**Demo Blaze Software Test Plan**

**Project Name:** Demo Blaze Web App Software Test Plan (STP)

**Prepared by:** Ehab Khalil

**Document Created:** 2024-08-02

**Table of Contents**

1. **Introduction**
   * Objective
   * Scope
2. **Features to be Tested**
   * 2.1 User Authentication
   * 2.2 Product Categories
   * 2.3 Shopping Cart
   * 2.4 Order Placement
   * 2.5 User Profile Management
   * 2.6 Product Search
   * 2.7 API Testing
3. **Approach**
   * 3.1 Testing Levels
   * 3.2 Testing Types
   * 3.3 Testing Techniques
   * 3.4 Test Automation
   * 3.5 Test Environment
   * 3.6 Entry and Exit Criteria
   * 3.7 Risk Management
   * 3.8 Item Pass/Fail Criteria
4. **Item Pass/Fail Criteria**
   * 4.1 Pass Criteria
   * 4.2 Fail Criteria
5. **Responsibilities**
   * 5.1 Yosef (Manager)
   * 5.2 Tzahi (Manager)
   * 5.3 Ehab (QA Automation Engineer)
   * 5.4 Majd (QA Automation Engineer)
   * 5.5 Rest of the Team
6. **Summary**

**1. Software Test Plan (STP)**

The Software Test Plan (STP) outlines the strategy, approach, resources, and schedule for the testing activities of the DemoBlaze web application. It includes the following sections:

**1. Introduction**

* **Objective:** The objective of this test plan is to outline the testing strategy, approach, resources, schedule, and scope for the DemoBlaze web application. The purpose is to ensure that the DemoBlaze application meets its requirements and functions correctly before it is deployed to production.
* **Scope:** This test plan covers the functional testing of the DemoBlaze web application. The focus will be on validating key functionalities such as user login, sign up, product categories, cart management, order processing, and logout processes.

**2. Features to be Tested**

1. **User Authentication**

1.1. **Sign Up**: Testing sign up with valid and invalid credentials.

1.2. **Login**: Testing user login functionality with valid and invalid credentials.

1.3. **Logout**: Ensuring the user can log out successfully.

1.4. **Register**: Testing registration with valid and invalid credentials.

1. **Product Browsing**

2.1. **Category Navigation**: Testing navigation through product categories (phones, laptops, monitors).

2.2. **Product Details**: Verifying product details display correctly.

1. **Cart Management**

3.1. **Add to Cart**: Testing adding products to the cart.

3.2. **View Cart**: Ensuring the cart displays the correct items.

3.3. **Remove from Cart**: Testing the removal of items from the cart.

1. **Order Processing**

4.1. **Place Order**: Verifying the order placement process.

4.2. **Order Confirmation**: Ensuring the order confirmation is displayed correctly.

1. **Contact The Store**

5.1. **Contact Form Successful Submission:** Testing the contact form submission with valid data.

5.2. **Close Functionality:** Verifying that the form can be closed properly without submission.

5.3 **Submitting Functionality:** Ensuring the "Send Message" button triggers the form submission correctly.

5.4 **Contact Information Display:** Verifying that the store's contact form (email, name, message) is displayed correctly.

1. **About Us (About The Store)**

6.1 **Content Display:** Verifying that the about us content is displayed correctly.

6.2 **Close Functionality:** Verifying that the about us close probably.

1. **API Testing**

7.1. **User Login API**

7.1.1. **Objective**: Validate the functionality of the user login API by ensuring that it correctly handles authentication requests.

7.1.2. **Test Cases**:

- 7.1.2.1. Test valid user login with correct credentials.

- 7.1.2.2. Test invalid user login with incorrect credentials.

- 7.1.2.3. Test login with missing credentials.

- 7.1.2.4. Test login with special characters in credentials.

- 7.1.2.5. Verify that a valid token is returned upon successful login.

- 7.1.2.6. Ensure the token is stored correctly for future authenticated requests.

- 7.1.2.7. Test token expiry and refresh mechanisms.

7.2. **Product Retrieval API**

7.2.1. **Objective**: Ensure the product retrieval API correctly returns product details.

7.2.2. **Test Cases**:

- 7.2.2.1. Retrieve the list of all products.

- 7.2.2.2. Retrieve details of a specific product by ID.

- 7.2.2.3. Test retrieval with an invalid product ID.

- 7.2.2.4. Verify response structure and data types.

- 7.2.2.5. Test performance and response time under load.

- 7.2.2.6. Check handling of large product datasets.

- 7.2.2.7. Validate the filtering and sorting functionalities.

7.3. **Cart Management API**

7.3.1. **Objective**: Validate the cart management API to ensure it handles cart operations correctly. 7.3.2.

**Test Cases**:

- 7.3.2.1. Add a product to the cart.

- 7.3.2.2. Remove a product from the cart.

- 7.3.2.3. Update product quantity in the cart.

- 7.3.2.4. Retrieve the current state of the cart.

- 7.3.2.5. Test cart operations with invalid product IDs.

- 7.3.2.6. Verify the response after each operation (e.g., correct totals, product list).

- 7.3.2.7. Ensure the cart is persistent across sessions for logged-in users.

7.4. **Order Submission API**

7.4.1. **Objective**: Ensure the order submission API correctly processes and confirms orders.

7.4.2. **Test Cases**:

- 7.4.2.1. Submit an order with valid data.

- 7.4.2.2. Submit an order with invalid data (e.g., missing address, payment information).

- 7.4.2.3. Test order submission with expired or invalid payment details.

- 7.4.2.4. Verify order confirmation and response structure.

- 7.4.2.5. Check order status retrieval after submission.

- 7.4.2.6. Test handling of large orders with multiple products.

- 7.4.2.7. Ensure order history is correctly updated for the user.

7.5. **Profile Update API**

7.5.1. **Objective**: Validate the profile update API to ensure it correctly updates user profile information.

7.5.2. **Test Cases**:

- 7.5.2.1. Update user profile with valid data.

- 7.5.2.2. Update user profile with invalid data (e.g., incorrect email format, missing fields).

- 7.5.2.3. Test updating only specific fields (e.g., changing password without altering other details).

- 7.5.2.4. Verify the response and data integrity after the update.

- 7.5.2.5. Ensure changes are reflected in subsequent profile retrievals.

- 7.5.2.6. Test security aspects, such as unauthorized updates.

7.6. **Search API**

7.6.1. **Objective**: Ensure the search API returns accurate and relevant search results.

7.6.2. **Test Cases**:

- 7.6.2.1. Perform a search with a valid query and verify results.

- 7.6.2.2. Perform a search with an invalid or empty query.

- 7.6.2.3. Test search functionality with special characters and different languages.

- 7.6.2.4. Verify response structure and data types.

- 7.6.2.5. Check performance and response time for search operations.

- 7.6.2.6. Validate relevance and accuracy of search results.

- 7.6.2.7. Ensure that search filters (e.g., price range, categories) work correctly.

**3. Approach**

**3.1 Testing Levels**

1. **Unit Testing**
   * Performed by developers to test individual components and ensure they function correctly.
   * Focus on testing methods and functions within classes.
2. **Integration Testing**
   * Test the interaction between integrated components.
   * Ensure that modules work together as expected.
   * Use of stubs and drivers if necessary.
3. **System Testing**
   * End-to-end testing of the entire application.
   * Verify that the system meets functional and non-functional requirements.
   * Conducted by the QA team.
4. **Acceptance Testing**
   * Performed to ensure the system meets the business requirements.
   * Involves end-users and stakeholders.
   * User Acceptance Testing (UAT) will be conducted before the release.

**3.2 Testing Types**

1. **Functional Testing**
   * Validate that the application functions according to requirements.
   * Includes testing of user authentication, product browsing, cart management, order processing, and security features.
2. **Regression Testing**
   * Ensure that new changes do not negatively impact existing functionality.
   * Automated regression tests will be run after each new build.
3. **Performance Testing**
   * Assess the application's responsiveness, stability, and scalability under load.
   * Includes load testing and stress testing.
4. **Usability Testing**
   * Evaluate the user interface and user experience.
   * Ensure the application is intuitive and user-friendly.
5. **Security Testing**
   * Identify vulnerabilities and ensure the application is secure from threats.

**3.3 Testing Techniques**

1. **Black Box Testing**
   * Test without looking at the internal code structure.
   * Focus on inputs and expected outputs.
2. **White Box Testing**
   * Test with knowledge of the internal code structure.
   * Focus on code coverage, including paths, branches, and conditions.
3. **Exploratory Testing**
   * Unscripted testing to discover defects not covered by existing test cases.
   * Conducted by experienced testers.

**3.4 Test Automation**

Test automation will be used to improve efficiency and coverage:

1. **Selenium WebDriver**
   * Used for automating web application testing.
   * Tests will be written in Python.
2. **Pytest, Unittest**
   * Testing framework for writing and running automated tests.

**3.5 Test Environment**

The test environment will mirror the production environment to ensure accurate testing results:

1. **Hardware**
   * Computers with similar specifications to production servers.
2. **Software**
   * Same operating system, database, and web server as production.
   * Browsers: Chrome, Firefox, Edge (for cross-browser testing).
3. **Test Data**
   * Representative data sets that reflect real-world usage.

**3.6 Entry and Exit Criteria**

**Entry Criteria:**

* Requirements are finalized and approved.
* Test environment is set up and ready.
* Test data is prepared.
* Test cases are reviewed and approved.

**Exit Criteria:**

* All planned test cases are executed.
* All critical and high-severity defects are resolved.
* Test summary report is prepared and reviewed.
* Stakeholders sign off on the testing phase.

**3.7 Risk Management**

Identify potential risks and mitigation strategies:

1. **Risk: Delays in test environment setup.**
   * **Mitigation:** Start setup early, have a backup environment.
2. **Risk: High number of defects.**
   * **Mitigation:** Prioritize critical test cases, perform early and frequent testing.
3. **Risk: Unavailability of key personnel.**
   * **Mitigation:** Cross-train team members, have backup resources

**4. Item Pass/Fail Criteria**

**Pass Criteria**

1. **Functional Requirements Met:**
   * The test item meets all specified functional requirements.
   * All test cases for the feature are executed and pass without any critical or high-severity defects.
2. **Performance Criteria Met**
3. **Usability Criteria Met**
4. **Security Criteria Met**
5. **Compatibility Criteria Met**
6. **No Critical Defects**

**Fail Criteria**

1. **Functional Requirements Not Met:**
   * The test item fails to meet one or more specified functional requirements.
   * Any critical or high-severity defects are identified in the feature.
2. **Performance Criteria Not Met**
3. **Usability Criteria Not Met**
4. **Security Criteria Not Met**
5. **Compatibility Criteria Not Met**
6. **Presence of Critical Defects**

**5. Responsibilities**

**Yosef (Manager)**

1. **Project Oversight and Coordination:**
   * Ensure alignment of QA automation efforts with project goals and timelines.
   * Coordinate with development and testing teams to prioritize tasks.
2. **Resource Management:**
   * Allocate resources effectively, including personnel and tools.
   * Monitor team workload and adjust assignments as necessary.
3. **Risk Management:**
   * Identify project risks related to QA automation.
   * Implement mitigation strategies and contingency plans.
4. **Stakeholder Communication:**
   * Communicate project status, risks, and achievements to stakeholders.
   * Address stakeholder concerns and feedback related to QA activities.

**Tzahi (Manager)**

1. **Quality Assurance Strategy:**
   * Define QA automation strategy aligned with organizational quality goals.
   * Establish standards, processes, and best practices for QA automation.
2. **Technical Leadership:**
   * Provide technical guidance and mentorship to the QA automation team.
   * Ensure adherence to coding standards and automation frameworks.
3. **Tool Selection and Implementation:**
   * Evaluate and select appropriate tools and technologies for QA automation.
   * Oversee the implementation and integration of automation tools.
4. **Continuous Improvement:**
   * Drive continuous improvement initiatives within the QA automation process.
   * Identify areas for efficiency gains and automation enhancement.

**Ehab (QA Automation Engineer)**

1. **Test Automation Development:**
   * Develop and maintain automated test scripts using Selenium WebDriver and Pytest.
   * Ensure automation scripts cover functional and non-functional test scenarios.
2. **Execution and Reporting:**
   * Execute automated test suites and report test results.
   * Investigate and troubleshoot test failures, identifying root causes.
3. **Collaboration:**
   * Collaborate with developers and QA team members to integrate automated tests into CI/CD pipelines.
   * Participate in code reviews and provide feedback on automation code.
4. **Documentation:**
   * Document automation frameworks, test scripts, and procedures.
   * Maintain test case repositories and version control for automation assets.

**Majd (QA Automation Engineer)**

1. **Test Planning and Strategy:**
   * Contribute to test planning and strategy discussions.
   * Identify test scenarios suitable for automation and prioritize them.
2. **Script Maintenance:**
   * Maintain and refactor existing automation scripts to ensure scalability and reliability.
   * Implement improvements based on feedback and changing requirements.
3. **Testing Environment:**
   * Set up and maintain test environments, ensuring they mirror production as closely as possible.
   * Configure test data and ensure data integrity for automated tests.
4. **Training and Support:**
   * Provide training and support to QA team members on automation best practices.
   * Assist in troubleshooting automation issues and provide guidance as needed.

**Rest of the Team**

1. **Manual Testing:**
   * Conduct manual testing activities as required, including exploratory testing and ad-hoc testing.
   * Report defects and collaborate with automation engineers to prioritize and verify fixes.
2. **Test Case Creation:**
   * Create detailed test cases based on functional and non-functional requirements.
   * Execute test cases manually and contribute to automated test case development.
3. **Cross-functional Collaboration:**
   * Collaborate with developers, product managers, and other stakeholders to ensure comprehensive test coverage.
   * Participate in sprint planning, reviews, and retrospectives to improve testing processes.
4. **Quality Advocacy:**
   * Advocate for quality throughout the development lifecycle.
   * Provide feedback on usability, performance, and reliability of the application under test.

**6. Summary**

The Software Test Plan (STP) for the Demoblaze web application outlines a comprehensive strategy for ensuring the quality and reliability of the software. By defining clear objectives, outlining testing approaches, and establishing criteria for success, this document aims to guide the testing efforts effectively. Responsibilities are defined, risks are identified and mitigated, and pass/fail criteria are established to ensure that the application meets functional, performance, usability, security, and compatibility requirements. This STP serves as a roadmap for the QA team and stakeholders, facilitating a structured approach to achieving a high-quality release of the Demoblaze software.

While (time < 10:30):

Print (TaKiNg a bReAk

BAck IN 10:30)

Time = update.time(10:00)