confirmation page");

    loginPage.login(config.getProperty("valid.username"),
                    config.getProperty("valid.password"));

    productsPage.addProductToCart("Sauce Labs Backpack");
    CartPage cartPage = productsPage.clickCartIcon();
    CheckoutPage checkoutPage = cartPage.clickCheckout();

    checkoutPage.enterFirstName("John");
    checkoutPage.enterLastName("Doe");
    checkoutPage.enterPostalCode("12345");
    checkoutPage.clickContinue();
    checkoutPage.clickFinish();

    Assert.assertTrue(checkoutPage.isConfirmationImageVisible(),
        "Confirmation image should be visible");
    Assert.assertEquals(checkoutPage.getConfirmationHeader(),
        "Thank you for your order!");
    Assert.assertTrue(checkoutPage.getConfirmationText()
        .contains("Your order has been dispatched"));
    Assert.assertTrue(checkoutPage.isBackHomeButtonVisible(),
        "Back Home button should be visible");
}
```

Execution Flow

1. Complete full checkout flow
2. Verify confirmation page loads
3. Verify header text
4. Verify confirmation message
5. Verify confirmation image visible
6. Verify Back Home button visible

Expected Result

✓ Order confirmation page displays with thank you message, dispatch info, image, and back home button.

8. Framework Features Tests

8 test cases in this category

Test #21 Screenshot on Failure

High Framework

Description

Verify that screenshots are automatically captured when a test fails.

TestNG Implementation

src/test/java/listeners/AllureListener.java

```
public class AllureListener implements ITestListener {

    @Override
    public void onTestFailure(ITestResult result) {
        Object testClass = result.getInstance();
        WebDriver driver = ((BaseTest) testClass).getDriver();

        if (driver instanceof TakesScreenshot) {
            byte[] screenshot = ((TakesScreenshot) driver)
                .getScreenshotAs(OutputType.BYTES);

            Allure.addAttachment(
                "Screenshot on Failure - " + result.getName(),
                "image/png",
                new ByteArrayInputStream(screenshot),
                "png"
            );

            logger.error("Test failed: " + result.getName());
            logger.info("Screenshot captured and attached to Allure
report");
        }
    }
```

```
    @Override
    public void onTestSuccess(ITestResult result) {
        logger.info("Test passed: " + result.getName());
    }
}
```

Execution Flow

1. Test executes and fails
2. Listener/Hook catches failure event
3. WebDriver captures screenshot as bytes
4. Screenshot attached to report
5. Failure logged with details

Expected Result

✓ Screenshot is automatically captured and attached to the test report when a test fails.

Test #22 Parallel Execution Support

High Framework

Description

Verify that tests can be executed in parallel across multiple threads.

TestNG Implementation

testng.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="SauceDemo Test Suite" parallel="methods" thread-
count="3">

    <listeners>
        <listener class-name="listeners.AllureListener"/>
    </listeners>
```

```
    <test name="Smoke Tests">
        <groups>
            <run>
                <include name="smoke"/>
            </run>
        </groups>
        <classes>
            <class name="tests.LoginTest"/>
            <class name="tests.ProductsTest"/>
            <class name="tests.CartTest"/>
            <class name="tests.CheckoutTest"/>
        </classes>
    </test>

    <test name="Regression Tests">
        <groups>
            <run>
                <include name="regression"/>
            </run>
        </groups>
        <classes>
            <class name="tests.LoginTest"/>
            <class name="tests.ProductsTest"/>
            <class name="tests.CartTest"/>
            <class name="tests.CheckoutTest"/>
        </classes>
    </test>
</suite>
```

Execution Flow

1. Test suite starts
2. Thread pool created (3 threads)
3. Tests distributed across threads
4. Each thread gets unique WebDriver
5. Tests execute concurrently
6. Results aggregated

Expected Result

✓ Tests execute in parallel, reducing overall execution time.

Test #23 Cross-Browser Testing

Medium Framework

Description

Verify that tests can be executed across different browsers (Chrome, Firefox, Edge).

TestNG Implementation

src/test/java/factory/DriverFactory.java

```
public class DriverFactory {
    private static ThreadLocal<WebDriver> driver = new ThreadLocal<>
();

    public static WebDriver initDriver(String browser) {
        WebDriver webDriver;

        switch (browser.toLowerCase()) {
            case "chrome":
                WebDriverManager.chromedriver().setup();
                ChromeOptions chromeOptions = new ChromeOptions();
                chromeOptions.addArguments("--headless");
                chromeOptions.addArguments("--disable-gpu");
                webDriver = new ChromeDriver(chromeOptions);
                break;

            case "firefox":
                WebDriverManager.firefoxdriver().setup();
                FirefoxOptions firefoxOptions = new FirefoxOptions();
                firefoxOptions.addArguments("--headless");
                webDriver = new FirefoxDriver(firefoxOptions);
                break;

            case "edge":
                WebDriverManager.edgedriver().setup();
                EdgeOptions edgeOptions = new EdgeOptions();
                edgeOptions.addArguments("--headless");
                webDriver = new EdgeDriver(edgeOptions);
                break;

            default:
                throw new IllegalArgumentException(
                    "Browser not supported: " + browser);
        }
```

```
        webDriver.manage().window().maximize();
        webDriver.manage().timeouts()
            .implicitlyWait(Duration.ofSeconds(10));
        driver.set(webDriver);

        return webDriver;
    }

    public static WebDriver getDriver() {
        return driver.get();
    }

    public static void quitDriver() {
        if (driver.get() != null) {
            driver.get().quit();
            driver.remove();
        }
    }
}
```

Execution Flow

1. Read browser config from properties/env
2. DriverFactory creates appropriate driver
3. WebDriverManager handles driver binaries
4. Browser-specific options applied
5. Tests execute on selected browser
6. Browser closed after tests

Expected Result

✓ Tests can be executed on Chrome, Firefox, or Edge browsers via configuration.

Test #24 Allure Test Reports

High Framework

Description

Verify that Allure reports are generated with test results, screenshots, and metadata.

TestNG Implementation

pom.xml (excerpt)

```
<!-- Allure Dependencies -->
<dependency>
    <groupId>io.qameta.allure</groupId>
    <artifactId>allure-testng</artifactId>
    <version>2.24.0</version>
</dependency>

<!-- Allure Maven Plugin -->
<plugin>
    <groupId>io.qameta.allure</groupId>
    <artifactId>allure-maven</artifactId>
    <version>2.12.0</version>
    <configuration>
        <reportVersion>2.24.0</reportVersion>
    </configuration>
</plugin>

<!-- Usage in tests -->
@Epic("Authentication")
@Feature("Login")
@Story("Valid Login")
@Severity(SeverityLevel.CRITICAL)
@Description("Test description here")
@Step("Step description")
public void testMethod() {
    // Test code
}
```

Execution Flow

1. Tests execute with Allure annotations
2. Results written to allure-results folder
3. Screenshots attached on failure
4. Run 'mvn allure:serve'
5. HTML report generated and opened
6. View test history and trends

Expected Result

✓ Comprehensive Allure reports generated with test results, screenshots, history, and trends.

Test #25 Logging Implementation

Medium Framework

Description

Verify that comprehensive logging is implemented for test execution tracking.

TestNG Implementation

src/main/resources/log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Properties>
        <Property name="LOG_PATTERN">
            %d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n
        </Property>
    </Properties>

    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="${LOG_PATTERN}"/>
        </Console>

        <RollingFile name="RollingFile"
                     fileName="logs/test-automation.log"
                     filePattern="logs/test-automation-%d{yyyy-MM-dd}-
%i.log.gz">
            <PatternLayout pattern="${LOG_PATTERN}"/>
            <Policies>
                <SizeBasedTriggeringPolicy size="10MB"/>
                <TimeBasedTriggeringPolicy interval="1"/>
            </Policies>
            <DefaultRolloverStrategy max="10"/>
        </RollingFile>
    </Appenders>

    <Loggers>
        <Logger name="tests" level="INFO" additivity="false">
```

```
            <AppenderRef ref="Console"/>
            <AppenderRef ref="RollingFile"/>
        </Logger>
        <Root level="INFO">
            <AppenderRef ref="Console"/>
            <AppenderRef ref="RollingFile"/>
        </Root>
    </Loggers>
</Configuration>
```

Execution Flow

1. Logger initialized in test class
2. Info/debug/error logs recorded
3. Logs output to console
4. Logs written to rolling file
5. Files rotated by date/size
6. Historical logs preserved

Expected Result

✓ Comprehensive logs are generated in console and file with proper formatting and rotation.

Test #26 CI/CD Pipeline Integration

High Framework

Description

Verify that the framework integrates with Jenkins CI/CD pipeline for automated execution.

TestNG Implementation

Jenkinsfile

```
pipeline {
    agent any
```

```
    tools {
        maven 'Maven-3.9.4'
        jdk 'JDK-17'
    }

    environment {
        BROWSER = 'chrome'
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main',
                    url: 'https://github.com/user/selenium-testng-
framework.git'
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean compile'
            }
        }

        stage('Run Tests') {
            steps {
                sh 'mvn test -Dbrowser=${BROWSER} -
DsuiteXmlFile=testng.xml'
            }
            post {
                always {
                    publishHTML([
                        reportDir: 'target/surefire-reports',
                        reportFiles: 'index.html',
                        reportName: 'TestNG Report'
                    ])
                }
            }
        }

        stage('Allure Report') {
            steps {
                allure includeProperties: false,
                       jdk: '',
                       results: [[path: 'target/allure-results']]
            }
        }
    }

    post {
        always {
```

```
            cleanWs()
        }
        failure {
            emailext subject: 'Test Failure',
                     body: 'Tests failed. Check Jenkins for details.',
                     to: 'team@example.com'
        }
    }
}
```

Execution Flow

1. Jenkins job triggered (manual/scheduled)
2. Checkout code from repository
3. Maven compiles project
4. Tests execute with parameters
5. Reports generated and published
6. Notifications sent on failure

Expected Result

✓ Tests run automatically in Jenkins with reports published and notifications configured.

Test #27 Data-Driven Testing

Medium Framework

Description

Verify that tests support data-driven execution using external data sources.

TestNG Implementation

src/test/java/tests/DataDrivenTest.java

```
public class DataDrivenTest extends BaseTest {

    @DataProvider(name = "loginData")
    public Object[][] getLoginData() {
```

```
        return new Object[][] {
            {"standard_user", "secret_sauce", true, "Products"},
            {"locked_out_user", "secret_sauce", false, "locked out"},
            {"invalid_user", "wrong_pass", false, "do not match"},
            {"", "secret_sauce", false, "Username is required"},
            {"standard_user", "", false, "Password is required"}
        };
    }

    @Test(dataProvider = "loginData", groups = {"regression", "data-
driven"})
    @Description("Data-driven login test")
    public void testLoginWithData(String username, String password,
                                  boolean shouldPass, String
expectedText) {
        logger.info("Testing login with: " + username);

        loginPage.enterUsername(username);
        loginPage.enterPassword(password);
        loginPage.clickLoginButton();

        if (shouldPass) {
            Assert.assertEquals(productsPage.getPageTitle(),
expectedText);
        } else {

Assert.assertTrue(loginPage.getErrorMessage().contains(expectedText));
        }
    }

    // Excel DataProvider
    @DataProvider(name = "excelData")
    public Object[][] getExcelData() throws IOException {
        return ExcelUtils.getTestData("testdata.xlsx", "LoginTests");
    }
}
```

Execution Flow

1. Test reads data from DataProvider/Examples
2. Test executes once per data row
3. Each iteration uses different credentials
4. Results validated against expected outcome
5. All iterations reported separately

Expected Result

✓ Tests execute multiple times with different data sets, all iterations tracked in reports.

Test #28 Tagged Test Execution

Medium Framework

Description

Verify that tests can be selectively executed using tags or groups.

TestNG Implementation

testng.xml

```
<!-- Running specific groups -->
<suite name="Tagged Tests">
    <test name="Smoke Tests Only">
        <groups>
            <run>
                <include name="smoke"/>
            </run>
        </groups>
        <classes>
            <class name="tests.LoginTest"/>
            <class name="tests.ProductsTest"/>
        </classes>
    </test>

    <test name="Negative Tests Only">
        <groups>
            <run>
                <include name="negative"/>
            </run>
        </groups>
        <classes>
            <class name="tests.LoginTest"/>
            <class name="tests.CheckoutTest"/>
        </classes>
    </test>
</suite>

<!-- Command line execution -->
// mvn test -Dgroups=smoke
```

```
// mvn test -Dgroups=regression,negative
// mvn test -DexcludedGroups=slow
```

Execution Flow

1. Specify tags via config or command line
2. Test runner filters tests by tags
3. Only matching tests execute
4. Other tests skipped
5. Report shows executed subset

Expected Result

✓ Only tests matching specified tags/groups are executed, others are skipped.

9. Utilities

ConfigReader.java

```
public class ConfigReader {
    private static Properties properties;

    static {
        try {
            String configPath = "src/main/resources/config.properties";
            FileInputStream fis = new FileInputStream(configPath);
            properties = new Properties();
            properties.load(fis);
        } catch (IOException e) {
            throw new RuntimeException("Config file not found", e);
        }
    }

    public static String getProperty(String key) {
        String value = System.getProperty(key);
        if (value == null) {
            value = properties.getProperty(key);
        }
        return value;
    }
```

```
    public static String getProperty(String key, String defaultValue) {
        String value = getProperty(key);
        return value != null ? value : defaultValue;
    }
}
```

DriverFactory.java

```
public class DriverFactory {
    private static ThreadLocal<WebDriver> driver = new ThreadLocal<>();

    public static WebDriver initDriver(String browser) {
        WebDriver webDriver;

        switch (browser.toLowerCase()) {
            case "chrome":
                WebDriverManager.chromedriver().setup();
                ChromeOptions options = new ChromeOptions();
                options.addArguments("--headless");
                options.addArguments("--no-sandbox");
                options.addArguments("--disable-dev-shm-usage");
                webDriver = new ChromeDriver(options);
                break;
            case "firefox":
                WebDriverManager.firefoxdriver().setup();
                webDriver = new FirefoxDriver();
                break;
            case "edge":
                WebDriverManager.edgedriver().setup();
                webDriver = new EdgeDriver();
                break;
            default:
                throw new RuntimeException("Browser not supported");
        }

        webDriver.manage().window().maximize();
        webDriver.manage().timeouts()
            .implicitlyWait(Duration.ofSeconds(10));
        driver.set(webDriver);
        return webDriver;
    }

    public static WebDriver getDriver() {
        return driver.get();
    }

    public static void quitDriver() {
        if (driver.get() != null) {
```

```
            driver.get().quit();
            driver.remove();
        }
    }
}
```

10. Configuration

config.properties

```
# Application Configuration
base.url=https://www.saucedemo.com
valid.username=standard_user
valid.password=secret_sauce

# Browser Configuration
browser=chrome
headless=true

# Timeout Configuration
implicit.wait=10
explicit.wait=15
page.load.timeout=30
```

testng.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="SauceDemo Test Suite" parallel="methods" thread-count="3">
    <listeners>
        <listener class-name="listeners.AllureListener"/>
    </listeners>

    <test name="All Tests">
        <classes>
            <class name="tests.LoginTest"/>
            <class name="tests.ProductsTest"/>
            <class name="tests.CartTest"/>
            <class name="tests.CheckoutTest"/>
        </classes>
    </test>
</suite>
```

11. CI/CD Pipeline

Jenkinsfile

```
pipeline {
    agent any

    tools {
        maven 'Maven-3.9.4'
        jdk 'JDK-17'
    }

    environment {
        BROWSER = 'chrome'
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main',
                    url: 'https://github.com/user/selenium-testng-
framework.git'
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean compile'
            }
        }

        stage('Run Tests') {
            steps {
                sh 'mvn test -Dbrowser=${BROWSER} -
DsuiteXmlFile=testng.xml'
            }
            post {
                always {
                    publishHTML([
                        reportDir: 'target/surefire-reports',
                        reportFiles: 'index.html',
                        reportName: 'TestNG Report'
                    ])
                }
            }
        }

        stage('Allure Report') {
```

```
            steps {
                allure includeProperties: false,
                       jdk: '',
                       results: [[path: 'target/allure-results']]
            }
        }
    }

    post {
        always {
            cleanWs()
        }
        failure {
            emailext subject: 'Test Failure',
                     body: 'Tests failed. Check Jenkins for details.',
                     to: 'team@example.com'
        }
    }
}
```

Running Tests

Command Line Execution

```
# Run all tests
mvn clean test

# Run smoke tests only
mvn test -Dgroups=smoke

# Run regression tests
mvn test -Dgroups=regression

# Run on specific browser
mvn test -Dbrowser=firefox

# Run parallel (3 threads)
mvn test -Dparallel=methods -DthreadCount=3

# Generate Allure report
mvn allure:serve
```

SauceDemo Selenium Automation Framework

TestNG Framework Documentation