Students' Auditorium Management System

(SAMS)

TESTING PLAN

Gurram Dhanunjay(22CS10029)   Sai Deepak Reddy Mara(22CS10066)

Eswara Adithya(22CS10023)

Contents

INTRODUCTION

OBJECTIVE

This document serves as a comprehensive outline outlining the testing strategy for the Students' Auditorium Management Software. Its primary aim is to establish project-wide quality standards and procedures. It provides an overview of the project status as of the conclusion of the planning phase and delineates the standards applicable to unit, integration, and system testing within the specified application. The testing approach encompasses both white box and black box methodologies, as well as system-testing paradigms, encompassing criteria, methods, and test cases derived from the overall design. Throughout the testing lifecycle, adherence to the test documentation specifications outlined in the IEEE Standard 829-1983 for Software Test Documentation will be maintained.

TESTING STRATEGY

Testing involves analyzing a software item to identify disparities between its current state and the desired conditions, as well as assessing its functionalities. Key components of a test plan encompass:

- The objective of the test level.
- Management and technical methodologies.
- Pass/Fail criteria.
- Hardware and software prerequisites.

SCOPE

Testing will occur at multiple stages throughout the product's life cycle as it undergoes construction. Given the interdependent nature of testing, test planning becomes an ongoing process conducted throughout the system development life cycle. It is imperative to create test plans for each level of product testing

.

TESTS

Unit Testing

Unit Testing is conducted at the source code level to identify language-specific programming errors, such as syntax errors or logical flaws, and to evaluate specific functions or code modules. Unit test cases are crafted to validate the correctness of the program. This testing phase encompasses examining all classes within the program, including the graphical user interface.

White Box Testing

White box testing does not involve the user interface (UI); instead, it directly tests inputs and outputs at the code level, comparing results against specifications. This approach disregards the program's functionality and concentrates solely on its code and structure. Test cases are generated to ensure that each condition is executed at least once, with all possible values tested.

Black Box Testing

Black box testing entails examining every potential input to ensure that it produces the correct outputs, mimicking how an end-user would interact with the software. For our application, we've opted to conduct Error Guessing and Boundary Value Analysis testing methodologies.

Interface Testing

Two main modules required integration: the Graphic User Interface module and the controller module linking to the local database. Once integrated, these components constitute the entire Student's Auditorium Automation Software. Below outlines these modules and the steps necessary for seamless integration. Incremental testing strategy will be utilized for the integration process.

Graphic User Interface Testing

This module offers a straightforward Graphical User Interface (GUI) allowing users to execute various functions. Testing of this module was conducted independently from the backend to verify the proper functioning of each interface element (e.g., Login button) and to ensure that mouse-event actions were functioning correctly. Testing involved creating stubs for each element within the interface.

Test(Connection Module)

The Controllers facilitate sending requests to servers and retrieving pertinent data from the database for transmission back to the GUI. This module underwent testing independently from the GUI, with results printed out to the Console. Testing followed an incremental approach, wherein one function was tested initially, followed by the addition of additional functions and subsequent testing until all required functionalities were validated.

The following interfaces were tested:

- Sign In: Manager/Account Clerk/Sales Person/Spectators
- Show profile: Manager/Account Clerk/Sales Person/Spectators
- Edit username/password: Manager/Account Clerk/Sales Person/Spectators
- Add & view SalesPerson: Manager
- Add & View Show: Manager
- View Balance Sheet(Yearly and Show-wise): Manager
- Add and modify expenses: Account Clerk
- Book/Cancel Ticket : Sales Person
- Show Details: Sales Person
- Monitor Seat Type and availability: Sales Person
- View Bookings: Sales Person/ Spectators
- Booking request : Spectators
- Ticket Cancellation Request: Spectators

System Testing

System testing aims to identify faults that may only surface when testing the entire integrated system or significant portions of it. Typically, this testing phase prioritizes areas such as performance, security, validation, load/stress, and configuration sensitivity. However, our focus was primarily on d.function validation and performance. In both cases, we employed the black-box testing method.

Function Validation Testing

The integrated "SAMS" underwent testing based on requirements to validate the accuracy of our application. During this testing phase, we focused on identifying errors in inputs and outputs. This involved testing each function to ensure proper implementation of parsing procedures and verification of expected results. Additionally, we conducted tests on:

- Interfaces to confirm their functionality (e.g., validating that each interface behaves as intended, specifically ensuring correct action association with mouse click events).
- Interaction between the GUI and backend controller classes, including insertion of data to verify proper transmission to the server and attainment of expected outcomes.

Performance Testing

This test was conducted to evaluate the fulfillment of a system with specified performance requirements. It was done using black-box testing method. Following things were tested:

- Creating an event which requires a large number of seats to see how much time it takes to store and retrieve information of that show from the database.
- Booking a large number of seats and storing their information to the database to see how much time it takes to retrieve them from the database.
- Calculating the expenditures statistics for a very large history to test the performance of expenditure table generation.