Test Plan

Group Assignment

# **Document Control**

|  |  |
| --- | --- |
| **Document Name** | Test Plan |
| **Reference Number** | - |
| **Version** | V1.0 |
| **Project Code** | UECS2354 Software Testing Group Assignment |
| **Status** | Completed |
| **Date Released** | 12 September 2025 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Position** | **Signature** |
| Prepared By: | **Edmund Chan Chee An** | **Team Member** | ***Edmund*** |
| Reviewed By: | **Quak Jing** | **Team Member** | ***Quak*** |
| Approved By: | **Leon Siow Yi Hong** | **Group Leader** | ***Leon*** |
| Approved By: | **Narvin A/L Chandrasegaran** | **Team Member** | ***Narvin*** |

# **Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Release Date** | **Section** | **Amendments** |
| 1.0 | 22 Aug 2025 | Introduction | None |
| 2.0 | 29 Aug 2025 | Test Plans | None |
| 3.0 | 12 Sept 2025 | Document Review | Test Plans Items |

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 Basis 2](#_Toc127346011)

[2.5 Test Conditions 2](#_Toc127346012)

[2.6 Entry Criteria 2](#_Toc127346013)

[2.7 Exit Criteria 2](#_Toc127346014)

# **Introduction**

## 1.1 Objective

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

This test plan is to ensure that the developed testing modules meet the specified 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 inconsistencies 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 would be able to validate the system function 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 System. Each individual class and method will be tested to ensure correctness of output. Unit testing will be applying these test design techniques:

1. Equivalence Partitioning (EP) and Boundary Value Analysis (BVA) for input validation (e.g. distance and day/time)
2. Decision Table (DT) to handle discount and surcharge combination
3. Test Doubles (stub and mocks) using Mockito to simulate external dependencies
4. Parameterized tests with JunitParams to cover multiple input scenarios

**Unit Testing**

Unit testing will be focusing on verifying the small and isolated parts of an application call “Units”, specifically unit methods. 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, CalculateFare.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 team members
* Development environment: Java Programming Language and Eclipse IDE
* External tools: Junit 4, Mockito, JunitParams

## 1.4 References

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

# **2.0 Test Plans**

## 2.1 Test Items

1. Booking.java

1.1 public double getTotalFare()

1.2 public double getDiscountedFare()

1.3 public List<String> getDiscountDetails()

1.4 public void makePayment()

2. CalculateFare.java

2.1 public void calculateTotalFare(String startStation, String endStation)

2.2 public void calculateDiscountedFare(String travelDay, String travelTime, String startStation, String endStation, List<String> passengerType, List<Integer> passengerQuantity)

2.3 public void calculatePayment(String paymentMethod)

3. FareAdjustment.java

3.1 public String validatePassengerType(String passengerType)

3.2 public double passengerAdjustment(String passengerType)

3.3 public boolean isWeekend(String travelDay)

3.4 public int validateTravelTime(String travelTime)

3.5 public int dayTimeAdjustment(String travelDay, String travelTime)

3.6 public String validatePaymentMethod(String paymentMethod)

3.7 public double paymentMethodAdjustment(String paymentMethod)

4. RouteInfo.java

4.1 public Station validateStation(String stationName)

4.2 public double getRouteDistance(String startStation, String endStation)

5. FileFunctionality.java

5.1 public void writeToFile(String[ ] inputArray, String fileName)

5.2 public String[ ] readFromFile(String fileName)

6. GuestFile.java

6.1 public List<Guest> readGuestFromFile(String fileName)

6.2 public void writeGuestToFile(Guest[ ] guestArray, String fileName)

7. UserFile.java

7.1 public List<User> readUserFromFile(String fileName)

7.2 public void writesUserToFile(User[ ] userArray, String fileName)

## 2.2 Features to be tested.

**1. Booking.java**

1.1 getTotalFare() - This method will get the total fare from bookingStatus

1.2 getDiscountedFare() – This method will get the total discounted fare from bookingStatus

1.3 getDiscountDetails() – This method will get the discount details from bookingStatus

1.4 makePayment() – This method will payment mock test

**2. CalculateFare.java**

2.1 calculateTotalFare() – This method calculates total fare

2.2 calculateDiscountedFare() – This method will calculate discounted fare

2.3 calculatePayment() – This method will calculate payment for fare based on payment method

**3. FareAdjustment.java**

3.1 validatePassengerType() – This method will validate passenger type and return formatted passenger type

3.2 passengerAdjustment() – This method will do fare adjustment based on passenger type

3.3 isWeekend() – This method will verify travel date and return Boolean isWeekend

3.4 validateTravelTime() – This method will validate travel time and return time as integer

3.5 dayTimeAdjustment() – This method will make payment adjustment based on day and time

3.6 validatePaymentMethod() – This method will validate the payment method

3.7 paymentMethodAdjustment() – This method make fare adjustment based on payment method

**4. RouteInfo.java**

4.1 validateStation() – This method will validate the station name and return station object

4.2 getRouteDistance() – This method will find start and end station based on algorithm created

**5. FileFunctionality.java**

5.1 writeToFile() – This method will write the array of String into file

5.2 readFromFile() – This method will rad data from specified file path and parse to array of String

**6. GuestFile.java**

6.1 readGuestFromFile() – This method will read guest data from file

6.2 writeGuestToFile() – This method will read new guest to specified file

**7. UserFile.java**

7.1 readUserFromFile() – This method will read user data from file

7.2 writesUserToFile() - This method will read new user to specified file

## 2.3 Features not to be tested.

1. Station.java – class to instantiate each station for path finding algorithm used in RouteInfo.java

2. Guest.java – class to store information such as name, email, phoneNo of a Guest

3. User.java – class to store information such as ID, name, email, phoneNo of a User

4. Payment.java – class created as a mock to interact with other core modules

5. emailNotification – an additional feature to email the receipt to the user/guest once payment is finished

## 2.4 Test Conditions

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

**1.2 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

**1.3 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

## 2.5 Entry Criteria

**System Testing can begin when**

* Bug tracking and test tracking system are in place and operational
* All components are under formal and automated configure
* Development Teams have completed all features and bug fixes scheduled for the release
* Development team deliver software build to the testing team at least 3 business days prior to System test start
* The project management team formally agrees in the system test phase entry meeting to proceed, with the following decisions made
  1. Confirming the code implementation and unit testing are complete
  2. 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 1 weeks
* No client system became inoperable due to failed updates during system test
* The development team has resolved all must fix bugs
* The project management team agrees that the project as defined during the final System Test cycle satisfies the customer reasonable expectations of quality
* A formal system test phase exit meeting was held, and the project management teams agreed that the system testing is complete