**~ Test Plan ~**

**Triple Seven Restaurant Management System**

1. **Introduction**

This test plan describes a detailed strategy to test the Restaurant Management System (RMS) designed for the Triple Seven restaurant. The system is intended to streamline restaurant operations, such as table assignments, order processing, customer interactions, and easy integration with external systems. This plan outlines the testing technique, objectives, environment requirements, schedule, deliverables, and exit criteria to ensure that the system operates effectively and meets all business needs.

1. **Testing Strategy**

This test plan includes several testing methodologies to guarantee the system performs as required:

* **Unit Testing:** Individual software units (classes, modules) will be tested for functionality, validity of data, and error handling using a unit testing framework such as JUnit or PHPUnit.
* **Integration Testing:** Modules will be integrated and tested to ensure interaction, data flow, and communication between components.
* **System Testing:** The entire system will be tested by replicating real-world scenarios, including order processing, table management, customer interactions, and integration with external systems (e.g., payment gateway).
* **Regression Testing:** To verify that new additions do not interfere with existing functionality, previously passed tests will be re-executed after code changes.
* **Usability Testing:** Real users (restaurant staff) will evaluate the system's ease of use, intuitiveness, and workflow efficiency.

1. **Testing Objectives**

* Ensure all system features operate as per requirements.
* Ensure the system is user-friendly and efficient for employees at restaurants.
* Evaluate system performance during peak hours and response times.
* Test system security protocols to prevent unwanted access and data breaches.
* Verify system compatibility with different operating systems and hardware configurations.
* Ensure seamless data sharing with external systems, such as payment channels.

1. **Hardware and Environment Requirements**

* Operating System: Windows 10 (minimum) or equivalent Linux distribution
* Hardware: Processor: Intel i5 or equivalent, RAM: 8GB minimum, Storage: 250GB SSD
* Database: MySQL, PostgreSQL, or equivalent
* Development Tools: IDE (Integrated Development Environment) like IntelliJ IDEA or Eclipse, Unit Testing Framework (e.g., JUnit, PHPUnit)

1. **Test Schedule**

The testing process will be divided into phases with defined timelines:

**Phase 1: Unit Testing (1 week)**

* Developers create unit tests for every code module.
* Unit tests are executed and reviewed.

**Phase 2: Integration Testing (1 week)**

* Integrated modules are checked for interactivity and data flow.
* Defects are found and corrected.

**Phase 3: System Testing (2 weeks)**

* System-wide functionality is tested using pre-defined test cases.
* User Acceptance Testing (UAT) is conducted with restaurant staff.
* Performance and stress testing mimic peak workloads.

**Phase 4: Regression Testing (ongoing)**

* Re-running previously passed test cases following code changes.
* Ensures new features. Do not add regressions.

1. **Staffing and Training Needs**

* **Test Lead/Manager:** Responsible for overall test strategy, execution, and reporting.
* **Software Testers:** Individuals with experience in restaurant management systems and software testing methodologies.
* **Restaurant Staff:** Participation in User Acceptance Testing (UAT) to evaluate system usability and workflow efficiency.

Training will be provided to all staff involved in testing. Testers will receive training on the specific functionalities of the RMS and the chosen testing methodologies. The restaurant staff will be trained in basic system navigation and functionalities relevant to their roles.

1. **Test Deliverables**

The following deliverables will be produced during the testing process:

* **Test Plan:** This document outlines the overall testing strategy.
* **Test Cases:** Detailed documents specifying test scenarios, expected results, and pass/fail criteria.
* **Test Data:** Data sets used for testing various functionalities.
* **Defect Tracking Logs:** Records of identified defects, their severity, and resolution status.
* **Test Reports:** Documents summarizing test execution results, identified defects, and overall system quality assessment.
* **User Acceptance Testing (UAT) Report:** Report capturing user feedback and any identified issues from restaurant staff testing.

1. **Exit Criteria**

The testing process will be considered complete when the following criteria are met:

* All functionalities mentioned in the system specifications were thoroughly tested.
* A set percentage of test cases (e.g., 90%) passed successfully.
* Identified errors are corrected and tested again for resolution.
* User Acceptance Testing (UAT) is accomplished with minimum or no critical issues reported by restaurant employees.
* Performance testing shows that the system can manage peak loads appropriately.

1. **Review and Approvals**

This test plan will be reviewed and approved by the following stakeholders:

* Project Manager.
* Development Lead.
* Restaurant General Manager.

Following approval, the test plan will serve as a guide for the system testing procedure. Any changes or deviations shall be recorded and informed to all relevant parties.

1. **Test Automation**

Automating **test execution** is important for efficient regression testing and ensuring system quality. Here are some methodologies and techniques to consider:

**- Test Automation Frameworks**

* Selenium: Open-source framework for web application UI automation.
* Appium: Cross-platform framework for automating mobile and web applications.
* JUnit/PHPUnit: Unit testing frameworks with built-in automation capabilities.

**- Test Automation Techniques**

* **Record and playback:** Record user activities during manual testing and repeat them back for test automation.
* **Data-Driven testing:** Use external data sources (e.g., spreadsheets) to present test data for different scenarios.
* **Keyword-Driven testing:** Using keywords to identify test stages makes it easier to maintain test scripts.
* **Behavior-Driven development (BDD):** Focuses on defining system behavior from the user's perspective, allowing automated tests based on those behaviors.

1. **Test Cases**

* **Test Case 1: Booking Table.**

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **Test Objective:** | **Pass/Fail Criteria:** |
| TC001 | To verify that a customer can successfully book a table. | -Pass: A booking is added to the customer's bookings list, and the booking is successfully made.  -Fail: The booking is not added to the customer's bookings list, or the booking fails to be made. |

**Preconditions:**

* An instance of Customer is created.
* A Table object and a DateTime object are available.

**Test Steps:**

1. Create an instance of Customer.
2. Create a Table object.
3. Create a DateTime object.
4. Call the Book(Table table, DateTime time) method of the Customer object with the table and time parameters.

**Expected Results:**

* + A new booking is added to the customer's bookings list.
  + The booking is successfully made.

**Result:** Passed.

**Screenshot** of the Code:

A screenshot of a computer

Description automatically generated

* **Test Case 2: Test Adding and Removing Items**

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **Test Objective:** | **Pass/Fail Criteria:** |
| TC002 | To verify that items can be added & removed from the order. | -Pass: Items are successfully added and removed from the order.  -Fail: Items are not added or removed correctly. |

**Test Steps:**

* + 1. Create an instance of **Order**.
    2. Create one or more **MenuItem** objects.
    3. Add items to the order using the **addItem(MenuItem item)** method.
    4. Remove items from the order using the **removeItem(MenuItem item)** method.

**Expected Results:**

* Items are successfully added and removed from the order.

**Result:** Passed.

A screenshot of a computer program

Description automatically generated**Screenshot for the code:**

* **Test Case 3: Test Generating Report**

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **Test Objective:** | **Pass/Fail Criteria:** |
| TC003 | To verify that the **generateReport()** method executes correctly. | -Pass: The method executes without throwing any exceptions and the expected message is printed.  -Fail: The method throws an exception or does not print the expected message. |

**Test Steps:**

1. Create an instance of **Manager**.
2. Call the **generateReport()** method of the **Manager** object.

**Expected Results:**

* The method executes without throwing any exceptions.
* The manager's name is printed along with the message indicating that a report is being generated.

**Result:** Passed.

**Screenshots:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

* **Test Case 4: Test Adding and Removing Observers**

|  |  |  |
| --- | --- | --- |
| **Test Case ID:** | **Test Objective:** | **Pass/Fail Criteria:** |
| TC004 | To verify that observers can be added and removed from the booking. | -Pass: Observers are successfully added and removed from the booking.  -Fail: Observers are not added or removed correctly. |

**Test Steps:**

* + 1. Create an instance of **Booking**.
    2. Create one or more **BookingObserver** objects.
    3. Add observers to the booking using the **addObserver(BookingObserver observer)** method.
    4. Remove observers from the booking using the **removeObserver(BookingObserver observer)** method.

**Expected Results:**

* Observers are successfully added and removed from the booking.

**Result:** Passed.

**Screenshot:**

A screenshot of a computer

Description automatically generated

1. **Conclusion**

This test plan outlines a comprehensive method to ensuring our Restaurant Management System's quality, functionality, and usability. By integrating diverse testing methodologies, automation tools, and a well-defined testing schedule, we can create a resilient system that fits the needs of Triple Seven restaurant and enables efficient operations.