**VALIDATION, VERIFICATION, AND TESTING PLAN**

*Railway Ticketing Portal*

**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

1.1 Purpose 1-1

1.2 Scope 1-1

1.3 System Overview 1-1

1.4 Project References 1-1

1.5 Acronyms and Abbreviations 1-1

2.0 TEST EVALUATION 2-2

2.1 Requirements Traceability Matrix 2-2

2.2 Test Evaluation Criteria 2-2

2.3 User System Acceptance Criteria 2-2

3.0 TESTING CHARACTERISTICS 3-1

3.1 Testing Conditions 3-1

3.2 Extent of Testing 3-1

3.3 Data Recording 3-1

3.4 Testing Constraints 3-1

3.5 Test Progression 3-1

3.6 Test Evaluation 3-1

3.6.1 Test Data Criteria 3-1

3.6.1.1 Tolerance 3-1

3.6.1.2 System Breaks 3-1

3.6.2 Test Data Reduction 3-2

4.0 TEST DESCRIPTION 4-3

5.x [Test Identifier] 4-3

5.x.1 System Functions 4-3

5.x.2 Test/Function Relationships 4-3

5.x.3 Means of Control 4-3

5.x.4 Test Data 4-3

5.x.4.1 Input Data 4-3

5.x.4.2 Input Commands 4-3

5.x.4.3 Output Data 4-4

5.x.4.4 Output Notification 4-4

5.x.5 Test Procedures 4-4

5.x.5.1 Procedures 4-4

5.x.5.2 Setup 4-4

5.x.5.3 Initialization 4-4

5.x.5.4 Preparation 4-4

5.x.5.5 Termination 4-5

# GENERAL INFORMATION

## 1.1 Purpose

When a validation and verification testing are used in a testing of a software, it is ensured that the software meets the business(client)’s essentials as their outlined in the “Functional” and in the “Non-functional” requirements at the same start of the project. The testing plan has the goal of being the documentation where the tester has been described how the system will be tested, so at the end to be sure that it meets the business needs.

## 1.2 Scope

Describe the scope of the Validation, Verification, and Testing Plan as it relates to the project.

## 1.3 System Overview

1. Responsible organization: TU Sofia, discipline: Software Testing
2. System name or title: Railway Ticketing Portal
3. System code: <https://github.com/DeislavaaSimeonova/RailwayTicketingPortal> , branch: main
4. System category
5. ***Major application*:** performs clearly defined functions for which there is a readily identifiable security consideration and need
6. *General support system*: provides general ADP or network support for a variety of users and applications
7. Operational status
8. Operational
9. **Under development:**
10. Undergoing a major modification
11. System environment and special conditions

## 1.4 Project References

* There is no documentation written.

## 1.5 Acronyms and Abbreviations

* There is no documentation written.

# TEST EVALUATION

## 2.1 Requirements Traceability Matrix

Requirements traceability matrix can be found in file: “lab6\_responsibility\_matrix”.

## 2.2 Test Evaluation Criteria

* All functional & non-functional requirements can be found in file: “lab2-req-improvement-and-task-2”.

## 2.3 User System Acceptance Criteria

The User System Acceptance Criteria can be found in file: “lab2-req-improvement-and-task-2”.

# TESTING CHARACTERISTICS

## 3.1 Testing Conditions

The manual testing will be made with data from the database and the normal input.

## 3.2 Extent of Testing

1. Limited testing:  
   Percentage of Test Coverage: 80%

* Number of Test Cases to cover critical/high-risk areas: 12/22
  + Test cases can be found in file: “lab1-requirements” and “lab6\_responsibility\_matrix”.
* Discrete Operating Conditions: testing of boundary values and equivalence class portioning
  + Examples can be found in file: “lab6\_functional\_testing”.

## 3.3 Data Recording

Indicate data recording requirements for the testing process, including data not normally recorded during system operation.

## 3.4 Testing Constraints

Tests are provided with Jupiter in Junit.

## 3.5 Test Progression

The testing will be made not only with unit tests but also with integration tests. The development of the program will also need manual testing when there is an interface and automation tests.

## 3.6 Test Evaluation

### 3.6.1 Test Data Criteria

The description of every feature includes how the exceptions will be handled and how the user will be redirected.

#### 3.6.1.1 Tolerance

If there is a case when the system is implemented in some way, but the refactoring would take a lot of developers’ time and the benefit would be not good, the problem will be marked as a “known issue”. In meantime the testing is working with only 3 destinations: Vienna, Munich, and Frankfurt. Any other city is invalid.

#### 3.6.1.2 System Breaks

At this moment the system is not dependent on any other systems.

### 3.6.2 Test Data Reduction

At this point the test success, could be seen only within the unit tests – there is a mocked data and processes so in this way to be ensured that the software is working by requirements.

In the upcoming time additional features could be added to the software – tools like JaCoco and Sonar for the backend.

# TEST DESCRIPTION

## 4.x [Test Identifier]

This can be found in file “lab6\_responsibility\_matrix”.

### 4.x.1 System Functions

* All functional & non-functional requirements can be found in file: “lab2-req-improvement-and-task-2”.

### 4.x.2 Test/Function Relationships

Provide a list of the tests that constitute the overall test activity. Include a test/function matrix summarizing the overall allocation of the system tests to the functions.

* This can be found in file “lab6\_responsibility\_matrix”.

### 4.x.3 Means of Control

Tests is to be conducted by manual means.

### 4.x.4 Test Data

Identify any security considerations in each of the following subsections.

* No security is implemented in the code.

#### 4.x.4.1 Input Data

Describe the manner in which input data are controlled in order to test the system with a minimum number of data types and values, exercise the system with a range of bona fide data types and values that test for overload, saturation, and other “worst case” effects, and exercise the system with bogus data and values that test for rejection of irregular input.

The input data is controlled via validation in the services that it is “travelling” through. By “wrong” data there is a signal that is set {an exception is thrown}.

#### 4.x.4.2 Input Commands

Currently, the only available test implementation is via Unit Tests in Java. The initialization starts with assigning the data to the corresponding needs (for example, for search by criteria – a city is assigned to field ‘startDestination’, another one to field ‘endDestination’ and an hour is passed to field ‘departureTime’). Then the needed functions are mocked, and the test is being started manually. By problem and unsuccessful end of the test, a debugging process is started so the error to be identified. After the repair of the error the test is started again manually.

#### 4.x.4.3 Output Data

The expected output data is introduced in the tests as constants, so after the test is being run – the values from the function are assigned to some variables. Then, the assigned variables and the constants are checked for equality.

#### 4.x.4.4 Output Notification

* There are none output notifications. The only way to understand if the tests are successful, is after manual running and result in the console – build success or build failure.

### 4.x.5 Test Procedures

#### 4.x.5.1 Procedures

Describe the step-by-step procedures to perform each test.

#### 4.x.5.2 Setup

The unit tests in this project are implemented via the test engine Junit Jupiter.

#### 4.x.5.3 Initialization

1. Add needed dependencies for Junit Jupiter.
2. Initialize the test class.
3. Introduce the needed annotations.

#### 4.x.5.4 Preparation

1. Prepare constant variables as expected results.
2. Mock the functions that are needed for the test.
3. Assign the results to some variables (actual values).
4. Check for equality between expected and actual values.

#### 4.x.5.5 Termination

In my case (testing with unit tests), the termination happens automatically when the test is run.