SC2006

Lim Li Ping Joey

Roger Kwek Zong Heng

Matz Chan

Ong Hong Xun

Wan Li Xin Yuan

Cheah Wei Jun

Bussin Buses

Bussin Buses is a mobile app designed for NTU students who live far from campus. It offers a direct shuttle service with no intermediate stops, aiming to provide a faster and more convenient commute. Key features include real-time tracking, seat booking, and optimized routes, making it a reliable transport option for students.

Advanced Features:
1. Routing
2. Rerouting
3. Using Computer Vision to detect congestion from traffic cameras
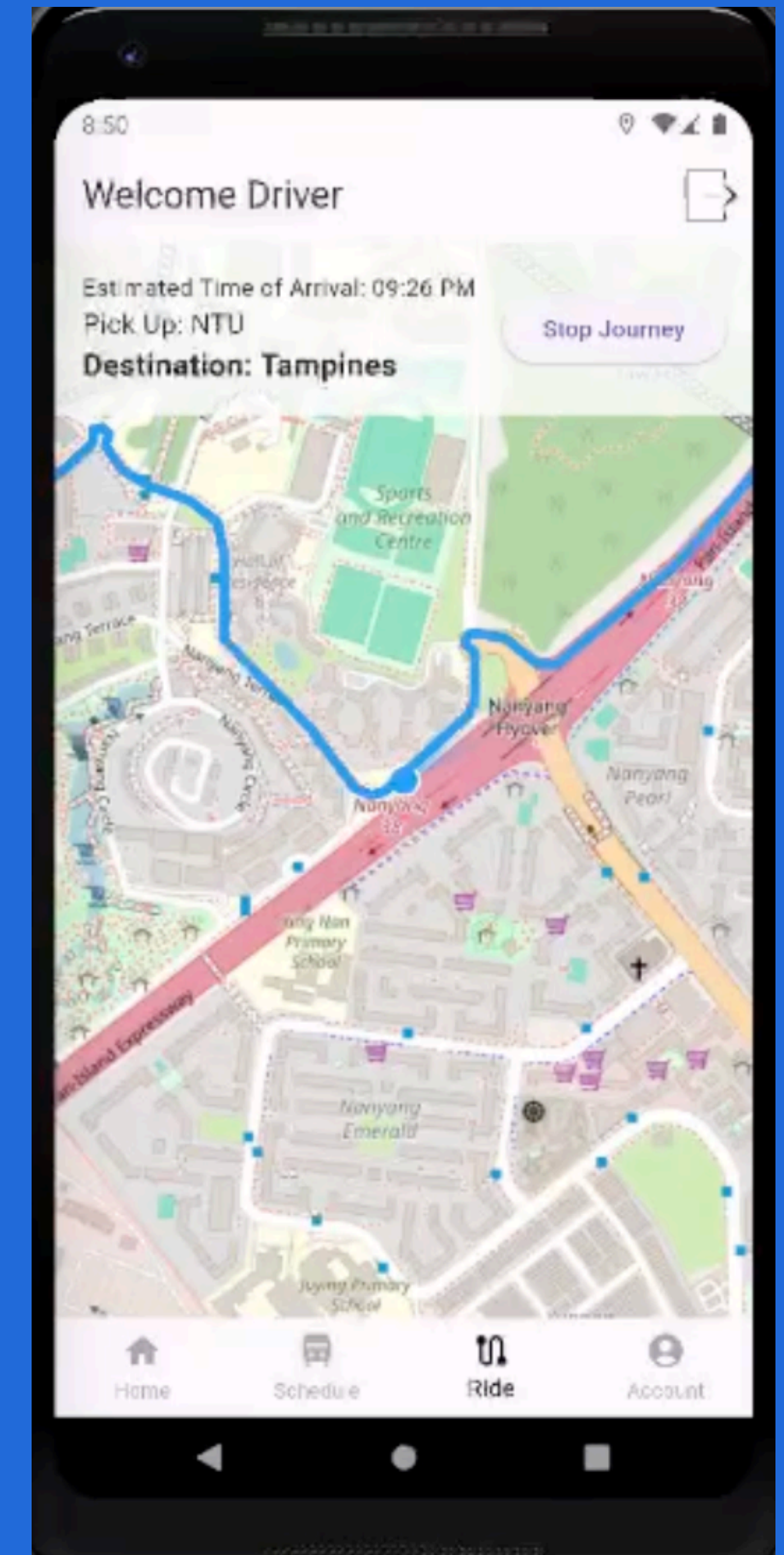
Single Responsibility Principle

- Each class is responsible for a single functionality

- Minimizes the risk of unintended changes

- Improves code maintainability

```
∨ 📁 models
      🔹 Booking.dart
      🔹 DriverProfile.dart
      🔹 Passengers.dart
      🔹 RouteResponse.dart
      🔹 Schedule.dart
      🔹 Trips.dart
```

```
∨ 📁 viewmodels
      🔹 auth_viewmodel.dart
      🔹 commuter_viewmodel.dart
      🔹 driver_viewmodel.dart
      🔹 journey_tracking_viewmodel.dart
      🔹 trip_viewmodel.dart
```

```
∨ 📁 DriverNav
      🔹 account_nav.dart
      🔹 feedback.dart
      🔹 home_nav.dart
      🔹 home_page_driver.dart
      🔹 passenger_details_UI.dart
      🔹 past_trips.dart
      🔹 personal_information.dart
      🔹 ride_nav.dart
      🔹 schedule_nav.dart
      🔹 trip_detail_screen.dart
      🔹 trip_list_UI.dart
      🔹 upcoming_trips.dart
```

Multi-Layered Architecture

Presentation Layer (UI)	trip_list_UI.dart
↓	
App Logic layer	trip_viewmodel.dart
↓	
Data Access Layer	driver_service.dart
↓	
Persistent Data Layer	Trips.dart

Multi-Layered Architecture

Presentation Layer (UI)

trip_list_UI.dart

↓

App Logic layer

trip_viewmodel.dart

↓

Data Access Layer

driver_service.dart

↓

Persistent Data Layer

Trips.dart

Frontend (Flutter)

Architecture: MVVM

UI layer: Handles layout and user interactions.

⬇

ViewModel layer: manages business logic and updating views.

⬇

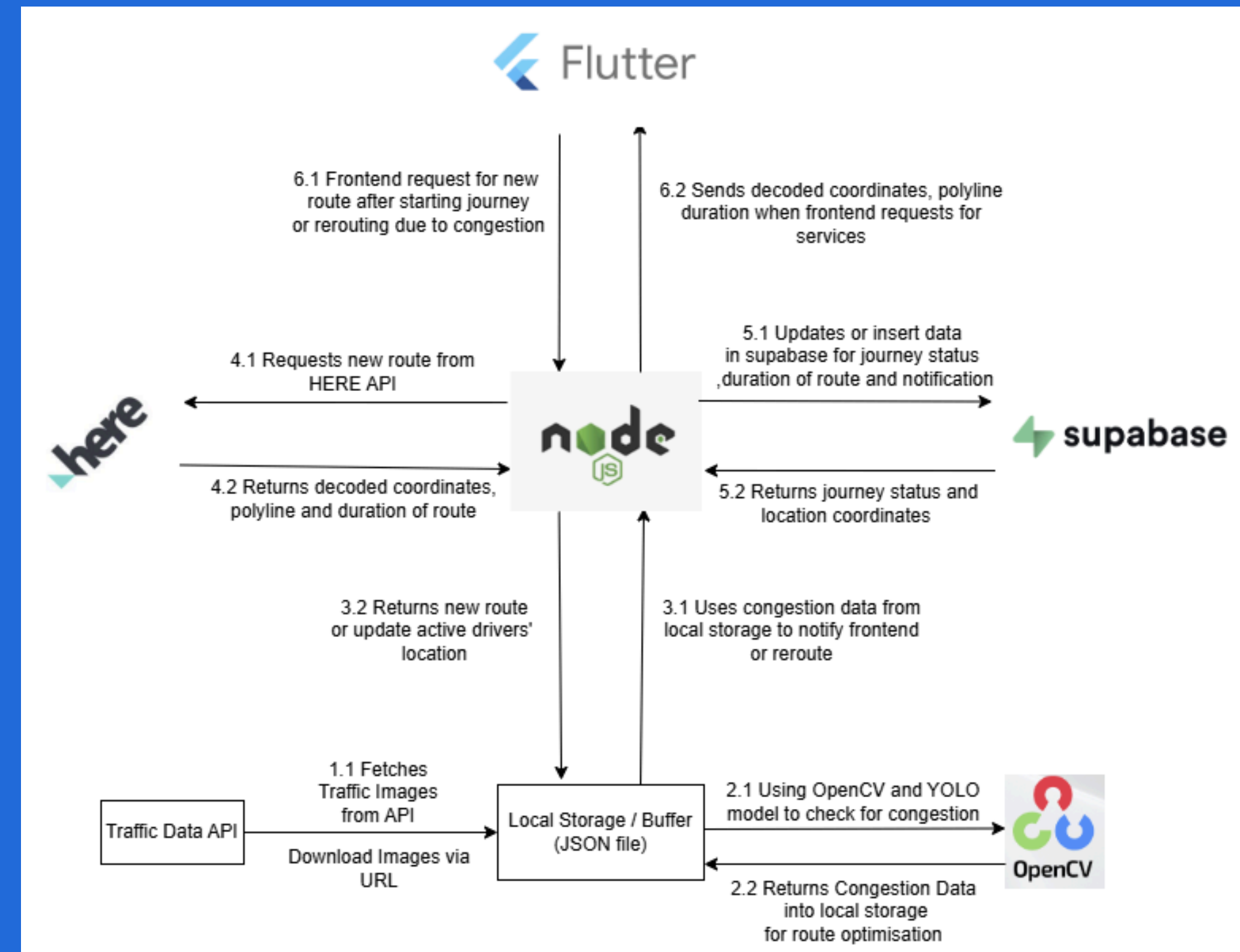**Service layer**: Handle data fetching from or storing into Supabase

Backend

Architecture: MCS

Presentation layer: Handles HTTP requests and responses (routes, APIs)

Service layer: Contains business logic. Includes validation, formatting, filtering, conditional logic

Model layer: Deals with data retrieval or persistence. Interacts with databases, file systems, or external APIs.

Database (Supabase)

bookings: Stores all booking details made by commuters.

buses: Stores information about the buses used for trips, along with driver assignments.

driver_location: Tracks the current location of each driver for real-time updates.

journey: Keeps track of whether a scheduled journey has started or completed.

location: Stores all possible pickup and destination points for the trips.

notifications: Stores notifications related to updates on traffic congestion.

profiles: Stores all the users of the app.

schedules: Stores all the scheduled trips for drivers.

seats: Stores the seat details for each bus

bookings
buses
driver_location
journey
location
notifications
profiles
schedules
seats

Software Engineering Practices

1. Modular Code Structure

2. Efficient State Management

3. Input Validation & Error Handling

Modular Code Structure

- Divided the project into separate layers (models, view, viemodels, services)

- Each function has a single responsibility, improving maintainability
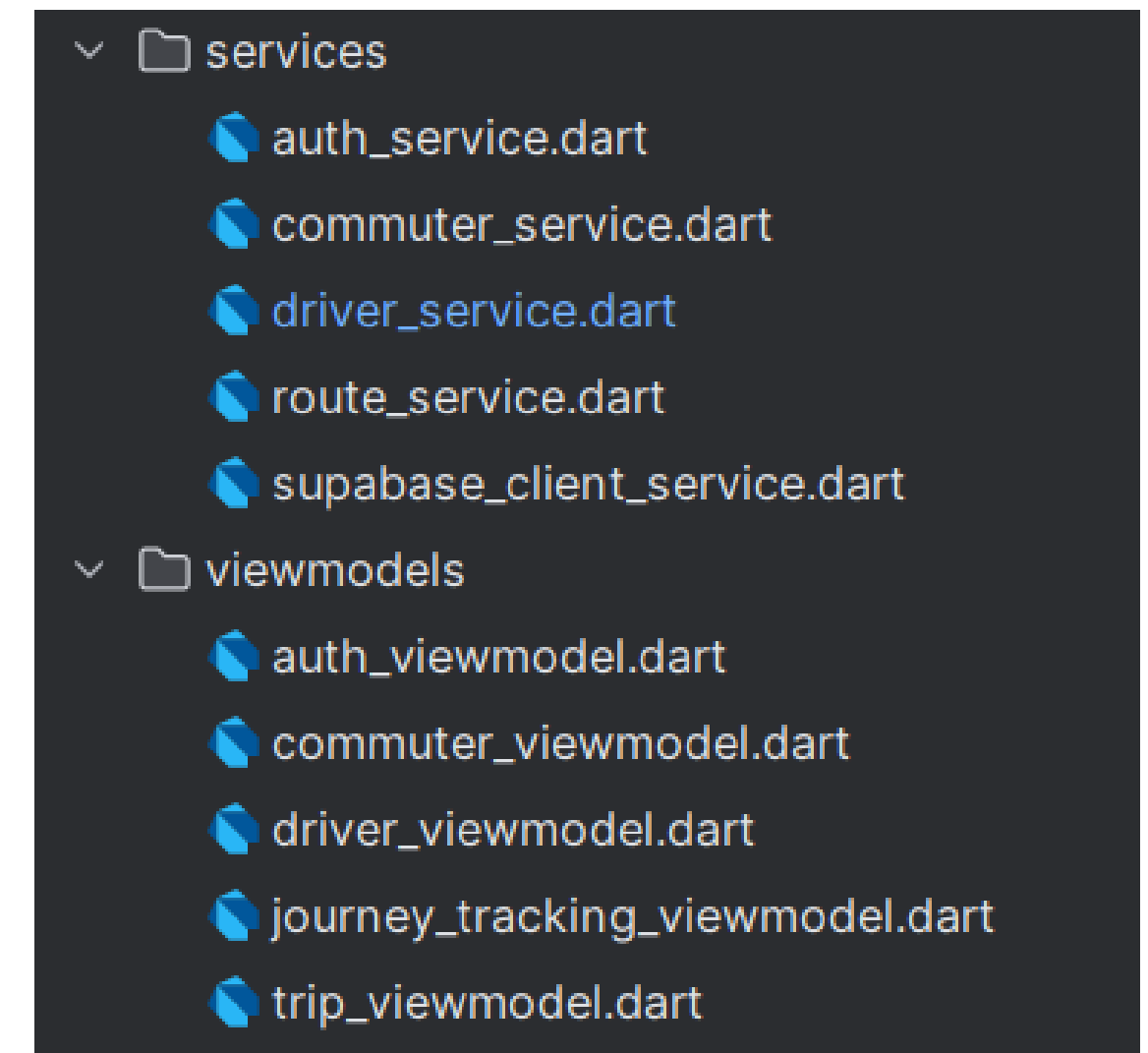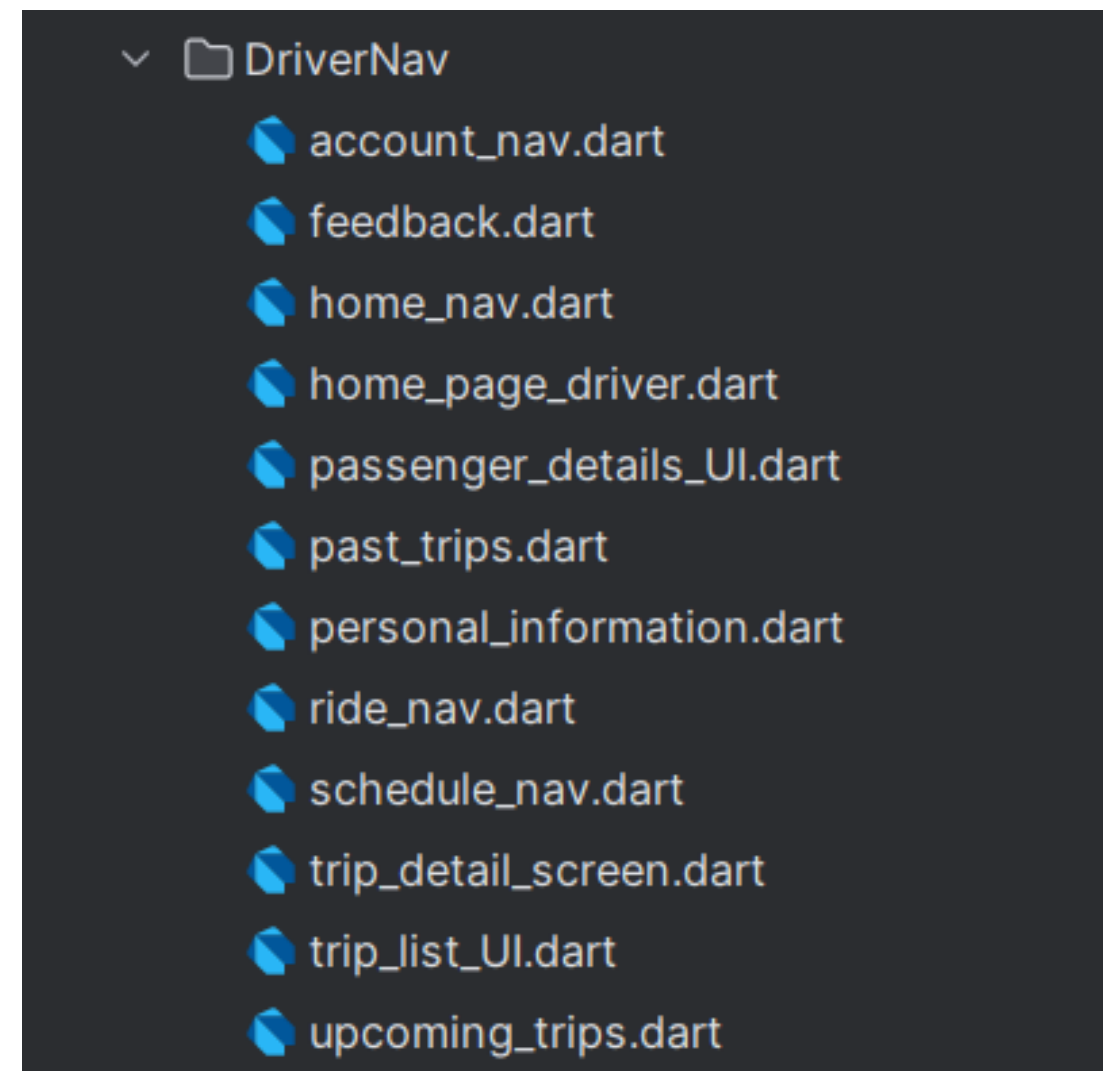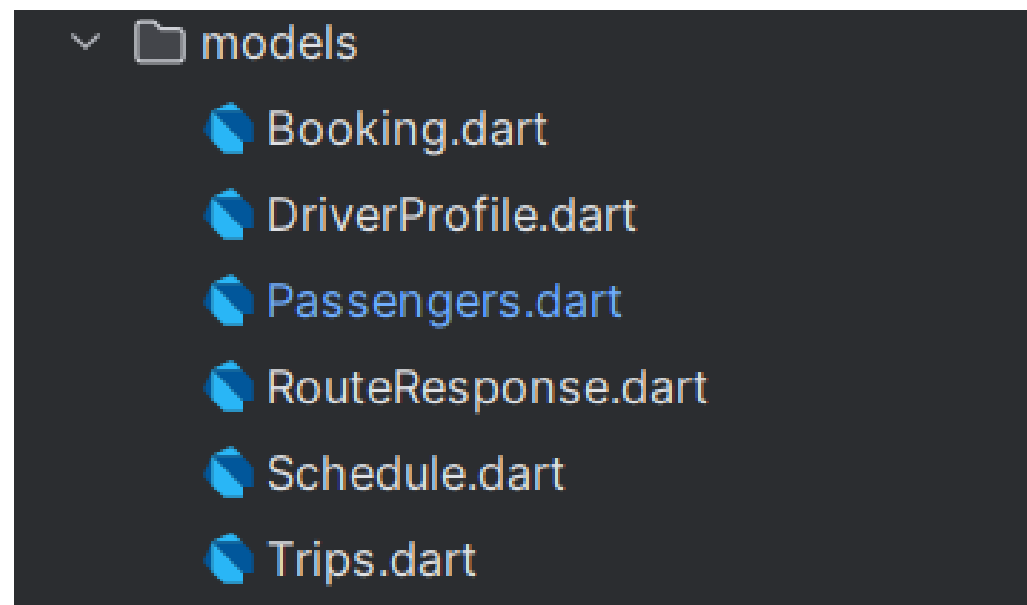
Efficient State Management

- Implemented using **ChangeNotifier** and **Provider**

- Allows the app to reactively rebuild UI when state changes, prevents manual refreshes and improves user experience

```
class TripViewModel extends ChangeNotifier {
```

```
notifyListeners();
```

Input Validation & Error Handling

- Clear and user-friendly error messages via **SnackBar** for invalid input

- Reduces system crashes and ensure systems behaves predictably

```dart
Future<String?> getUserType(String userId) async {
  try {
    final response = await _supabase
        .from('profiles')
        .select('user_type')
        .eq('id', userId)
        .single();
    return response['user_type'] as String?;
  } catch (e) {
    print('Error getting user type: $e');
    return null;
  }
}
```

```dart
Future<Map<String, dynamic>> fetchDriverProfile(String driverId) async {
  try {
    final driverProfile = await _supabase
        .from('profiles')
        .select('username, user_type, created_at')
        .eq('id', driverId)
        .single();
    return driverProfile;

  } catch (e) {
    print('Error fetching driver profile: $e');
    return {};
  }
}
```

How we support new updates?

Separating UI from Business Logic

- **Easier UI changes**: UI updates such as layout changes and new UI elements can be made without affecting the core business logic.

- **Better Maintainability**: Changes to UI can be implemented without disrupting the underlying logic, making the system more maintainable.

How we support new updates?

Input Validation & Error Checking

- **Faster Fixes:** Allows for quicker isolation and resolution of issues as new updates being added might accidentally break existing functionality.

- **Adaptability**: Adding new features becomes easier as the system already has mechanisms to handle edge cases.

Use Case:

Use Case ID:	3.3
Use Case Name:	Create Bus Schedule

Actor:	Driver, Database
Description:	The "Create Bus Schedule" use case is for drivers to indicate to commuters when they are available. It also provides commuters details about the seats available, pickup point, drop off point, and departure time of the scheduled ride.
Preconditions:	The user must be logged in as driver in the systemThe database must be operational
Postconditions:	Database is updated accordinglyThe data on the client displays the newly created bus schedule
Priority:	High
Frequency of Use:	High
Flow of Events:	The driver navigates to the bus schedules screen. The driver enters information such as the pickup point, drop-off point, departure date and time. The driver clicks the "Create" button. The client reflects the newly created bus schedule in the user interface
Special Requirements:	All fields are required
Assumptions:	The layout of seats will be options provided by the system, not for the driver to manually upload

Sequence Diagram:
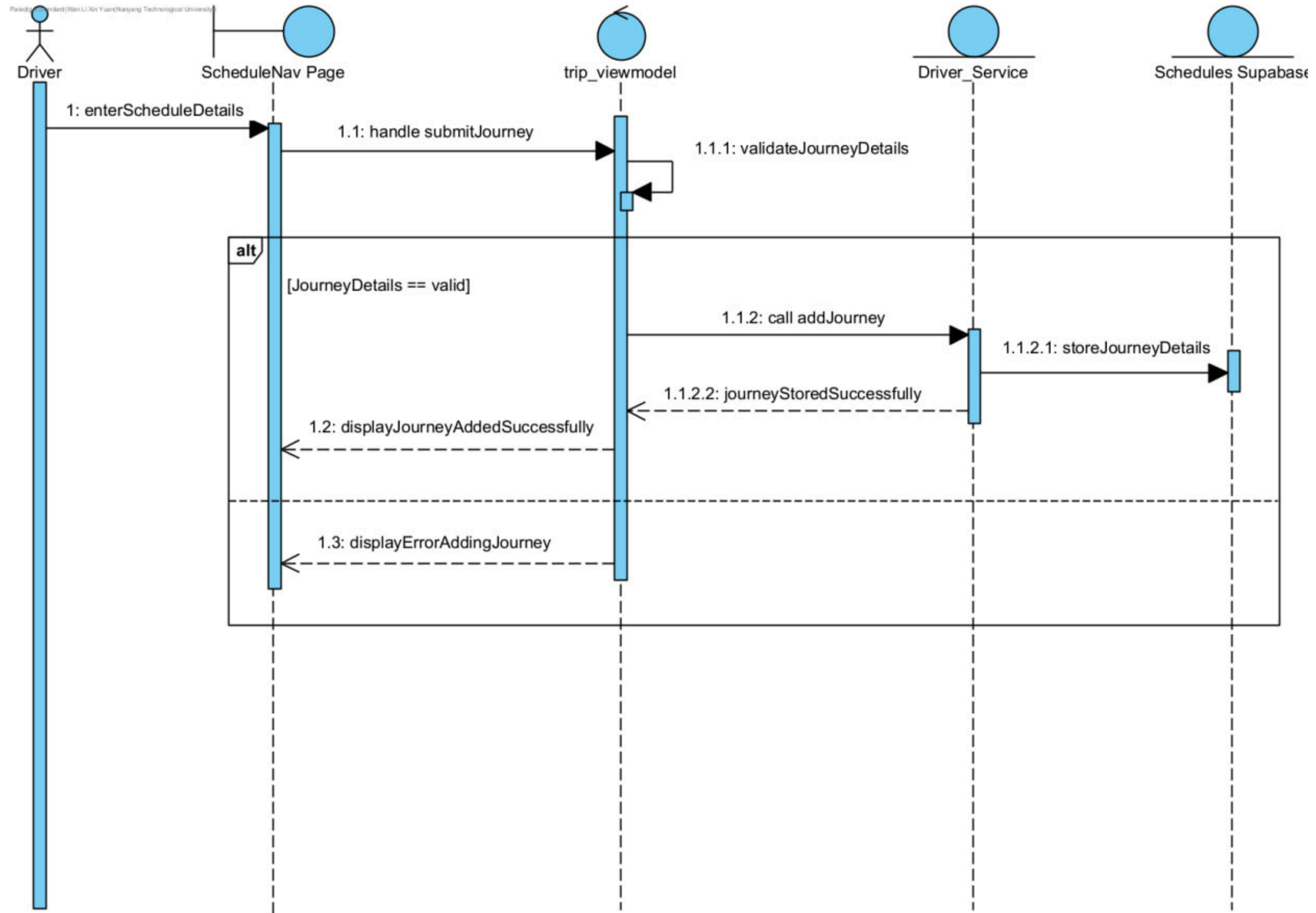
Class Diagram:
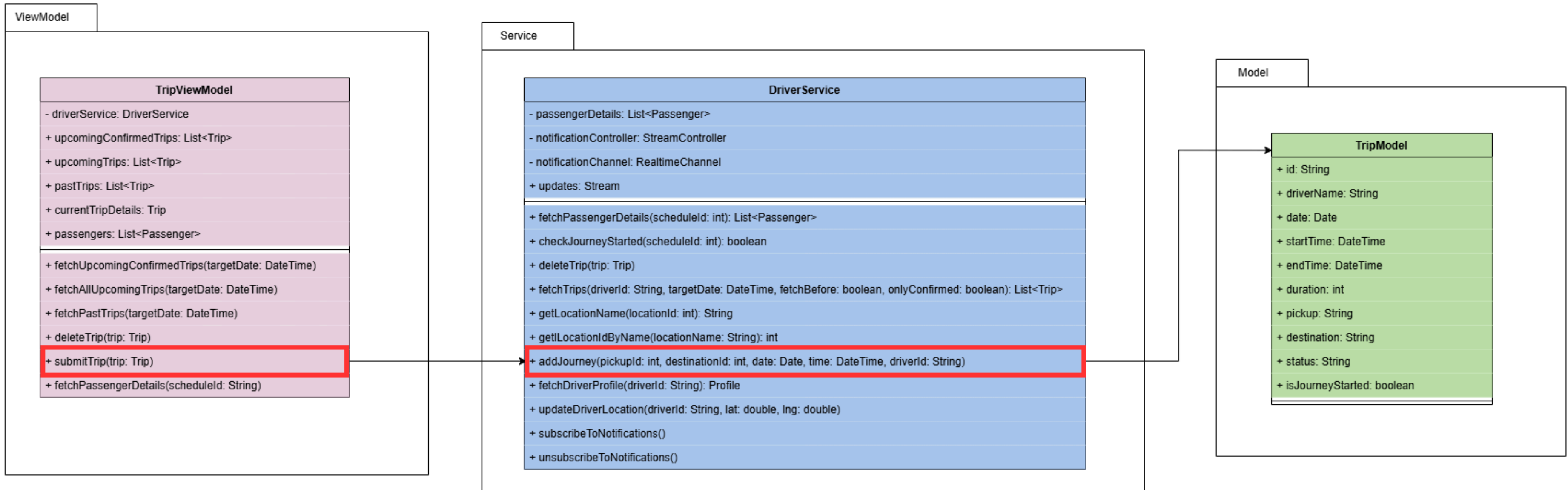
White Box Testing

1. Drivers fail to fill in all the required fields.
2. Drivers enter identical pickup and destination points.
3. Drivers input a date and time that is in the past.
4. Drivers attempt to schedule a new trip where they already have another scheduled trip within 5 hours of the new schedule.
5. Drivers correctly fill in all the fields, providing valid information.

Cyclomatic Complexity (CC) =

|binarydecisionpoint| + 1 = 5 + 1 = 6

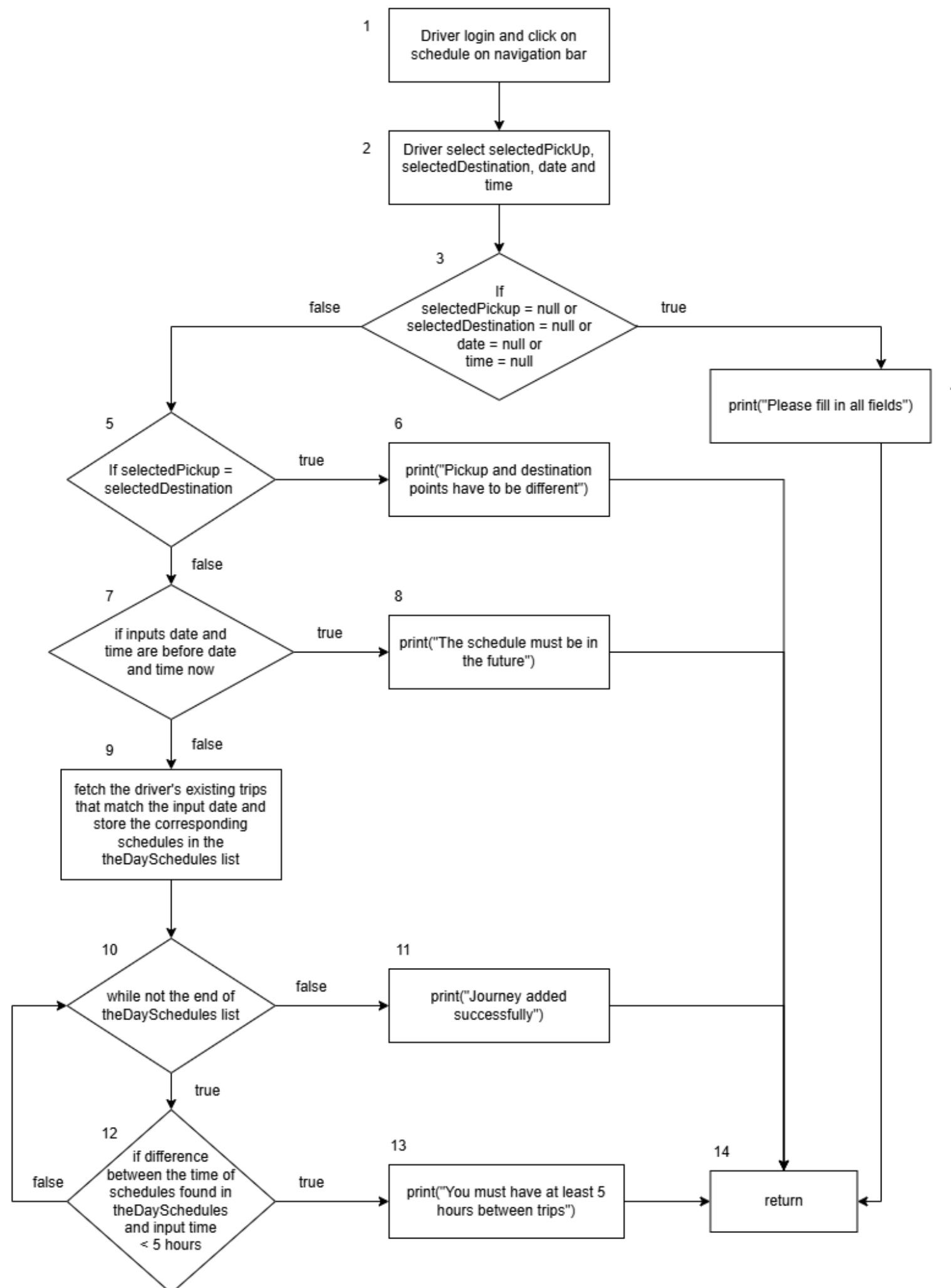Basis Path #1: 1, 2, 3, 5, 7, 9, 10, 11, 14

Basis Path #2: 1, 2, 3, 4, 14

Basis Path #3: 1, 2, 3, 5, 6, 14

Basis Path #4: 1, 2, 3, 5, 7, 8, 14

Basis Path #5: 1, 2, 3, 5, 7, 9, 10, 12, 10, 11, 14

Basis Path #6: 1, 2, 3, 5, 7, 9, 10, 12, 13, 14

Testing Result

Test Case Name	Test Input	Expected Output	Actual Output	Test Result
Submit-01	selectedPickUp = NTU selectedDestination = Punggol date = 2025-05-31 time = 18:00	Success Message: "Journey added successfully"	Success Message: "Journey added successfully"	Pass
Submit-02	selectedPickUp = null selectedDestination = Punggol date = 2025-05-31 time = 18:00	Error Message: "Please fill in all fields"	Error Message: "Please fill in all fields"	Pass
Submit-03	selectedPickUp = NTU selectedDestination = NTU date = 2025-05-31 time = 18:00	Error Message: "Pickup and destination points have to be different"	Error Message: "Pickup and destination points have to be different"	Pass

Testing Result

Test Case Name	Test Input	Expected Output	Actual Output	Test Result
Submit-04	selectedPickUp = NTU selectedDestination = Punggol date = 2025-01-01 time = 18:00	Error Message: "The schedule must be in the future"	Error Message: "The schedule must be in the future"	Pass
Submit-05	selectedPickUp = NTU selectedDestination = Punggol date = 2025-05-31 time = 10:00	Success Message: "Journey added successfully"	Success Message: "Journey added successfully"	Pass
Submit-06	selectedPickUp = NTU selectedDestination = Punggol date = 2025-05-31 time = 14:00	Error Message: "You must have at least 5 hours between trips"	Error Message: "You must have at least 5 hours between trips"	Pass

Thank You