Software Testing Plan

Project: Real-Time Queue & Appointment Optimizer System

1. Introduction

This Software Testing Plan outlines the proposed testing approach for the Real-Time Queue & Appointment Optimizer project. Currently, the project is in the planning and design phase, and software development has not yet begun. Therefore, this document focuses on defining the testing activities that will be carried out once the implementation phase starts.

The main objective of this testing plan is to ensure that the system meets all specified requirements, functions correctly under different conditions, and provides a reliable and user-friendly experience for end users.

2. Test Strategy

The following testing strategies will be adopted during the development lifecycle:
Unit Testing:
Each individual module such as queue management, counter service, and database operations will be tested separately to verify that they work as expected.
Integration Testing:
After unit testing, modules will be combined and tested together to ensure proper interaction between APIs, database, and real-time communication components.
System Testing:
The complete system will be tested as a whole to verify real-time queue updates, token assignment, counter handling, and overall system behavior.
Acceptance Testing:
The system will be tested against user requirements to confirm that it meets expected functionality and usability standards.

3. Test Environment

The testing environment will consist of the following resources:
Hardware Requirements:
• Desktop or Laptop Computer
• Minimum 8 GB RAM

Software Requirements:
• Operating System: Windows or Linux
• Backend Framework: FastAPI (Python)
• Database: MySQL
• Web Browser: Google Chrome

Test Data:
• Sample usernames
• Sample counters
• Simulated queue entries

4. Test Cases

The following table lists the detailed test cases planned for the Real-Time Queue & Appointment Optimizer. These test cases will be executed once development begins. Each test case includes clear inputs, expected outputs, and pass/fail criteria.

Test Case ID	Test Scenario	Test Description	Input Data	Expected Output	Actual Output	Status
TC-01	Queue Registration	Verify user can join queue	Username	Token generated	To be tested	Pending
TC-02	Multiple Users	Verify multiple users can join	Multiple names	Queue order maintained	To be tested	Pending
TC-03	Auto Token Assignment	Assign next token to free counter	Counter free	Next token assigned	To be tested	Pending
TC-04	Real-Time Update	Verify live updates	Queue change	Display updates instantly	To be tested	Pending
TC-05	Counter Heartbeat	Detect active counter	Heartbeat signal	Counter remains active	To be tested	Pending
TC-06	Counter Failure	Handle counter disconnect	No heartbeat	Token reassigned	To be tested	Pending
TC-07	Database Storage	Store token in DB	New token	Data saved in MySQL	To be tested	Pending

TC-08	System Restart	Recover queue after restart	Server restart	Queue restored	To be tested	Pending
TC-09	Invalid Input	Handle empty input	Blank name	Error message shown	To be tested	Pending
TC-10	System Load	Handle multiple users	Simultaneous joins	Stable performance	To be tested	Pending

Test cases will cover functionalities such as queue joining, automatic token assignment, real-time updates, counter failure handling, and database storage.

5. Defect Management

Defect management is the process of identifying, tracking, and resolving issues found during software testing. It ensures that problems in the system are handled in an organized and effective manner.

The defect management process for this project will include the following steps:

- Defects are identified and logged whenever the actual output does not match the expected output.
- Each defect is assigned a severity level such as **Low**, **Medium**, or **High** based on its impact on system functionality.
- Developers analyze the reported defects and implement appropriate fixes.
- After fixing the defects, re-testing is performed to verify that the issue has been successfully resolved.

This structured approach helps improve software quality, reduces the chance of recurring issues, and ensures the reliability of the system throughout the development and testing phases.