SOFTWARE VERIFICATION, VALIDATION AND TESTING

TESTING DOCUMENTATION

“ TECHNO SHOP MARKET „

Prepared by:
Amela Redžić
Namir Hlivnjak
Dinela Hadžajlić

Proposed to:
Samed Jukić, Assist. Prof. Dr.
Aldin Kovačević, Teaching Assistant

22.01.2023.

TABLE OF CONTENTS

Contents

1. Introduction

1.1. About the Project

The project which are we testing is a techno shop market application. It is an e-commerce platform that allows users to browse and purchase various technology products such as smartphones, laptops, and other gadgets. We are currently testing a specific pages of this application, which could be a feature such as the product catalog, the shopping cart, or the checkout process. The goal of our testing is to ensure that the page functions correctly and provides a smooth user experience.

LINK TO PAGE :

1.2. Project Functionalities and Screenshots

Describe or list the main features of the project and provide a few screenshots of those features.

2. Test Plan

2.1. Scope

Every project can test the entire application or focus on specific parts/features of it, depending on the requirements, constraints and the resources available. It's important to prioritize the most important features and test them thoroughly. Our goal is to test this application to the end and to put all functions and features to high level of work with full potential. In our project we will provide full testing of pages and functions which this market platform have. During testing, the goal is to find all possible weak points and repair them. In the multitude of important factors of each page, we aim to focus especially on the basic things and functions that are important to all users. This includes optimization, loading speed, security and appearance.

2.2. Testing Environment and Tools

In out project, we are using Java as the programming language and Eclipse as the development framework. Eclipse is a popular open-source Integrated Development Environment (IDE) that supports a wide range of programming languages, including Java. It provides a powerful set of tools for editing, debugging, and testing code, and it's widely used by developers around the world. Additionally, more people but definitely some of our college used some other tools and frameworks like JUnit, TestNG, or Selenium for testing and Maven or Gradle for managing dependencies, building and deploying application.

3. Test Execution

In the “Test Execution” section, you need to list your **test scenarios** and **test cases**. *Test scenario* is a statement whose purpose is to inform what particular area of the application will be tested. Test scenarios consist of multiple *test cases*, which test specific parts of an application. You can use the following template approach for test scenarios and test cases, but you can also use a custom template or your own “design”, as long as you have all the necessary information.

3.1. Test Scenario Name

Give a brief one-two sentence description of the test scenario. Afterwards, list the test cases in that test scenario. You can use the following table.

Test Name: Short test name				
Description: Test description				
Pre-condition(s): Are there any conditions or other tests that need to be executed before this test; are there any test fixtures? If not, leave this blank.				
Test Steps: 1. Number the steps needed to execute this test case. 2. Like this.	**Test Data:** - is there any specific input data you are using for this test (e.g. some input field values, credentials, etc.)	**Expected Result:** What was the expected result of this test?	**Actual Result:** What was the actual result of this test, once you executed it?	**Status:** PASS or FAIL (you can color code them)
Notes: Are there any notes about this test you would like to add? If not, leave this blank.				

Provide a screenshot of your test code after the test case table.

The following section will be one test scenario example for you, with two tests.

3.2. Example Scenario: Visit the Home Page

Users want to be able to visit the home page, and get a quick glance at the most important features it has to offer.

Test Name: Test sidebar links				
Description: Check if all sidebar items are clickable and lead to their respective page				
Pre-condition(s):				
Test Steps: 1. Go to the landing page 2. Check if all links have been rendered 3. Click on each link individually 4. Check if the URL of the opened page corresponds to the link in the sidebar	**Test Data:**	**Expected Result:** The user is taken to the correct new page when clicking on the link.	**Actual Result:** The user is taken to the correct new page when clicking on the link.	**Status:** PASS
Notes:				

```java
    @Test
    public void SidebarLinksTest() {
        System.out.println("All sidebar items lead to their respective pages");

        String[] links = {"landing", "shop", "cart", "about", "contact"};


        for (String link : links) {
            String finalUrl = "https://flink-webshop.herokuapp.com/#" +  link;
            driver.findElement(By.id(link+"-link")).click();

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            assertEquals(finalUrl, driver.getCurrentUrl());
        }

    }
```

Test Name: Test “Find out more” button				
Description: Check if “Find out more” button works and leads to proper page (either a product page, or an “About us” page)				
Pre-condition(s):				
Test Steps: 1. Wait until “Find out more” button is loaded 2. Click on the button 3. Check if it leads to the expected page	**Test Data:**	**Expected Result:** The user is taken to a new page.	**Actual Result:** The page gives a 404 error because the requested page does not exist.	**Status:** FAIL
Notes:				

```java
    @Test
    public void FindOutMoreButtonTest() {
        System.out.println("General UI part testing");

        String finalUrl = "https://flink-webshop.herokuapp.com/#product";

         try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        if(driver.findElement(By.linkText("FIND OUT MORE")).isDisplayed()) {
            driver.findElement(By.linkText("FIND OUT MORE")).click();

            assertEquals(finalUrl, driver.getCurrentUrl());
        }
    }
```

...

9. Conclusion

9.1. Testing Summary

Provide a summary of all your executed tests. Something like this would be alright:

Testing Tool	Total Tests	Passed Tests	Failed Tests
Framework or tool(s) used for testing. If you wrote tests in multiple different tools, create a row with the number of tests for each of them.	total number of tests	# of passing tests	# of failing tests

If there are failing tests, provide a list of their names. Ideally, you can also create bookmarks in the document and make links to those failing test cases.

9.2. Final Thoughts

In the end, we would like to point out that we are especially glad that we participated in a project like this and where we were able to learn some new things when it comes to web sites. Namely, during the entire project, we aimed to fulfill all the requirements that a single user might have. With this project, we conclude that it is very necessary to do a detailed analysis before the launch of each project in order to prevent all possible errors and flaws of a website, application, etc. It's important to keep in mind that, thorough testing can help identify and fix any issues early on and that can save time and money in the long run. Additionally, testing is a critical step in ensuring that the application is of high quality and that it meets the needs of its users.

Provide some closing statements or your final thoughts about the project you tested. Did you find that it was implemented well? Did you notice any obvious mistakes or flaws in it? Are there some recommendations or observations you would note for the site?