VALIDATION, VERIFICATION, AND TESTING PLAN

Project or System Name

Revision Sheet

Release No.	Date	Revision Description
Rev. 0	5/30/00	Validation, Verification, and Testing Plan Template and Checklist
Rev. 1	4/12/02	Conversion to WORD 2000 format

VALIDATION, VERIFICATION AND TESTING PLAN

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION

1.1 Purpose

The purpose of this document is to ensure that the system meets all requirements and can perform its functions to full effect. This plan outlines the testing strategies being undertaken and serves to mitigate risk and increase confidence in the product. This plan also serves as a documentation to the validation and verification of the system.

1.2 Scope

The scope of this document consists of an overview of the system and its requirements. It also consists of comprehensive test cases and information to prepare, run and evaluate them.

1.3 System Overview

The railway ticketing portal is a fullstack web application, which consists of a React-based frontend application and a Java Spring API. The system uses Postgres for its database operations. The ticketing portal allows users to book tickets for one-way or multistep train journeys. The users can apply for different discounts if they meet certain criteria.

- Software Solutions Ltd.
- Railway ticketing portal
- 1100
- System category
  - *Major application*: performs clearly defined functions for which there is a readily identifiable security consideration and need
- Operational status
  - Under development
- The system requires Docker to run.

1.4 Project References

Additional references are the files provided in the documentation folder of the project.

168931.5 Acronyms and Abbreviations

No particular abbreviations have been used in this document.

2.0 TEST EVALUATION

2.1 Requirements Traceability Matrix

A requirement traceability matrix is provided as an additional document in the documentation folder of the project.

2.2 Test Evaluation Criteria

In this section the functional requirements for a minimum viable product that will fulfill all our users' needs are listed.
The system must allow the user to choose a start and end destination and in addition up to 4 intermittent destinations can be chosen for a multi-stop trip. Based on the departure time chosen and if the user meets certain criteria, multiple discounts can be applied to the fixed ticked price. The system also allows the user to modify their booking or outright cancel it. Users can also modify their specific user data as well. Online payments are enabled, and the system is available in English, German, Spanish and French. Furthermore, the users will have access to a discount application form if they meet the required criteria.

2.3 User System Acceptance Criteria

The system must be resilient enough to perform without fault under high traffic conditions. All relevant operations must happen within a small timeframe as to not impede the user expectations. All relevant values are provided in document "Lab 01". The system will handle exceptions without impeding the user experience. GDPR will be upheld to ensure the safety of all sensitive user data.

3.0 TESTING CHARACTERISTICS

3.1 Testing Conditions

The testing process requires a Postgres database. The script to build the Docker container that houses the testing database is provided within the docker folder of the project.

3.2 Extent of Testing

The project is covered by integration test for the web controllers and unit testing for the calculating function. Additionally, frontend test will be written as well and End to End tests are considered once the project nears completion. Coverage will be measured by line, statement, and decision coverage.

3.3 Data Recording

Data will be recorded via testing logs.

3.4 Testing Constraints

Currently, testing limitation is the lack of testers in the project, as all our testers are spread thin across multiple projects. However, experienced candidates are being considered for testing position within the company, which will alleviate the pressure and better the situation.

3.5 Test Progression

Tests will be run from unit to integration to end-to-end tests, to ensure linear progression.

3.6 Test Evaluation

3.6.1 Test Data Criteria

Test results must show that the piece of code being tested performs its function correctly, as in fulfills its purpose, and reliably – as in a performant way and with fault tolerance.

3.6.1.1 Tolerance

Functionality-wise the system must be faultless. For example, the price calculation function must be pure and calculate the price precisely. However, non-functional requirements can deviate in a range of one to two seconds, as unexpected faults can occur in the system, which can slow it down.

3.6.1.2 System Breaks

The system must be able to recover from a maximum of three sequential system breaks.

3.6.2 Test Data Reduction

Test data will be generated by our testers after careful study of the user requirements. This test data will consist of identification numbers, test case descriptions, test inputs and expected test outputs. This data will be stored in an internal testing tool and more test case data will be derived from it by combining functional and non-functional test cases.

4.0 TEST DESCRIPTION

This section contains an example description of one test case for the project.

4.1 Family user booking a ticket for 2 destination in rush hour with one child

This test case explores if the calculation function will produce the correct output of the price (price reduced by 50% with no additional discounts applied) in this specific situation.

4.1.1 System Functions

This test tests the calculation function in the BookingService.

4.2.2 Test/Function Relationships

This test is independent of other tests.

4.1.3 Means of Control

This test is run manually by our developers.

4.1.4 Test Data

Identify any security considerations in each of the following subsections.

4.1.4.1 Input Data

This test uses data relevant to the successful run of the test. This includes the id of a test user with the family discount card, information that the user will travel with a child, a departure time within early or late rush hour and a list of 2 ids that represent valid destination in our testing database.

4.1.4.2 Input Commands

For the test to be executed the developer requires their IDE, which is fully setup with the relevant project dependencies such as Mockito for testing. Then the developer must have a valid Junit configuration, after which they must simply execute the test and observe the logs.

4.1.4.3 Output Data

The test will show in the testing logs produced by the application if it has passed or not. If failure has occurred, the test will provide information about at which line of code this has happened.

4.1.4 Test Procedures

4.1.4.1 Procedures

No procedures are required, besides a fully setup-ed project.

4.1.4.2 Setup

The developer requires a fully seup-ed project on a Macintosh machine with an M2 processor to ensure that all tests or all batches of tests are executed as fast as possible.

4.1.4.3 Initialization

To initialize the running of the test the developer musty have written the respective unit test in the testing file located in the project.

4.1.4.4 Preparation

The developer must check the defined test case's inputs and encode them in the unit test.

4.1.4.5 Termination

After test termination we do not need to clean up the testing database, as we are testing the function with DTOs that are prepared within the test itself.