PINJAM SULTAN APP
TEST PLAN

Created By : Ananda Thalia - QA Engineer
Created Date : December 5th 2025

Document History

Version	Date	Author	Description of Change
V1	December 5th 2025	Ananda Thalia	Initial draft created.
V2	December 6th 2025	Ananda Thalia	Finalized document

Table of Contents

Objective

This test plan aims to ensure the quality, functionality, and reliability of the Pinjam Sultan App in alignment with the Product Requirements Document (PRD). It outlines the testing approach, methodology, and scope to ensure all core and supporting features meet the required standards of usability, performance, security, and compatibility.

Scope

Features to be Tested

The scope of this test plan includes the following features:

1. Product Catalog
   Ensure accurate display and accessibility of the product catalog.
2. Product Information
   Verify correctness and completeness of product details.
3. Rental Shop Information
   Confirm the accuracy and availability of rental shop data.
4. User Registration
   Validate smooth and secure user registration processes.
5. Product Booking
   Test functionality and reliability of the product booking system.
6. Task Terms and Conditions
   Ensure users can access, understand, and accept relevant terms and conditions.
7. Chat
   Validate real-time messaging between users and rental shops, including notifications.
8. Product Categorization
   Verify the correct grouping of products into predefined categories for better navigation.
9. Product Filter
   Test the accuracy and usability of filtering products based on criteria such as category, price, or popularity.

Testing Types

Testing will cover the following aspects for each feature:

1. Functional Testing :
   - Verify the functionality of all features in the app.
   - Test various scenarios for feature creation, modification, and deletion to ensure proper functionality.

2. Data Validation Testing:
   - Ensure that input forms correctly validate data, rejecting invalid or incomplete inputs.
   - Test boundary values for input fields to identify unexpected behaviors.
   - Validate the accuracy and consistency of data returned in responses.

3. Integration Testing:
    - Verify the interactions between various app modules and external services.
    - Ensure data consistency across related features and modules.

4. Error Handling Testing:
    - Verify that appropriate error codes and messages are returned for invalid requests.
    - Ensure error responses do not disclose sensitive information.

5. Concurrency Testing:
    - Assess the app's behavior when multiple users perform actions like booking or modifying data simultaneously.

6. Compatibility Testing:
    - Test the app across multiple devices and operating systems to ensure cross-compatibility.

7. Usability Testing:
    - Evaluate the app's user interface and overall user-friendliness from an end-user perspective.
    - Ensure intuitive navigation and accessibility for users.

8. Documentation Review:
    - Review the clarity, completeness, and accuracy of the Product Requirements Document (PRD).
    - Verify alignment between the PRD and the actual behavior of the app.

9. Automation Testing:
    - Develop automated test scripts for features that are identified as suitable for automation.
    - Run automated tests to cover repetitive test cases, especially regression tests, to ensure faster and more reliable execution.

10. Regression Testing:
    - Perform regression testing after bug fixes, updates, or new feature additions to ensure existing functionality remains unaffected.
    - Automated tests will be used for repetitive tasks such as regression testing to increase efficiency and consistency.

11. Performance Testing:
    - Measure the app's response time under normal and peak loads to identify potential bottlenecks.
    - Evaluate the app's throughput and scalability for handling concurrent user requests.
    - Monitor performance metrics in real-time to ensure stability and reliability during high traffic.

12. Security Testing:
    - Scan basic vulnerabilities, including input handling issues that may lead to injection attacks such as SQL injection or XSS.
    - Validate HTTPS implementation and ensure secure data transmission practices by analyzing traffic between the client and server.
    - Check session management settings such as cookie security flags.
    - Focus on identifying insecure redirects, improper error handling, or exposed sensitive data in API responses.

Schedule

The testing process is estimated to take between 19 and 26 working days, involving one QA Lead/Manager and three QA Engineers. Below is the tentative schedule for the testing activities of the Pinjam Sultan App:

STLC Phase	Activity	Duration	Target Date	Responsibility
1. Test Planning	- Finalize the test plan, outlining the scope, objectives, and strategies for all features. - Define testing types, tools, resources, and deliverables. - Define automation strategy and identify features suitable for automation.	1-3 days	DD/MM/YYYY - DD/MM/YYYY	QA Lead or QA Manager
2. Test Design	- Create detailed test cases, test scenarios, and test scripts for each feature (manual & automated). - Prepare and verify test data for execution. - Develop and finalize automation scripts for eligible test cases.	7-9 days	DD/MM/YYYY - DD/MM/YYYY	QA Engineers
3. Test Execution	- Execute functional, security, performance, and other tests (manual & automated).	10-12 days	DD/MM/YYYY - DD/MM/YYYY	QA Engineers

	- Run automated test cases and log defects, track progress. - Report defects and monitor defect life cycle (log, track, retest). - Conduct regression testing on fixed defects (manual & automated).			
4. Test Closure	- Perform final checks and deliver a comprehensive test summary report. - Conduct a final round of regression testing with automation tests.	1-2 days	DD/MM/YYYY - DD/MM/YYYY	QA Lead & QA Engineers

Test Environment

The testing activities for the Pinjam Sultan App will be conducted in the test environment are as follows:

1. Hardware Configuration
   Devices:
   - Android smartphones (Android versions 10 and 15)
   - iOS smartphone (iOS versions 14 and 18)

2. Network Configuration
   Test Scenarios:
   - High-speed Wi-Fi connection.
   - Mobile data simulation for 4G networks.
   - Low-speed and unstable network conditions to test app performance and error handling.

3. Environment Setup
   - Development Environment:
     - Used for initial functional testing and smoke tests.
     - Includes test builds directly from the development team with debugging enabled.
   - Staging Environment:
     - Mirrors the production environment with mock data for UAT (User Acceptance Testing) and regression testing before deployment to production.

4. User Accounts
   Test Users:
   A set of test accounts with predefined roles:
   - Admin accounts for testing privileged features.
   - Regular users account for general app functionality.
   - New user accounts to test the onboarding flow.

   Accounts representing various scenarios:
   - Accounts with pending payments.
   - Accounts without any bookings.

5. Tools and Platforms
   - Bug Tracking Tools: Jira.
   - Test Management Tools: AIO Test.
   - Automation Tools: Appium.
   - Performance Testing Tools: Gatling.
   - Security Testing Tools: Burp Suite.

Test Data

The test data for the Pinjam Sultan App will be designed to ensure comprehensive test coverage, including functional, performance, and security testing. The following strategies will be employed:

1. Manual Test Data Creation
   - Valid Data:
     Manually created to represent typical user inputs that align with the app's requirements.
   - Invalid Data:
     Created to test input validation and error handling, covering edge cases such as boundary values, unsupported formats, and empty fields.

2. Real-World Data Usage
   Real-world data may be used for specific scenarios where realistic inputs are critical, such as financial or location-based testing.

3. Test Data Management
   Data Storage:
   All test data will be securely stored in a designated test database or file system, separate from production environments.

Test Strategy

The testing strategy for the Pinjam Sultan App is designed to ensure comprehensive coverage of all core and supporting features while maintaining efficiency and adaptability.

Step 1: Test Design

- Create detailed test scenarios and test cases using:
    - Equivalence Partitioning
    - Boundary Value Analysis
    - Decision Table Testing
    - Use Case Testing
- Prioritize test cases based on feature criticality and risk assessment.

Step 2: Execution Flow

- Smoke Testing: Conducted on each build to verify basic functionality. Builds that fail smoke testing will be rejected.
- In-depth Testing: Performed on stable builds, including functional, regression, usability, and exploratory testing.
- Defect Reporting: Bugs will be logged in Jira and addressed based on priority.
- Test Cycles: Repeated until all critical issues are resolved and the app meets quality standards.

Step 3: Best Practices

- Shift-Left Testing: Engage in testing early during development to identify and address issues promptly.
- Context-Driven Testing: Tailor the testing approach based on the application's specific context.
- End-to-End Flow Testing: Simulate real user scenarios to validate complete workflows.
- Exploratory Testing: Utilize tester expertise to uncover potential issues outside predefined test cases.

Defect Management

To ensure efficient and systematic handling of defects during the testing process for the Pinjam Sultan App, the following practices and processes will be followed:

1. Defect Identification Criteria
   A defect will be logged when any of the following deviations are observed:
    - Requirement Deviation: Functionality does not meet the specified requirements outlined in the PRD.
    - User Experience Issues: Inconsistent or suboptimal user interface, navigation, or usability.
    - Technical Errors: Application crashes, performance bottlenecks, or backend issues.
    - Integration Failures: Incompatibilities between API endpoints or system modules.

2. Defect Reporting Steps
    - Use a standard defect reporting template to maintain consistency.
    - Provide clear and detailed reproduction steps.
    - Attach supporting evidence such as screenshots, videos, or log files.
    - Categorize defects by module or feature to simplify tracking.

3. Defect Triage and Prioritization
    - Assign severity levels:
        - Critical: Blocks essential functionality or causes system crashes.
        - High: Affects key functionalities but has workarounds.
        - Medium: Impacts non-critical functionalities or features.
        - Low: Minor issues, such as cosmetic or spelling errors.
    - Assign priority levels based on the defect's business impact and urgency.
    - Defects will be reviewed and assigned to the responsible team for investigation and resolution.

4. Tools for Defect Management
    - Bug Tracking Tool: Jira will be used to log, track, and manage defects.
    - Test Management Tool: AIO Test for test case management and integration with defect tracking.

5. Roles and Responsibilities
    - Testers: Identify and report defects, providing detailed information.
    - Developers: Investigate and fix assigned defects.
    - QA Lead/Manager: Review reported defects, oversee prioritization, and track progress.
    - Stakeholders: Receive regular updates on defect status and resolution progress.

6. Communication Protocols
    - Channels: Use Slack for immediate updates and discussions.
    - Frequency: Daily updates during defect triage meetings and weekly progress reports for stakeholders.

7. Defect Metrics for Evaluation
    - Defect Detection Rate (DDR): Number of defects found per testing cycle.
    - Defect Resolution Time: Average time taken to fix defects.
    - Defect Leakage Rate: Percentage of defects found after release.
    - Defect Rejection Rate: Percentage of defects marked as invalid or duplicate.
    - Severity Distribution: Breakdown of defects by severity level.

8. Defect Handling Points of Contact (POC)

Module/Component	Point of Contact (POC)
Front-End	Mr. X
Back-End	Mr. Y
DevOps	Mr. Z

Risk Management

Risk	Mitigation Strategy
Outdated or incomplete documentation of application features and APIs	Collaborate with stakeholders to obtain accurate and up-to-date documentation.
Unclear or insufficient error messages, making defect analysis challenging	Implement extensive negative testing to validate error-handling scenarios.
Non-availability of critical team members or resources	Maintain a backup resource plan and cross-train team members to ensure continuity.
Dependency on external APIs or services causing unpredictable test results	Use mock APIs or virtualized services to simulate the behavior of external dependencies.
Frequent changes or updates in application features introducing regression issues	Perform comprehensive regression testing after every update or release.
Limited time for testing due to tight deadlines	Prioritize high-risk and critical functionalities first.
Insufficient communication or collaboration between teams and stakeholders	Establish clear and open communication channels using Slack.
Poorly defined test data leading to inconsistent test results	Create a standardized test data set covering all scenarios, including edge cases.
Inadequate infrastructure or tools leading to delays in testing activities	Ensure all necessary testing tools (e.g., Appium, Jira, Burp Suite) are set up and accessible before the testing phase begins.
High defect leakage rate due to insufficient testing scope	Regularly review and update the test cases to cover all new features and scenarios. Also, Conduct peer reviews of test cases and execution results to ensure thorough coverage

Test Deliverables

The following deliverables will be provided to the client throughout the testing process:

STLC Phase	Activity	Completion Date
1. Test Planning	- Finalize the test plan, outlining the scope, objectives, and strategies for all features. - Define testing types, tools, resources, and deliverables. - Define automation strategy and identify features suitable for automation.	
2. Test Design	- Create detailed test cases, test scenarios, and test scripts for each feature (manual & automated). - Prepare and verify test data for execution. - Develop and finalize automation scripts for eligible test cases.	
3. Test Execution	- Execute functional, security, performance, and other tests (manual & automated). - Run automated test cases and log defects, track progress. - Report defects and monitor defect life cycle (log, track, retest). - Conduct regression testing on fixed defects (manual & automated).	
4. Test Closure	- Perform final checks and deliver a comprehensive test summary report. - Conduct a final round of regression testing with automation tests.	

Approvers List

No	Name	Role	Approver Review	Date