Test Plan

Group Assignment

# **Document Control**

|  |  |
| --- | --- |
| **Document Name** | Test Plan |
| **Reference Number** | - |
| **Version** | V1.0 |
| **Project Code** | UECS2354 Software Testing Group Assignment |
| **Status** | [to be updated] |
| **Date Released** | [to be updated] |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Position** | **Signature** |
| Prepared By: | Edmund Chan Chee An |  |  |
| Reviewed By: |  |  |  |
| Approved By: |  |  |  |
| Approved By: |  |  |  |

# **Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Release Date** | **Section** | **Amendments** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Content

[Document Control 1](#_Toc127346001)

[Version History 2](#_Toc127346002)

[1.0 Introduction 1](#_Toc127346003)

[1.1 Objective 1](#_Toc127346004)

[1.2 Scope 1](#_Toc127346005)

[1.3 References 1](#_Toc127346006)

[2.0 Test Plans 2](#_Toc127346007)

[2.1 Test Items 2](#_Toc127346008)

[2.2 Features to be tested. 2](#_Toc127346009)

[2.3 Features not to be tested. 2](#_Toc127346010)

[2.4 Test Conditions 2](#_Toc127346012)

[2.5 Entry Criteria 2](#_Toc127346013)

[2.6 Exit Criteria 2](#_Toc127346014)

# **Introduction**

## 1.1 Objective

The objective of this test plan is to define the testing approach and activities for Travel Ticket Booking and Fare Calculation System. The main purpose of this project is to verify the system core functionalities including ticket booking, user registration, fare calculation, user detail management, route distance handling, discount and surcharge application, and payment method adjustment.

This test plan is to ensure that the developed testing modules meet the specifies functional requirements (FR1 – FR11) by identifying and addressing defects early in the development cycle. The testing objectives are:

* Verify that the system produces expected and correct output for valid input.
* Detect errors, failure and inconsistences through systematic test design techniques.
* Determine the reliability, quality and stability of the module before integration with other systems.
* Ensuring the implemented system aligns with user requirement including distance-based fare rates, passenger-type discounts, time/date surcharges and payment method adjustments.

By executing this test plan, the development team able to validate that system functions as intended, ensuring accuracy and reliability of system.

## 1.2 Scope

The scope of this test plan covers both unit testing and integration testing for the modules implemented in Travel Ticket Booking and Fare Calculation System

Each individual class and method will be tested to ensure correctness of output. Unit testing will be applying these test design techniques

* Equivalence Partitioning (EP) and Boundary Value Analysis (BVA) for input validation (example, fare ranges, passenger types and distances)
* Decision Table (DT) to handle discount and surcharge combination
* Test Doubles (stub and mocks) using Mockito to simulate external dependencies such as routeInfo and applyDiscountSurcharge
* Parameterized tests with JunitParams to cover multiple inputs scenarios

**Unit Testing**

Unit testing will be focus on verifying the small and isolated parts of an application call “Units” like function, method, and class. Modules such as Booking.java. CalculateFare.java, FareAdjustment.java, FileFunctionality.java, GuestFile.java, RouteInfo.java and UserFile.java will be tested.

**Integration Testing**

Integration Testing will focus on verifying interaction between related modules such as Booking.java, CalculateFareIntegration.java, RouteInfo.java, GuestFile.java and UserFile.java. The integration test will ensure that combined outputs match the expected results when modules interact.

## 1.3 Test Basis

The test condition and designs are based on following resources

* UECS2354 Software Testing Group Assignment Details
* Feature listing table (FR1 to FR11)
* Decision Table and application code created by other members
* Development environment: Java Language and Eclipse IDE
* External tools: Junit 4, Mockito, Junit Params

## 1.4 References

## UECS2354 Software Testing Lecture Slide Chapter 5 Entry Criteria and Exit Criteria

# **2.0 Test Plans**

## Test Items

1. Booking.java
   1. public List<String> getDiscountDetails()
   2. public void makePayment()
2. CalculateFare.java

2.1 public void calculateTotalFare()

2.2 public void calculateDiscountFare()

2.3 public void calculatePayment()

1. FareAdjustment.java

3.2 public String validatePassengerType()

3.3 public double passengerAdjustment()

3.4 public Boolean isWeekend()

3.5 public int validateTravelTime()

3.6 public int daytTimeAdjustment()

3.7 public String validatePaymentMethod()

3.8 public double paymentMethodAdjustment()

1. FileFunctionality.java

4.1 public void writeToFile(String[] input Array, String fileName)

4.2 public String[] readFromFile(String fileName)

1. GuestFile.java

5.1 public List<Guest> readGuestFromFile()

5.2 public void writeGuestFile()

1. RouteInfo.java

6.1 private List<Station> stationList = new ArrayList<>()

6.2 public RouteInfo()

1. UserFile.java()

7.1 public List<User> readUserFromFile(String filename)

7.2 public void writeUserToFile(User[ ] userArray, String filename)

## Features to be tested.

1. **Booking.java**
2. **Booking Status**

* Validate that booking status changes to “Confirmed Booking” after successful payment
* Confirm that the default booking status is not Paid

1.2 **Payment Status**

* Ensure correct handling of invalid payment scenarios
* Validate that payment status update correctly after payment (“Paid” or failure”)
* Verify successful payment when a valid fare is calculated and supported payment method is used

1.3`**Discounted Fare Calculation**

* Verify calculation of total fare after applying discount based on personal type
* Verify day and time-based adjustment
* Ensure adjustment details are correctly generated
* Confirm system throw exception for invalid output

1.4 **Discount Details**

* Verify that detailed adjustment breakdown is available when discount apply
* Ensure an exception is thrown when no discount details are available

1. **CalculateFare.java**

2.1 **Total Fare Calculation**

* Correct calculations of fare based on route distance
* Invalid input handling (null station, invalid station, same station, invalid route)

2.2 **Calculate Discounted Fare**

* Valid discounted fare calculation applied correctly
* Invalid output handling (null/invalid travel day, null/invalid travel time, null/invalid stations, null passenger type, null passenger quantity, incorrect type & quantity length)

2.3 **Total Fare Calculation**

* Correct calculations of fare based on route distance
* Invalid input handling (null station, invalid station, same station, invalid route)

2.4 **Payment Calculation**

* Ensure exception is thrown if discounted fare has not been calculated (IllegalArgument Exception(Fare Has Not Been Calculated”)
* Correctly apply payment method and adjustment, example credit surcharge and e-wallet discount)
* Verify final paymentAmount is updated

1. **FareAdjustment.java**

3.1 **Verify Passenger Type**

* Ensure that the passenger types are valid (e.g adult, senior citizen, student and child)
* The passenger type cannot be empty else system will throw “Passenger Type Cannot Be Null”
* The passenger value must be adult, senior citizen, children or student, else system will throw “Invalid Passenger Type”

3.2 **Passenger Adjustment**

* Verify the travel distance range if the distance is smaller than 1 or greater than 30 throw “Invalid Travel Distance”
* Ensure the travel distances and passenger types are valid

3.3 **Is Weekend Checks**

* Ensure days of the week are valid (weekend & weekday)
* Invalid input handling (unrecognized day

3.4 **Validate Travel Time**

* **E**nsure time formats within range 0000 to 2359
* Invalid time handling (null, wrong format, out of range)

3.5 **Day Time Adjustment**

* Valid day and time combination
* Invalid input handling (null, wrong format, out of range)

3.6 **Payment Method Adjustment**

* Valid payment method (credit card, e-wallet, online banking)
* Invalid payment method handling (null or unrecognized)

3.7 **Payment Method Adjustment**

* Valid payment method adjustment factors
* Invalid payment method handling (null or unrecognized)

1. **FileFunctionality.java**

4.1 **Read and Write File Functionality**

* Verify data can be correctly written to file and read back

4.2 **Write to File**

* Invalid input array (null)
* Invalid file name (null)
* Invalid file path (empty string)

4.3 **Read from File**

* Invalid file name (null)
* Invalid file name (empty string)

1. **GuestFile.java**

5.1 **Read and Write Guest Data**

* Verify that guest objects can be correctly written to file and read back

5.2 **Read Guest from File**

* Invalid file data format (invalid field)
* Invalid file name (null)
* Invalid file path (empty string)

5.3 **Write Guest to File**

* Null guest list
* Null file name
* Invalid file path (empty string)

1. **RouteInfo.java**

6.1 **Station Validation**

* Valid station names return the correct Statiom
* Invalid station names (null, unknown station) throw IllegalArgumentException

6.2 **Route Distance Calculation**

* Valid start and end station return correct station
* Invalid scenarios (null station name, invalid station name, same start and end station, no route between station) throw IllegalArgumentException

1. **UserFile.java**

7.1 **Read and Write User Data**

* Verify that user file can be correctly written to file and read back

7.2 **Read User from File**

* Invalid file data format throw IllegalArgumentException
* Invalid file parameters (null file name, empty file path) throw IllegalArgumentException

7.3 **Write User to File**

* Null user array
* Null file name
* Invalid file path (empty string)
  1. **Feature will not be tested**

1. Booking.java

* Getter method for this class

1. CalculateFare.java

* Getter method for this class
* Setter method for this class

1. FareAdjustment.java

* Getter method for this class

1. Guest.java
   * Getter method for this class
   * Setter method for this class
2. IUser.java

* Getter method for this class
* Setter method for this class
* Public String toString()
* Public Boolean equals(Object o)
* Public int hashCode()

1. Payment.java

* Public void makePayment(double payment)
* Getter method for this class
* Public void emailReceipt

1. Station.java

* Getter method for this class
* Public HashMap<Station, Double> getLinkedStations()
* Public void addStation(Station nextStation, double distance)
* Public Boolean equals(Object obj)
* Pulic int hashCode()

1. User.java

* Getter method for this class
* Setter method for this class
* Public String toString
* Public Boolean equals(Object o)
* Public in hashCode()

## 2.4 Test Condition

1. **Booking.java**

1.1 **Get Total Fare**

* Valid inputs
  + Start: KL Sentral, End: Subang Jaya = RM15.00
  + Start: KL Sentral, End: MidVally = RM2.00
* Invalid input
  + Null start and end station
  + Invalid start and end station
  + Same start and end station (KL Sentral to KL Sentral)
  + Route not supported (Kajang to Taman Melati)
  + Route exceed distance limit
  1. **Get Discount Fare + Discount Details**
* Valid Input
  + Travel Day: Wednesday, Time: 1200, Start: Midvally, End KL Sentral, Passengers: 2 Adult, 2 Child, 1 Senior Citizens = RM5.00
  + Travel Day: Saturday, Time: 2100, Start KL Sentral, End: Gombak, Passengers: (2 Adult, 1 Child, 3 Senior Citizens) = RM92.00
* Invalid Input
  + Null or invalid travel day
  + Null or invalid travel time
  + Null or invalid start station
  + Null or invalid end station
  + Null or empty passenger type
  + Null or empty passenger quantity
  + Mismatched length between passenger type and quantity lists
  1. **Make Payment**
* Valid Input
  + Travel Day: Monday, Time 0900, Start: Titiwangsa, End: KL Sentral, Passengers: (2 Adult, 2 Child), Payment method: e-wallet, Booking status: “Confirm Booking”.
* Invalid Input
* Null payment method
* Invalid payment method
* Payment attempted before calculating fare

1. **CalculateFare.java**

2.1 Calculate Total Fare

* Valid cases
  + Titiwangsa to Batu Kentonmen = RM15.00
  + KL Sentral to Gombak = RM20.00
* Invalid cases
  + Start station is null or invalid
  + End station is null or invalid
  + Start and end stations are same
  + Invalid route
  + Route exceed distance limit

2.2 Calculate Discount Fare

* Valid cases
  + Correct discount fare calculated when travel date, travel time, stations, passenger type and passenger quantities are valid
  + Adjustment details generated correctly
* Invalid cases
* Null or invalid travel day
* Null or invalid travel time
* Null or invalid start station
* Null or invalid end station
* Passenger type is null
* Passenger quantity is null
* Passenger type and quantity list length does not match

2.3 Calculate Payment

* Valid cases
  + Payment amount is correctly calculated (e.g., payment amount = RM10)
    - E-Wallet = RM10.00
    - Credit Card = RM10.50
    - Online Banking = RM9.50
* Invalid cases
  + Total fare = 0.0
  + Payment method is null or invalid

1. **FareAdjustment.java**

3.1 Validate Passenger Type

* Valid cases
  + Passenger type normalized correctly (adult, student, senior citizen, child)
* Invalid cases
  + Invalid or null passenger type

3.2 Passenger Adjustment

* Valid cases
  + Correct adjustment applied based on passenger type and distance
  + Child fare adjustment differs correctly for distance < 5 km and >5 km
  + Senior citizen discount applied correctly
  + Student discount applied correctly
* Invalid cases
  + Travel distance is negative, zero or greater than 30 km
  + Invalid or null passenger type

3.3 Is Weekend

* Valid cases
  + Correct identified Monday to Friday is non-weekend
  + Correct identified Saturday and Sunday is weekend
* Invalid cases
  + Invalid or null travel day

3.4 Validate Travel Time

* Valid cases
  + Travel time correctly validated as values like 0000, 0900, 2359
* Invalid cases
  + Invalid or null travel time
  + Travel time out of range (2400, -1000)
  + Travel time has invalid format (0060, 1099)

3.5 Day Time Adjustment

* Valid cases
  + Peak hour surcharge correctly applied
  + No adjustment applied during normal weekday hours
  + Midnight hours adjustment handled
  + Weekend discount correctly applied
* Invalid cases
  + Invalid or null travel day
  + Invalid or null travel time

3.6 Validate Payment Method

* Valid cases
  + Payment method normalized correctly
* Invalid cases
  + Invalid or null payment method

3.7 Payment Method Adjustment

* Valid cases
  + Correct surcharge applied for credit card
  + Correct discount applied for online banking
  + No discount and surcharge for e-wallet
* Invalid cases
  + Invalid or null payment method

1. FileFunctionality.java

4.1 Write to File

* Valid cases
  + Data successfully written to the file when both input array and file path are valid
* Invalid cases
  + Null input array
  + Null or empty file path

4.2 Read data from files

* Valid cases
  + Date is successfully read from file when valid file path is provided
  + Data read matches the content written previously
* Invalid cases
  + Empty and null file path

4.3 Read and Write Function

* Valid cases
  + Combined test ensures data written to file can be read back correctly

1. GuestFile.java

5.1 Write and Read Guest Integration

* Valid cases
  + Verify that guest are successfully written to file and can be read back
  + Ensure data integrity between written and read guests

5.2 Read data from files

* Valid cases
  + Guest are written successfully when guest list and file path are valid
* Invalid cases
  + Empty and null file path
  + Guest list is null

5.3 Read and Write Function

* Valid cases
  + Guest are read successfully when file has valid guest data
* Invalid cases
  + Empty and null file path
  + File contains invalid guest data (missing phone number)

1. RouteIndo.java

6.1 Validate Station

* Valid cases
  + Verify that a valid station name returns the correct Station object
  + Example: “KL Sentral” return Station(“KL Sentral”)
* Invalid cases
  + Null and invalid station name

6.2 Get Route Distance

* Valid cases
  + Verify correct distance is returned when valid start and end are provided
  + Example: KL Sentral to Mid Valley = 5.0 km
  + Ensure different pairs of valid station return expected distances
  + Example: KLCC to Ampang Park = 1.0 km, Sungai Buluh to Rawang = 12.0 km
* Invalid cases
  + Null or invalid start station = throw exception
  + Null or invalid end station = throw exception
  + Same station start and end station (KLCC to KLCC) throw exception
  + Invalid route combination where no connection exists (Kajang to Taman Melati) = throw exception

1. UserFile.java

7.1 Read and Write User

* Valid cases
  + Verify that a valid user list written to file can be read back correctly
  + Example: [User(“USER001”, “MIKU”, miku@gmail.com, “0123456789”)]
* Invalid cases
  + Empty and null file path
  + User object with missing or invalid field values in file data

7.2 Write User to File

* Valid cases
  + User list with valid details and valid file path successfully written to file
* Invalid cases
  + Empty and null file path
  + Valid user list but null file name

## 2.5Entry Criteria

## System Testing can begin when

1. Bug tracking and test tracking system are in place and operational
2. All components are under formal and automated configure
3. The operation teams have configured the system test environment, including all target hardware components and subsystems
4. Development Teams have completed all features and bug fixes scheduled for the release
5. Development Teams have performed unit testing on all features and bug fixes scheduled for release
6. Less than 50 bugs remain open against the release
7. Development team deliver software build to the testing team at least 3 business days prior to System test start
8. The team completes a 3-day smoke test and reports the results at the system test phase entry meeting
9. The project management team formally agrees in the system test phase entry meeting to proceed, with the following decisions made

* Confirming the code implementation and unit testing are complete
* Assign target fix dates for any detected bugs

## 2.6 Exit Criteria

## System Testing can be considered complete when

## No design, code or feature changes have occurred in the prior 3 weeks

## No client system became inoperable due to failed updates during system test

## The test team has executed all planned test against the FA-candidate software build.

## Development team have resolved all must fix-bugs

## The test team has confirmed that all issues in the bug tracking system are differed and fixes have been validated through regression and confirmation testing

## The project management team agrees that the project as defined during the final System Test cycle satisfies the customer reasonable expectations of quality

## The test closure report prepared and approved by teams

## A formal system test phase exit meeting is held and the project management teams agrees that the system testing is complete