Automation Test Scripts

PROJECT A

General information	
Customer	<Project name>
Created by (Author)	
Preparation date	
Version	
Status	

Revision History

Version	Description	Author	Date	Approved by	
				Author	Date

Summary

1. General information

The sample is a Maven project. The project has the following dependencies implemented: Selenide (framework for test automation), JUnit (framework for testing), AssertJ (library that provides a rich set of assertions) and Allure (framework for report generation)

2. Project structure

For project structure we will use Page Object design pattern.

It is a Design Pattern which has become popular in test automation for enhancing test maintenance and reducing code duplication. A page object is an object-oriented class that serves as an interface to a page of your AUT. The tests then use the methods of this page object class whenever they need to interact with the UI of that page. The benefit is that if the UI changes for the page, the tests themselves don't need to change, only the code within the page object needs to change. Subsequently all changes to support that new UI are located in one place.

The Page Object Design Pattern provides the following advantages:

- There is a clean separation between test code and page specific code such as locators (or their use if you're using a UI Map) and layout.
- There is a single repository for the services or operations offered by the page rather than having these services scattered throughout the tests.

3. Login page tests examples

First of all let's create our tests class, that extends Base Test class, give it meaningful name and inject all required dependencies (page objects with locators and methods)

```
package com.example.web.tests.account;

import ...

@DisplayName("Log in tests")
public class LoginTests extends BaseTest {
  @Inject
  private HomePage homePage;
  @Inject
  private NavBarComponent navBar;
  @Inject
  private AuthorizationPage authorizationPage;
  @Inject
  private MyAccountPage myAccountPage;
```

After that we can start developing of automated test cases:

User can access Login form test

```
@Test
@DisplayName("User can access Login form")
public void userCanAccessLoginFormTest() {
  homePage.navigate();
  navBar.clickCustomerPortal();
  authorizationPage.getLoginButton().shouldBe(Condition.visible);

  SoftAssertions.assertSoftly(softAssertions -> {
    softAssertions.assertThat(authorizationPage.getLoginTab().isDisplayed())
        .as("Login tab should be displayed on the page")
        .isEqualTo(true);
    softAssertions.assertThat(authorizationPage.getSignUpTab().isDisplayed())
        .as("Signup tab should be displayed on the page")
        .isEqualTo(true);
    softAssertions.assertThat(authorizationPage.getLoginEmailField().isDisplayed())
        .as("Username input should be displayed on the page")
        .isEqualTo(true);
    softAssertions.assertThat(authorizationPage.getLoginPasswordField().isDisplayed())
        .as("Password input should be displayed on the page")
        .isEqualTo(true);
    softAssertions.assertThat(authorizationPage.getForgotPasswordLink().isDisplayed()).
        as("Forgot password link should be displayed on the page")
        .isEqualTo(true);
    softAssertions.assertThat(authorizationPage.getLoginButton().isDisplayed())
        .as("Login button should be displayed on the page")
        .isEqualTo(true);
  });
}
```

Log in with valid credentials test

```
@Test
@DisplayName("Log in with valid credentials")
public void loginValidCredentialsTest() {
  homePage.navigate();
  navBar.clickCustomerPortal();
  authorizationPage.getLoginTab().shouldBe(Condition.visible);
  authorizationPage.logIn("username", "password");

  assertThat(myAccountPage.getLogoutButton().isDisplayed())
      .as("Log out button should be displayed in case of successful login")
      .isEqualTo(true);
}
```

Log in with invalid username test

This test will fail on purpose for reporting capabilities demonstration

```
@Test
@DisplayName("Log in with invalid username test")
public void loginInvalidUsernameTest() {
  homePage.navigate();
  navBar.clickCustomerPortal();
  authorizationPage.getLoginTab().shouldBe(Condition.visible);
  authorizationPage.logIn("example", "example");

  assertThat(authorizationPage.getErrorMessage().getText())
        .as("Error should be shown in case of invalid username")
        .isEqualTo("Unbekannter Benutzername");
}
```

Log in with invalid password test

```
@Test
@DisplayName("Log in with invalid password test")
public void loginInvalidPasswordTest() {
  homePage.navigate();
  navBar.clickCustomerPortal();
  authorizationPage.getLoginTab().shouldBe(Condition.visible);
  authorizationPage.logIn("username", "wrongPass");

  assertThat(authorizationPage.getErrorMessage().getText())
        .as("Error should be shown in case of invalid password")
        .contains("Fehler: Das Passwort, das du für den Benutzernamen username
eingegeben hast, ist nicht korrekt. Passwort vergessen?");
}
```

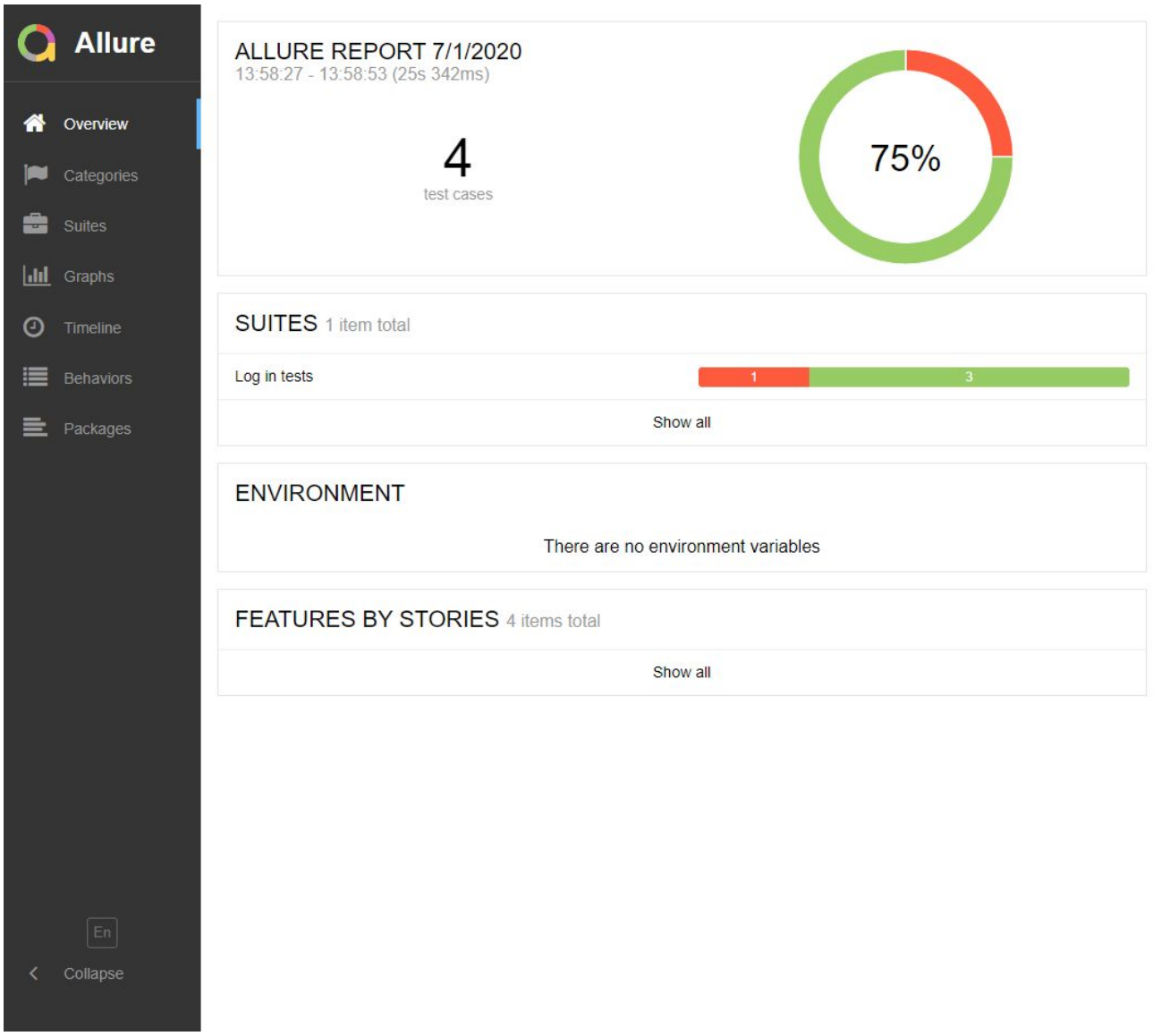
4. Test report example

After running those tests on desired browser locally or inside CI/CD pipeline using simple command as:

mvn clean test -Dselenide.browser=chrome

We can obtain results summary and details in the Allure:

Summary:



Details: