Testing Plan

Yağmur Sabırlı 21070006017
Melike Aytaç 21070006056
Beren Elçin Polat 22070006064
Sultan Esen Murat 21070006039

Table of Contents

1 Overview

We will conduct a series of different tests to fully exercise the "The Restaurant Automation System". These tests will validate core modules such as Order Management, Inventory Control, and Report Generation.

Outlined below are the main test types that will be performed:

1.1 Unit Test

- Done by each developer.
- Examples: `OrderManager`, `Inventory`, `getTotalPrice()` in `Order`
- Purpose:
- Ensure that methods return correct outputs for valid/invalid input
- Validate internal logic such as inventory deduction

1.2 Integration Test

- Performed once the GUI and backend are ready
    - Examples:
    - `OrderScreen` ↔ `OrderManager`
    - `InventoryScreen` ↔ `Inventory`
    - Purpose:
    - Check that subsystems work smoothly together
    - Detect interface/data flow bugs

1.3 System Test

- Run after full system is assembled
- Involves simulating realistic usage scenarios
- Ensures the system functions as per requirements

1.3.1 Functional Test

- Verify that users can place orders, view reports, and manage inventory

1.3.2 Performance Test

- Test UI response time, database query latency
- Example: Placing 100 orders in rapid succession

1.3.3 Security Test

- Test login functionality
- Ensure only admins can access reports or inventory updates

1.3.4 Recovery Test

- Simulate system crash during order placement
- Check whether data persists and restores correctly

1.4 User Acceptance Test

Once the system is ready for implementation, user representatives will perform User Acceptance Testing. The purpose of these tests is to confirm that the system is developed according to the specified user requirements and is ready for operational use.

2 High-Level Milestones

Four major milestones are listed here:

During **Iteration 1**, we will primarily focus on *Unit Testing*. Each core backend class—such as `Order`, `Inventory`, and `OrderManager`—will be tested independently using JUnit. We aim to complete this phase by **April 15, 2025**, ensuring that individual methods behave as expected across valid and invalid input ranges.

In **Iteration 2**, the focus will shift to *Integration Testing*. At this stage, we will connect the user interface with the backend logic, testing the interactions between modules like `OrderScreen` and `OrderManager`, or `InventoryScreen` and `Inventory`. This will help verify that the system components communicate correctly and data flows seamlessly. The integration testing phase is planned to be completed by **May 11, 2025**.

Once all components are integrated, we will proceed with *System Testing* during **Iteration 3**, which is expected to be finalized by **May 26, 2025**. This testing phase will validate the system as a whole under realistic usage scenarios. We will check for bugs, inconsistencies, and performance issues to confirm that the system behaves according to the functional and non-functional requirements.

Finally, in the **last week before project delivery**, we will conduct *User Acceptance Testing (UAT)*, targeted for completion by **June 4, 2025**. During UAT, designated team members will simulate real-world user roles to evaluate if the application fulfills end-user expectations. This will serve as the final verification phase before deployment.

3 Test Staffing

The participants for testing will be:

- Unit Testing: All Developers
- Integration Testing: Developers + Peer Review
- System Testing: QA responsibility shared by Melike & Esen
- UAT: All team members participate as testers

All developers are responsible for:

- writing the detailed plan for the test (if necessary)
- supervising the execution of the test
- providing resources for the test
- documenting the results of the test
- updating the plan because of the test

4 Test Process

4.1 Test Tasks

4.1.1 Front End

- LoginScreen → Auth flow
- OrderScreen → Order placement
- InventoryScreen → View/update stocks
- ReportScreen → Generate reports
- Error message validation

4.1.2 Back End Data Storage

- Inventory class → `checkAvailability()`, `deductForOrder()`
- OrderManager → `createOrder()`, `getOrdersByDate()`
- ReportGenerator → `generateDailyReport()`

4.2 Test Process Approach

Phase	Responsible	Description
Unit Testing	Melike Aytaç, Sultan Esen Murat	Each class and method will be tested individually using JUnit.
Integration Testing	Yağmur Sabırlı, Beren Elçin Polat	Connections between GUI and backend (e.g., OrderScreen ↔ OrderManager) will be tested.
System Testing	Melike Aytaç, Sultan Esen Murat, Yağmur Sabırlı	Entire application will be tested under user scenarios to ensure full functionality.
User Acceptance Testing	All team members	Final testing performed from an end-user perspective to validate the system.

5 Testing Methods

Special programs may be written to supply the appropriate environment and inputs to the unit being tested, especially during unit test phase. We plan to use Black-box technique as the primary testing approach, and White-box technique as a secondary testing approach.

A wide range of inputs will be entered to determine that outputs are correct. Here, we plan to use Boundary Test technique to test error handling. Start with known good and valid values, then try expected bad values, move through reasonable and predictable mistakes, at last, try extreme errors and crazy inputs in order to catch problems that might affect the system's functioning.

We will also have intensive testing of the Front-End fields and screens. We will make sure that Java GUI Standards, and screen & field look and feel maintain consistency with the rest of the application. We will check for valid, invalid and limit data input.

6 Measurements

We plan to track the following measurements:

- System's response time
- Mean time between two failures
- Total number of defects
- Severity of each defect
- Mean time required to find a defect

7 Testing Risks

7.1 Requirements

The test plan and test schedule are based on the current Requirements Document. Any changes to the requirements could affect the test schedule.

7.2 Time

The schedule for each phase is very aggressive and could affect testing. A delay in one of the other phases could result in a subsequent delay in the test phase. Project management support is required to reduce the risk and meet the forecasted completion date.

7.3 Environment

Missing/unstable local MySQL setup could hinder integration testing

7.4 Personnel

All developers should be familiar with the testing tools. Unexpected turnovers can impact the schedule.

All developers will also need the following resources available: developers and test users.

8 Software Tools

- **JUnit** for unit testing
- **GitHub Issues** for defect tracking
- **MySQL Workbench** for DB inspection
- **IntelliJ IDEA / VSCode** for test execution
- **JIRA** (optionally) for managing tasks

9 Testing Support

- **Hardware:** Team members' laptops with minimum 8 GB RAM

- **Database:** Local MySQL Server with test dataset
- **GitHub Repo:** Up-to-date with all development branches
- **Sample Data:** Pre-filled DB records for orders, menu, inventory

10 Personnel Support

- **Instructor:** Prof. Dr. Mehmet Süleyman Ünlütürk
- **Team review:** Peer testing for unbiased feedback
- **Mock users:** Teammates simulate waiter/manager actions