NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

SOFTWARE QUALITY ENGINEERING

FAST UNIVERSITY FOOD ORDERING APPLICATION PROJECT TEST PLAN

Team Members	1. Faaiz Kadiwal (21K-3830) 2. Ammar Asdaque (21K-3923) 3. Anser Tayyab (21K-3909)
Course Instructor	Ms. Syeda Rubab Jaffar
Submission Date	03-December-2023

Table of Contents

1. Purpose

Fast University Food Ordering Application is a real-time application designed to streamline the process of ordering food within the university campus. The application targets the university management, faculty, workers, and students, providing them with a user-friendly platform to order and collect food from various on-campus eating venues. The eating venues include Shwarma Corner, Pizza Fast, Barbeque Dhaba, and University Cafeteria. The application displays menus and rates for each venue, allowing users to place orders efficiently. It also generates estimated collection times based on the order details. Users can choose to input their preferred collection times, and the application supports multiple orders from different venues simultaneously.

The Fast University Food Ordering Application aims to provide a real-time and high-quality food ordering experience within the university campus. The purpose of this test plan is to ensure that the application meets its objectives by conducting comprehensive testing.

2. Features to be tested

2.1 User Authentication Module

User Login Functionality:

- Verify that users can successfully log in with valid credentials.
- Check for appropriate error messages for invalid credentials.

Signup Functionality:

- Verify that new users can successfully create accounts.
- Check for validation of unique usernames and proper handling of errors.

2.2 Navigation Module

Navigation Functionality:

- Test navigation between different modules (customer, admin, manager portals).
- Ensure intuitive navigation and proper redirection.

2.3 Canteen Operations Module

Canteen Operations Functionality:

- Test the ability of administrators to monitor order progress and manage users.
- Verify that managers can efficiently manage canteen operations.

2.4 Ordering Module

Icon Selection and Navigation Functionality:

- Test the selection of icons for different food items.
- Ensure proper navigation and display of selected items.

Ordering Functionality:

- Test the process of placing, managing, and tracking food orders.
- Verify real-time updates on order status and estimated collection times.

Confirmation Functionality:

- Test the confirmation of orders by users.
- Verify that administrators can confirm orders on their end.

2.5 Queue and Linked List Operations Module

Queue and Linked List Operations Functionality:

- Test the functionality of queues and linked lists in managing orders.
- Verify the proper sequence of orders and efficient handling.

2.6 Menu Management Module

Menu Management Functionality:

- Test the ability to add, update, and delete menu items.
- Verify that changes are reflected in real-time.

2.7 View Orders Module

View Orders Functionality:

- Test the view orders feature for customers.
- Verify that administrators and managers can view orders efficiently.

2.8 Sales and Receipts Module

Sales Functionality:

- Test the tracking of sales data.
- Verify accuracy in sales calculations.

Receipts Functionality:

- Test the generation and display of receipts for completed orders.
- Verify the correctness of receipt information.

2.9 View Module

View Functionality:

- Test the overall view module for customers, admins, and managers.
- Verify that relevant information is displayed appropriately

3. Features not to be tested

No features have been explicitly excluded from testing.

The statement "No features have been explicitly excluded from testing" implies that all features within the Fast University Food Ordering Application will undergo testing. Here's a detailed explanation considering real-life assumptions and referencing the quality plan:

1. Comprehensive Testing Philosophy:

- The decision to test all features aligns with a comprehensive testing philosophy, ensuring that every aspect of the application is subjected to scrutiny.

Assumption: This approach is based on the assumption that each feature contributes to the overall functionality, usability, and security of the application.

2. Risk Mitigation:

- Excluding specific features from testing may introduce unforeseen risks, as even seemingly minor components can impact the application's performance or user experience.

Assumption: Risk mitigation strategies, as outlined in the quality plan, consider testing all features to minimize potential issues.

3. User-Centric Focus:

- Users interact with various features, and excluding any could lead to overlooking critical elements that contribute to a positive user experience.

Assumption: User-centric testing is a priority, and excluding features might result in overlooking potential usability issues.

4. Interconnected Dependencies:

- Features in an application are often interconnected. Testing in isolation may not uncover issues that arise from the combined functionality of different features.

Assumption: The application's complexity and interdependencies necessitate testing all features to ensure seamless integration.

5. Regulatory Compliance:

- Certain features may be crucial for regulatory compliance or adherence to specific standards. Omitting testing on these features could lead to non-compliance issues.

Assumption: The application must adhere to regulatory requirements, and all features are subject to validation to meet these standards.

Reference to Quality Plan:

1. Test Coverage Criteria:

- The quality plan likely outlines specific criteria for test coverage, emphasizing the need to cover all features.

Assumption: Test coverage metrics, as per the quality plan, will guide the testing team to ensure a comprehensive approach.

2. Stakeholder Expectations:

- Stakeholders, including end-users, management, and developers, may expect a thorough evaluation of the entire application.

Assumption: Meeting stakeholder expectations, as stated in the quality plan, involves testing all features to ensure a robust and reliable product.

3. Continuous Integration and Deployment (CI/CD):

- The CI/CD pipeline, as mentioned in the quality plan, likely involves the integration of changes across all features.

Assumption: Continuous integration practices imply that every feature is part of the automated testing process to maintain a consistently stable application.

Real-Life Assumptions:

1. Evolution of Requirements:

- Requirements might evolve during the development lifecycle. Features initially considered non-critical could gain importance.

Assumption: Testing all features accommodates potential changes in requirements and ensures adaptability.

2. User Behavior Variability:

- Users may exhibit diverse behaviors, and excluding any feature from testing might overlook scenarios reflecting actual user interactions.

Assumption: Real-life usage patterns are unpredictable, and testing all features accounts for this variability.

3. Security Concerns:

- Security threats can emerge from any part of the application. Omitting testing on specific features may leave vulnerabilities unaddressed.

Assumption: A holistic security testing approach, covering all features, is essential for safeguarding the entire application.

In conclusion, the decision not to exclude any features from testing is a strategic choice to ensure a robust, reliable, and user-friendly Fast University Food Ordering Application. This approach aligns with best practices in software testing and risk management.

4. Approach

The testing approach will involve a combination of manual and automated testing to evaluate the correctness and reliability of each feature. This includes functional testing, usability testing, security testing, and performance testing where applicable.

- Conduct unit testing for individual components.
- Perform integration testing to ensure seamless interaction between modules.
- Conduct user acceptance testing to validate the application against user requirements.

1. Functional Testing:

Description: Functional testing ensures that each function of the Fast University Food Ordering Application operates as designed. This includes testing user interactions, database operations, and system integrations.

Methods:

- **Unit Testing:** Individual components are tested in isolation to validate their correctness.
- **Integration Testing:** Interaction between modules is tested to ensure seamless collaboration.
- **System Testing:** The entire system is tested to verify end-to-end functionality.
- **UI Testing:** The user interface components needs to be properly tested for maintaining user friendly interaction with the application.

Assumption: Test cases cover a range of scenarios, including boundary cases, to thoroughly validate functional requirements.

2. Usability Testing:

Description: Usability testing evaluates the user-friendliness of the application, ensuring that users can easily navigate and interact with the interface.

Methods:

- **User Interface Inspection:** Experts review the interfaces for intuitiveness and consistency.
- **User Feedback Sessions:** Actual users provide feedback on the application's ease of use.

Assumption: Usability testing involves representatives from different user groups, including students, faculty, and canteen staff.

3. Security Testing:

Description: Security testing assesses the robustness of the application against potential vulnerabilities and threats, ensuring data confidentiality and system integrity.

Methods:

- **Penetration Testing:** Attempt to exploit vulnerabilities to identify potential security risks.
- **Code Review:** Analyze the code base for security best practices and potential weaknesses.

Assumption: Security testing involves both automated tools and manual examination to provide a comprehensive security assessment.

4. Performance Testing:

Description: Performance testing evaluates the application's responsiveness, stability, and scalability under varying conditions.

Methods:

- **Load Testing:** Assess the application's performance under expected load conditions.
- **Stress Testing:** Evaluate how the system behaves under extreme load conditions.

Assumption: Performance testing scenarios simulate realistic usage patterns, including peak usage times within the university campus.

Contingency Plan:

If any critical issues are identified during testing, the contingency plan involves immediate triage and prioritization. Critical issues affecting core functionality are addressed with the highest priority, followed by issues related to usability, security, and performance. The project team collaborates to determine the root cause of identified problems, and developers work on resolving issues promptly. Automated tests are rerun, and affected areas undergo manual retesting to ensure the stability of the application. Continuous communication with stakeholders keeps them informed about the testing progress and any adjustments to the release timeline.

5. Suspension Criteria and Resumption Requirements

Testing will be suspended if critical issues affecting core functionality are discovered. Resumption will occur after resolving identified issues and retesting to ensure stability.

Suspension Criteria:

1. Critical Issues Affecting Core Functionality:

- **Test Case Example:** Test cases covering essential functions, such as user login, order placement, and confirmation, are considered critical. If any of these fail, it may be a suspension trigger.

Real-life Assumption: For example, if users cannot log in, place orders, or receive confirmations, the testing process should be suspended. These functionalities are at the core of the application's purpose.

2. Security Vulnerabilities:

- **Test Case Example:** Security testing reveals critical vulnerabilities that could compromise user data or system integrity.

Real-life Assumption: If the application fails security tests, especially in protecting sensitive user information or preventing unauthorized access, testing should be suspended immediately.

3. Performance Issues Impacting User Experience:

- **Test Case Example:** Load testing identifies severe performance issues that significantly affect response times or cause system crashes.

Real-life Assumption: If the application cannot handle the expected load or exhibits unacceptable response times, especially during peak usage, testing should be suspended.

4. Data Corruption or Loss:

- **Test Case Example:** Database testing reveals issues such as data corruption, loss, or inconsistency.

Real-life Assumption: If data integrity is compromised, and there's a risk of losing important information, testing should be suspended until data issues are addressed.

Resumption Requirements:

1. Issue Resolution:

- **Procedure:** When testing is suspended due to critical issues, the development team should promptly address and resolve identified problems.

Verification: After fixing the issues, the affected test cases should be rerun to ensure that the problems have been successfully resolved.

2. Retesting:

- **Procedure:** Once issues are resolved, the testing team should conduct a round of retesting on the affected areas and related functionalities.

Verification: The retest should confirm that the identified issues have been successfully addressed, and the system is stable.

3. Regression Testing:

- **Procedure:** Any changes made to fix issues should not introduce new problems. Therefore, regression testing is crucial to ensure that existing functionalities are not negatively impacted.

Verification: Successful regression testing ensures that fixing one issue hasn't led to the introduction of new defects.

4. Stakeholder Communication:

- **Procedure:** Communicate the suspension and resumption of testing to stakeholders, including project managers, developers, and quality assurance team members.

Verification: Stakeholders should be informed about the reasons for suspension, steps taken for resolution, and the outcome of retesting.

Real-Life Assumptions:

1. Timely Communication:

Assumption: Communication channels between development and testing teams are effective and enable quick dissemination of information regarding critical issues.

2. Collaborative Approach:

Assumption: Development and testing teams work collaboratively to address and resolve issues promptly, with a shared commitment to ensuring software quality.

3. Version Control:

Assumption: A robust version control system is in place to track changes, making it easier to identify modifications made to address issues and ensuring that they don't adversely affect other functionalities.

4. Documentation:

Assumption: Comprehensive documentation of test cases, identified issues, and resolution steps is available, aiding in a smooth resumption process and helping in regression testing.

By adhering to these suspension criteria and resumption requirements, the testing process becomes a dynamic and iterative cycle, contributing to the overall quality and stability of the Fast University Food Ordering Application.

6. Environmental Needs

The testing environment should include:

Hardware:

The application is designed to run on standard computers, and the hardware requirements are not explicitly specified in the provided code. It is assumed that the application is compatible with standard computer hardware.

Networking:

The application requires a stable internet connection for real-time functionality. This indicates that the real-time features of the application, such as order tracking or updates, rely on an active and stable internet connection.

Databases:

The application connects to a Microsoft SQL Server database hosted at asdaque.database.windows.net. The database name is FASTCanteens, and the connection is established using a specific set of credentials (user=asdaque@asdaque;password=Ammar123). This suggests that access to this specific database is necessary for testing purposes.

Software:

- **Web Browsers:** The code doesn't explicitly mention web browsers, indicating that the application is not web-based.
- **Operating Systems:** The code doesn't specify any particular operating system requirements, but since it's a Java application, it should be platform-independent. However, testing on different operating systems is recommended.
- **Other Software Dependencies:** The application uses JavaSwings, JavaFX, and JDBC for GUI development and database interaction. It also uses JOptionPane for displaying dialog boxes. The code assumes the availability of Java Runtime Environment (JRE) for execution.

In summary, the testing environment should include standard computers with a stable internet connection, access to the specified Microsoft SQL Server database, and the necessary software dependencies, including Java Runtime Environment. Additionally, testing on different operating systems can help ensure the application's compatibility across platforms.

7. Schedule

The testing schedule for the Fast University Food Ordering Application is designed to ensure thorough testing of each feature while accommodating the dynamic nature of testing activities. The schedule is divided into distinct phases, each with specific milestones and timelines.

Assumptions:

- **Resource Availability:** It is assumed that testing resources, including personnel and testing environments, are readily available as per the project plan.
- **Stable Development:** The development team follows an agile methodology, delivering features incrementally and providing a stable build for testing at scheduled intervals.
- **Communication Channels:** Effective communication channels exist between development and testing teams for timely issue reporting and resolution.

Testing Phases:

1. Unit Testing Phase:

- **Objective:** Validate the functionality of individual components.
- **Timeline:** 3 weeks

Real-life Assumption: Developers conduct initial unit testing during development, addressing most issues before the dedicated testing phase.

2. Integration Testing Phase:

- **Objective:** Ensure seamless interaction between integrated modules.
- **Timeline:** 2 weeks

Real-life Assumption: Interfaces between modules are well-defined, and integration testing builds upon successful unit testing results.

3. User Acceptance Testing (UAT):

- **Objective:** Validate the application against user requirements.
- **Timeline:** 2 weeks

Real-life Assumption: UAT involves stakeholders, including university staff and students, ensuring that the application aligns with their expectations.

4. Functional and Non-Functional Testing Phase:

- **Objective:** Evaluate functional and non-functional aspects, including performance and security.
- **Timeline:** 2 weeks

Real-life Assumption: Performance testing involves simulating real-world usage scenarios to identify and address any bottlenecks.

5. Regression Testing Phase:

- **Objective:** Ensure that new developments and issue resolutions do not negatively impact existing functionalities.
- **Timeline:** 2 weeks

Real-life Assumption: Regression testing is performed after each development iteration and focuses on areas affected by changes.

6. Deployment Readiness Testing:

- **Objective:** Confirm that the application is ready for deployment.
- **Timeline:** In progress(up to 3 weeks)

Real-life Assumption: This phase includes final checks for database consistency, system stability, and any last-minute adjustments.

Milestones:

1. Completion of Unit Testing:

- **Milestone Date:** 27-09-2023

Criteria: All unit tests pass, and issues identified during unit testing are addressed.

2. Completion of Integration Testing:

- **Milestone Date:** 17-10-2023

Criteria: Integrated modules work seamlessly, and critical issues are resolved.

3. User Acceptance Testing Sign-off:

- **Milestone Date:** 31-10-2023

Criteria: Stakeholders, including university staff and students, provide sign-off, indicating satisfaction with the application.

4. Completion of Functional and Non-Functional Testing:

- **Milestone Date:** 14-11-2023

Criteria: Functionalities meet requirements, and non-functional aspects like security and performance are within acceptable limits.

5. Completion of Regression Testing:

- **Milestone Date:** 28-11-2023

Criteria: Existing functionalities are unaffected by recent developments, and any regressions are addressed.

6. Deployment Readiness Confirmed:

- **Milestone Date:** 14-12-2023

Criteria: Final checks confirm that the application is ready for deployment, and all stakeholders are informed.

Test Step	Duration (in days)	Start Date	End Date
Test Planning	8 days	08/08/2023	16/08/2023
Build Test Plan	6 day	08/08/2023	14/08/2023
Define the Acceptance Criteria	1 day	14/08/2023	15/08/2023
Define the Environment	1 day	15/08/2023	16/08/2023

Test Suites	8 days	16/08/2023	24/08/2023
Develop test data	4 day	16/08/2023	20/08/2023
Develop test cases	4 days	20/08/2023	24/08/2023
Design + Development	33 days	24/08/2023	27/09/2023
Development + Test Execution	100 days	27/09/2023	16/12/2023
Setup and Testing	98 days	27/09/2023	14/12/2023
Evaluation	2 day	14/12/2023	16/12/2023
Test Log	1 day	16/12/2023	17/12/2023
Mention pass/fail test cases	0.5 day	16/12/2023	16/12/2023
Determine Efficiency	0.5 day	16/12/2023	17/12/2023

Contingency Plan:

In the event of unforeseen delays or critical issues discovered during testing, a contingency plan involves reassessing timelines, allocating additional resources if necessary, and adjusting the project plan accordingly. Regular communication between the testing and development teams is crucial for effective issue resolution and timely progress.

8. Acceptance Criteria

The software is considered ready for release if it meets the following criteria:

- Stakeholder sign-off and consensus.
- Tested under specified environments.
- Minimum defect counts at various priority and severity levels.
- Minimum test coverage numbers.

1. Stakeholder Sign-off and Consensus:

Definition: All key stakeholders, including university management, faculty, workers, and students, provide formal sign-off and consensus that the Fast University Food Ordering Application meets their requirements and expectations.

Verification:

- Formal sign-off documents from representatives of each stakeholder group.
- Recorded minutes of meetings indicating consensus.

2. Tested Under Specified Environments:

Definition: The application is thoroughly tested in the specified environments, including hardware, networking, databases, and software, to ensure compatibility and reliability.

Verification:

- Test reports confirming successful execution of test cases in different environments.
- Logs indicating successful testing on standard computers, stable internet connections, and relevant databases.
- Compatibility testing results with various web browsers and operating systems.

3. Minimum Defect Counts at Various Priority and Severity Levels:

Definition: The application undergoes rigorous testing, resulting in minimal defects categorized by priority and severity.

Verification:

- Defect tracking system reports showing resolved and outstanding defects.
- Severity levels for any remaining defects are within acceptable thresholds.

- Priority levels for any remaining defects are aligned with project priorities.

4. Minimum Test Coverage Numbers:

Definition: The testing process achieves comprehensive coverage of functional and non-functional aspects, including unit testing, integration testing, user acceptance testing, and performance **testing.**

Verification:

- Test coverage reports indicating the percentage of code covered by test cases.
- Traceability matrices linking test cases to requirements to ensure comprehensive coverage.
- Evidence of performance testing covering various scenarios and usage patterns.

Assumptions:

1. Effective Test Case Design:

Assumption: Test cases are designed to cover various scenarios, including both typical and edge cases, to ensure a thorough evaluation of the application.

2. Active Stakeholder Engagement:

Assumption: Stakeholders actively participate in the testing process, providing timely feedback and clarifications to enhance the quality of testing.

3. Continuous Monitoring of Defect Resolution:

Assumption: The defect resolution process involves continuous monitoring and retesting to confirm that resolved issues do not introduce new problems.

4. Defined Severity and Priority Criteria:

Assumption: The project team has clear criteria for assigning severity and priority levels to defects, ensuring consistent evaluation and prioritization.

5. Test Coverage Metrics Definition:

Assumption: The project team has established metrics and benchmarks for acceptable test coverage, considering the complexity and criticality of different modules.

Expected Outcomes

Upon completion of testing, the following outcomes are expected:

- Efficient food ordering system.
- Real-time updates on order status.
- User-friendly interfaces.
- Administrative management tools.
- Inventory control features.

Contingency Plan:

In the event that the acceptance criteria are not fully met, a contingency plan involves reassessing the remaining tasks, retesting the impacted areas, and obtaining stakeholder feedback for prioritizing necessary adjustments. The project team collaborates to address any outstanding issues promptly, ensuring that the software achieves the required quality standards before release. Regular communication channels are essential to keep stakeholders informed of the progress and any deviations from the original plan.

9. Roles and Responsibilities

1. **Quality Assurance Manager (Faaiz Kadiwal):**

   Oversight of overall quality control process.

2. **Project Manager (Syed Ammar Asdaque):**

   Coordination of quality efforts within the project.

3. **Developers (Faaiz, Ammar, Anser):**

   Adherence to coding and quality standards.

4. **QA Team Members (Faaiz, Ammar, Anser):**

   Execution of testing and quality assurance tasks.

Conclusion

This test plan provides a comprehensive outline for testing the Fast University Food Ordering Application. It ensures that each feature is thoroughly tested to meet the application's objectives. The collaboration between development and testing teams, along with the involvement of key stakeholders, will contribute to the success of the application.