MSc in Enterprise Application Development

SE5060 – Enterprise Software Analysis & Design Implementation Walkthrough Assignment

SmartSL

Ticketing System

Group Details (Group 05):

- **R.P.D. Lenora - MS21929328**
- **Perera W.S.S – MS21930904**
- **L.R.S. Subasinghe - MS21929250**
- **M.B.K.Hasarinda – MS21930690**

Table of Contents

Assumptions 3
Justifications for the modified class diagram 10
Modified Sequence Diagram 14
1. User SmartCard Management 14
2. Schedule Bus Timetable 15
Web 25
Login 25
Home 26
Passengers 27
Manage vehicles 28
Schedule Timetable 29
Design Patterns 31
1. Singleton 31
2. Facade 32
Using Facade supplies the client with an interface via which the client may access the system 32
3. Strategy 35
4. Factory 37

Smart SL is a bus ticketing system with both mobile and web applications. Passengers, Drivers and Admins use the mobile application and Transport Manager, Admin, Ticket Inspector uses the Web application. All the Passengers must register with the application at first in order to use the system. After a successful registration passengers can activate the card with the QR code. The passengers can see the available bus timetables.

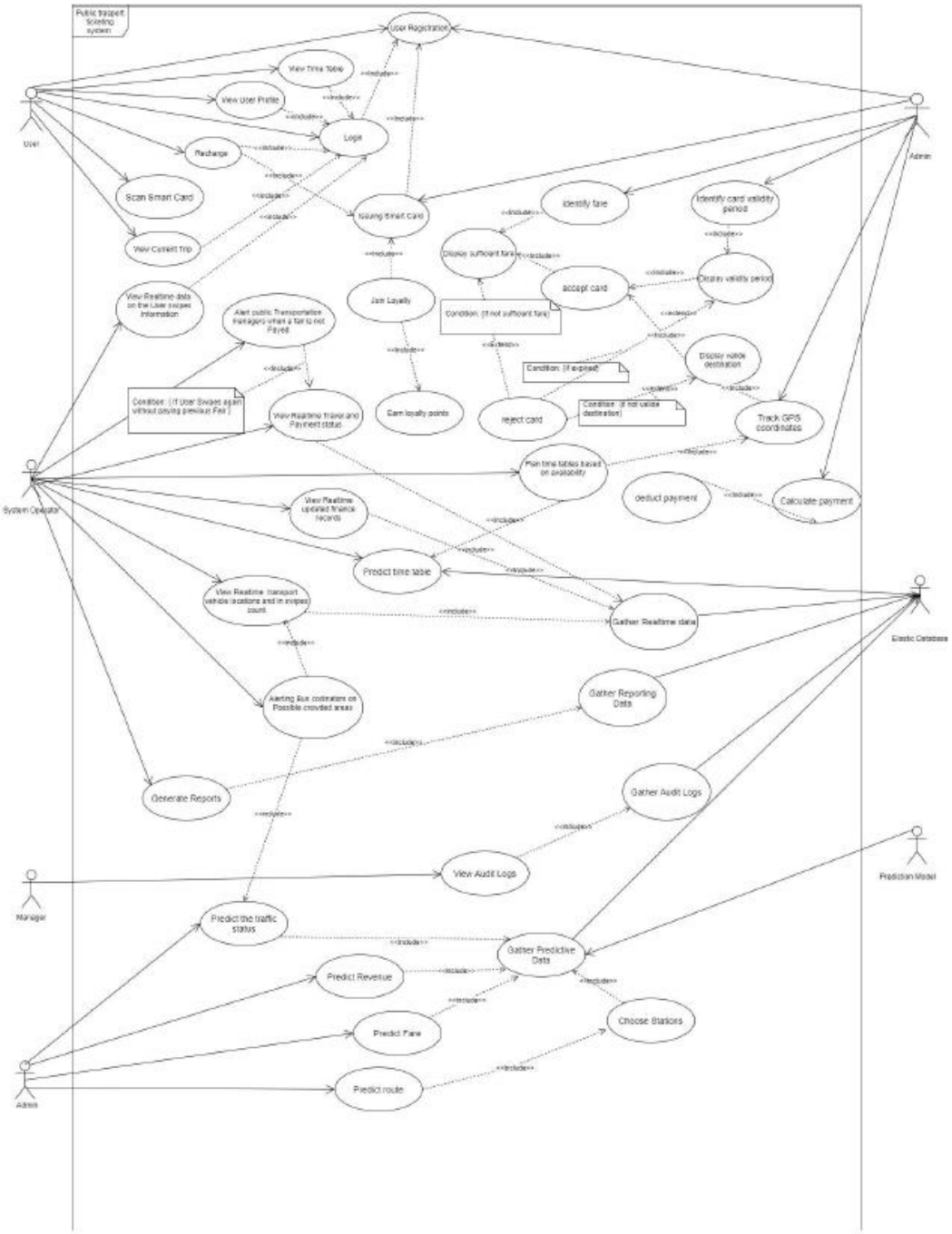
Passengers can scan the card when they are getting into a bus and when getting out of the bus at the destination. At the starting point, the reader scans the QR code and reads the ID number and location using GPS. After that the system will check the account balance. If there is no sufficient balance, system will return a notification.
The Ticket Inspectors can also check that passenger list to see whether they have properly used their token for the journey. If the notification displayed is 'Not sufficient account balance, the passenger must recharge the account.

There is an Admin Portal website for the Transport Managers. This website (admin portal) is mainly handled by the Transport Manager. Bus details are added by the Transport Manager, and he can check the availability of the buses and schedule the timetables. And also he can add and update the bus unit charges. Public transport manager can plan the finances using the information from the website. The website generates statistical reports to check the income.

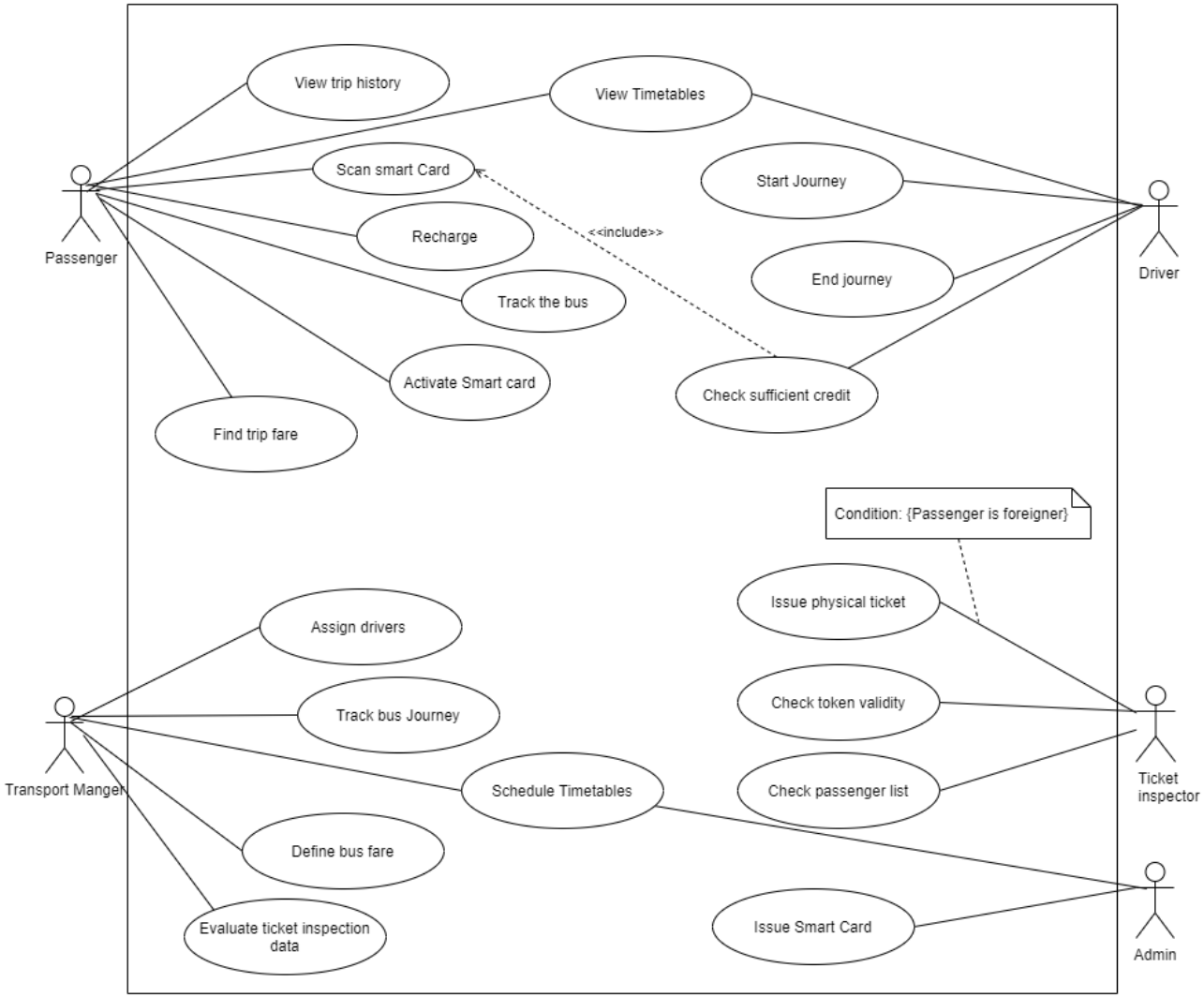
Assumptions

- Scan the starting point and ending point using a GPS tracker when Checking-in and Checking-out from the bus.
- Foreigner is issued a physical ticket by the ticket inspector for cash.

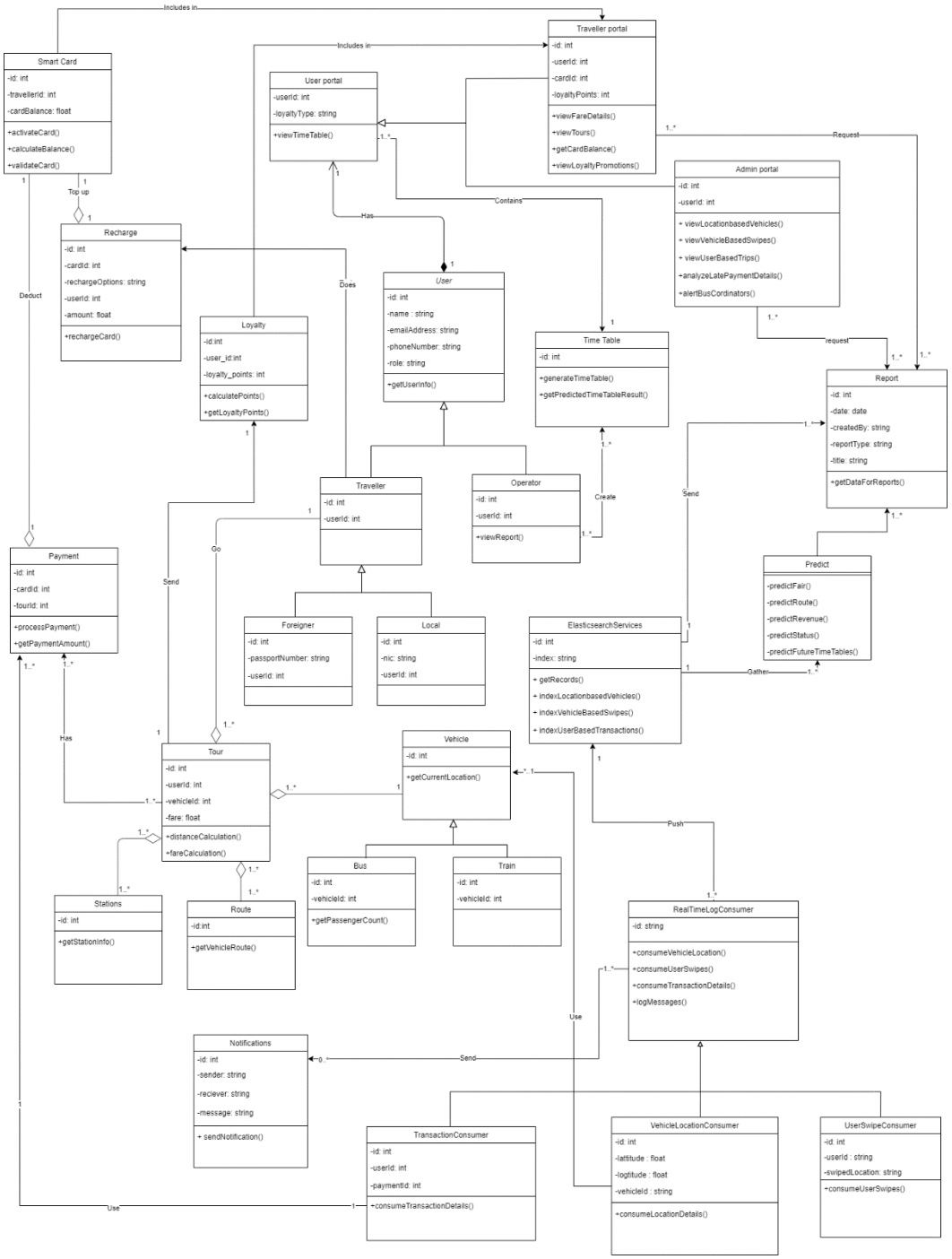
Previous Use Case Diagram



New Use Case Diagram



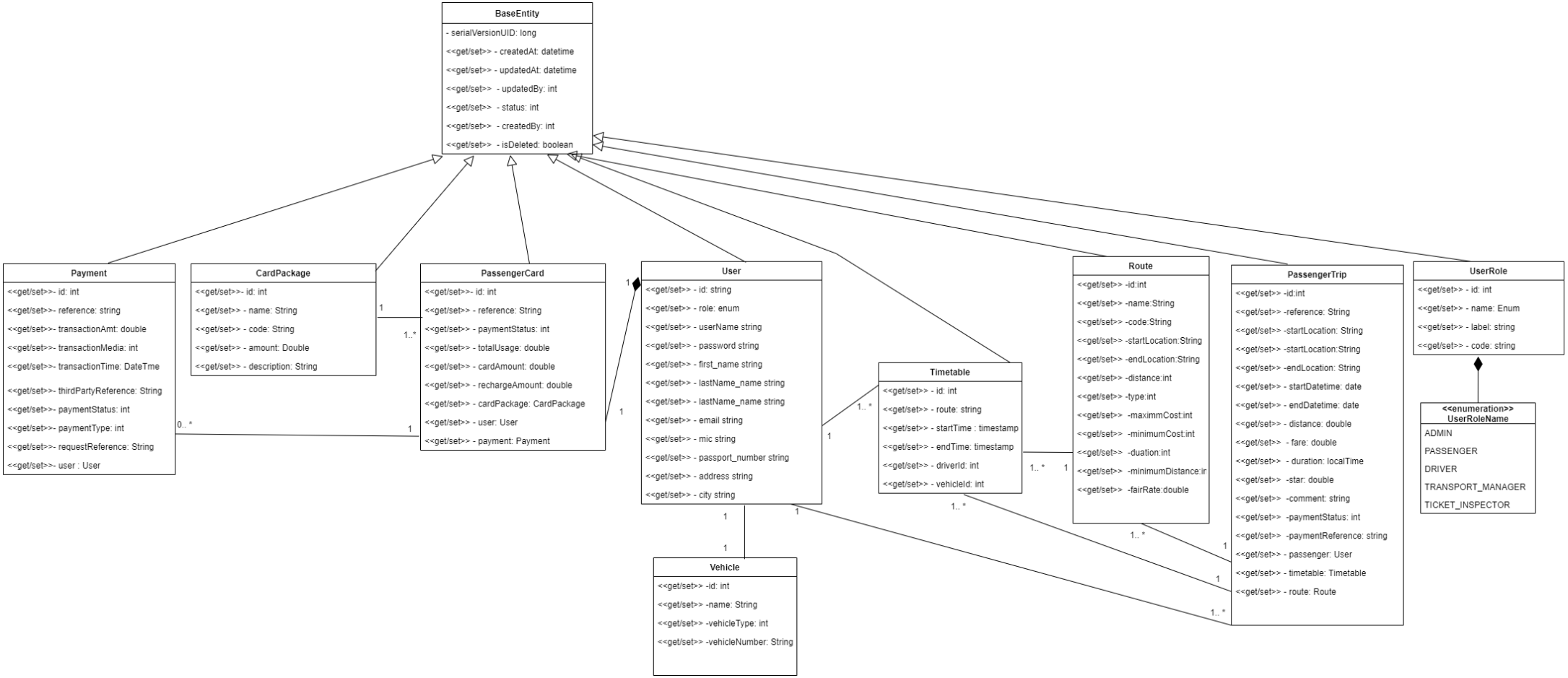
#	Actor	Modification	Justification	Status
1	Transport Manager, Ticket Manager	New Actors introduced	General terms used in previous use case diagram	Added
2	Passenger	User actor modified in to Passenger	User is ambiguous and common for all system users such as Passenger, Driver and Administrator	Added
3	Passenger	Remove trip check facility	Location is automatically detected when scanning the card during start and end of the journey	Removed
4	Passenger	Include check card balance to scan smart card	Need to check card balance when scanning the smart card	Added
5	Passenger	Can View the trip history		Added
6	Transport Manager/Admin	Schedule timetables	Add, View, Search timetables	Added
7	Transport Manager	Evaluate ticket inspection detail	Evaluate ticket inspection details taken from the ticket inspectors about the passenger token validity	Added
8	Transport Manager	Assign drivers to buses		
9	Transport Manager	Track bus trips		
10	Transport Manager	Define bus fare		
11	Driver	Start and end journey to track the bus		
12	Driver	Check passenger credit card balance	Check sufficient card credit of the passengers to permit the journey	
13	Driver	View timetables		
14	Ticket Inspector	Token validity of passenger		Added
15	Ticket Inspector	View Passenger list		
16	Ticket Inspector	Issue Physical Ticket foreigner		
17	Admin	Issue smart cards		



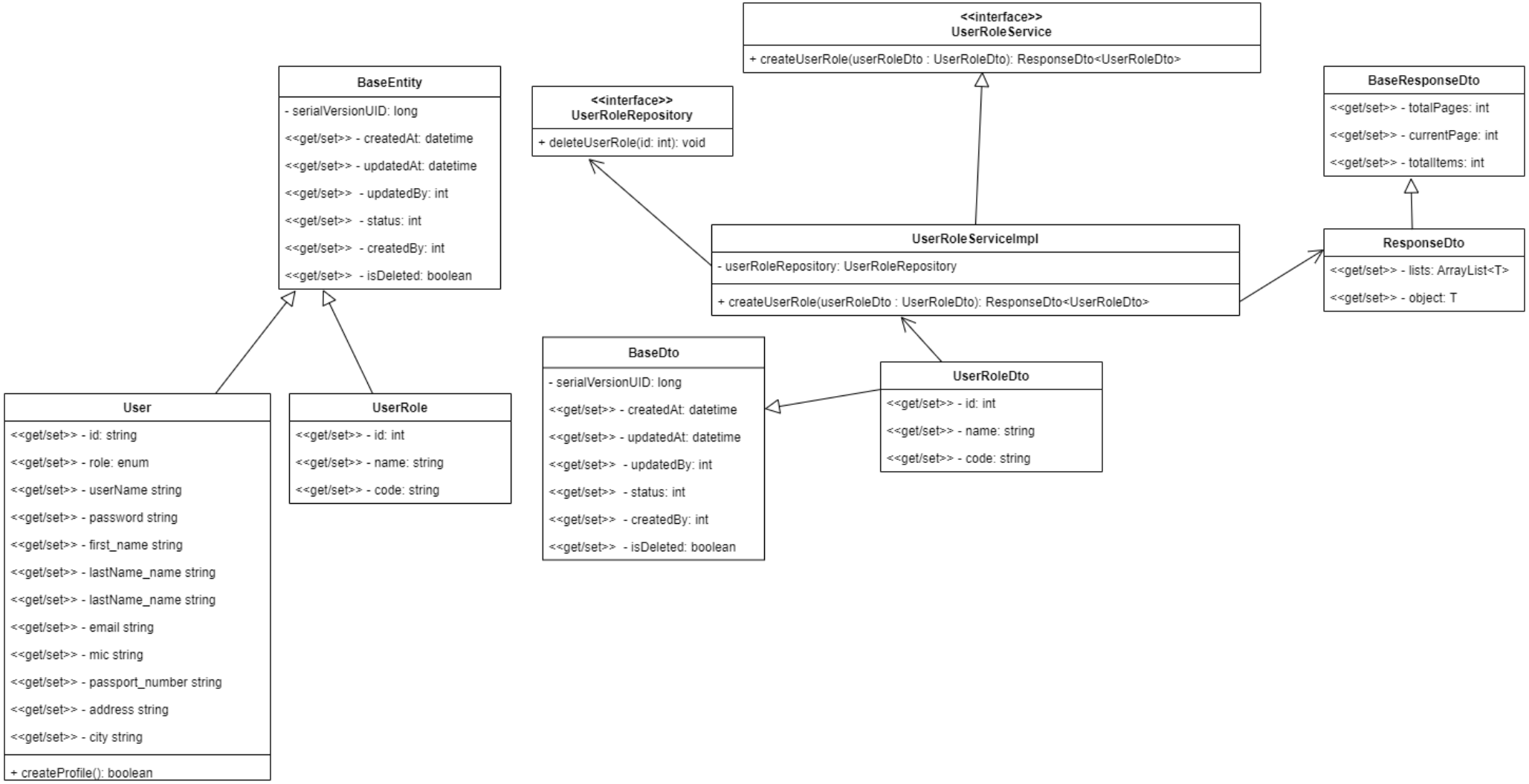
Previous Class Diagram

- User portal and all users are added into user and user role service layer
- Only bus is implemented so a common class vehicle is used.
- loyalty is not implemented if we want, we can add it for future enhancements.
- Predict classes are removed

Modified Class Diagram



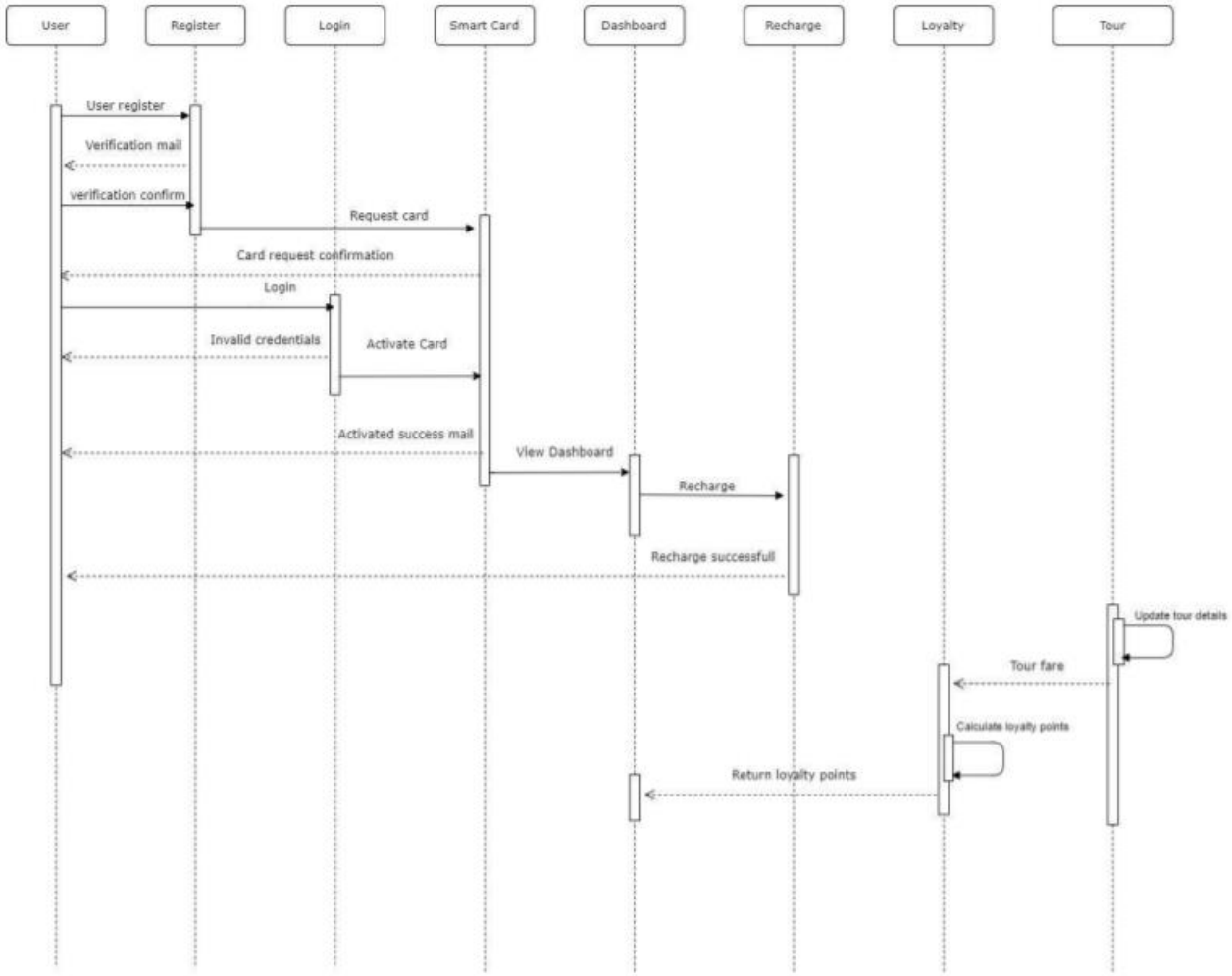
Service layer

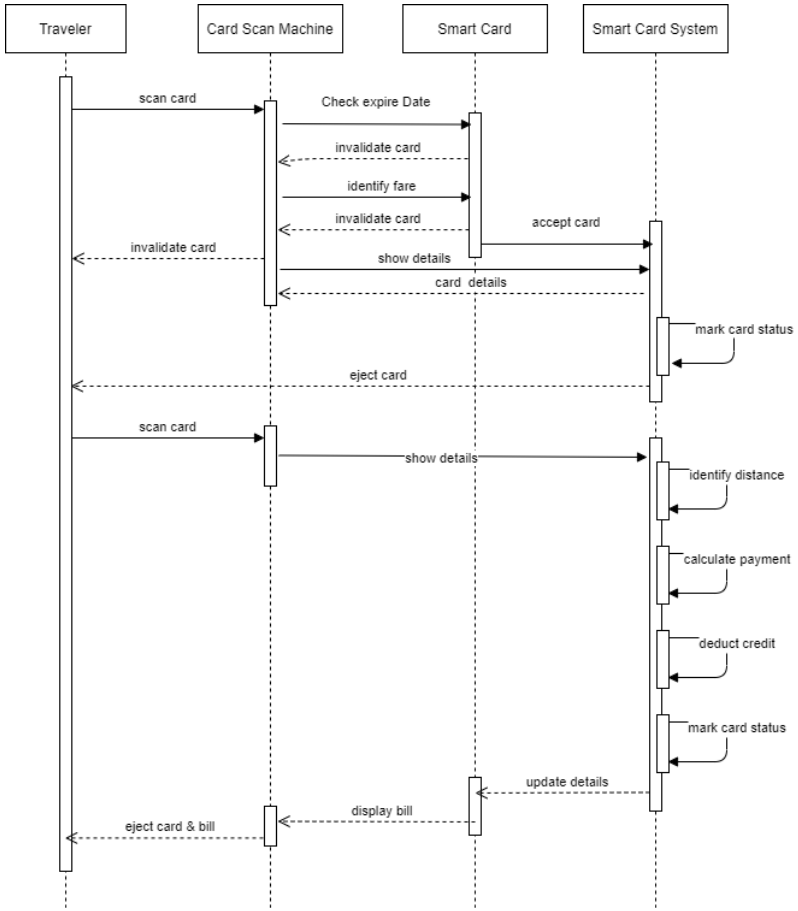


Justifications for the modified class diagram

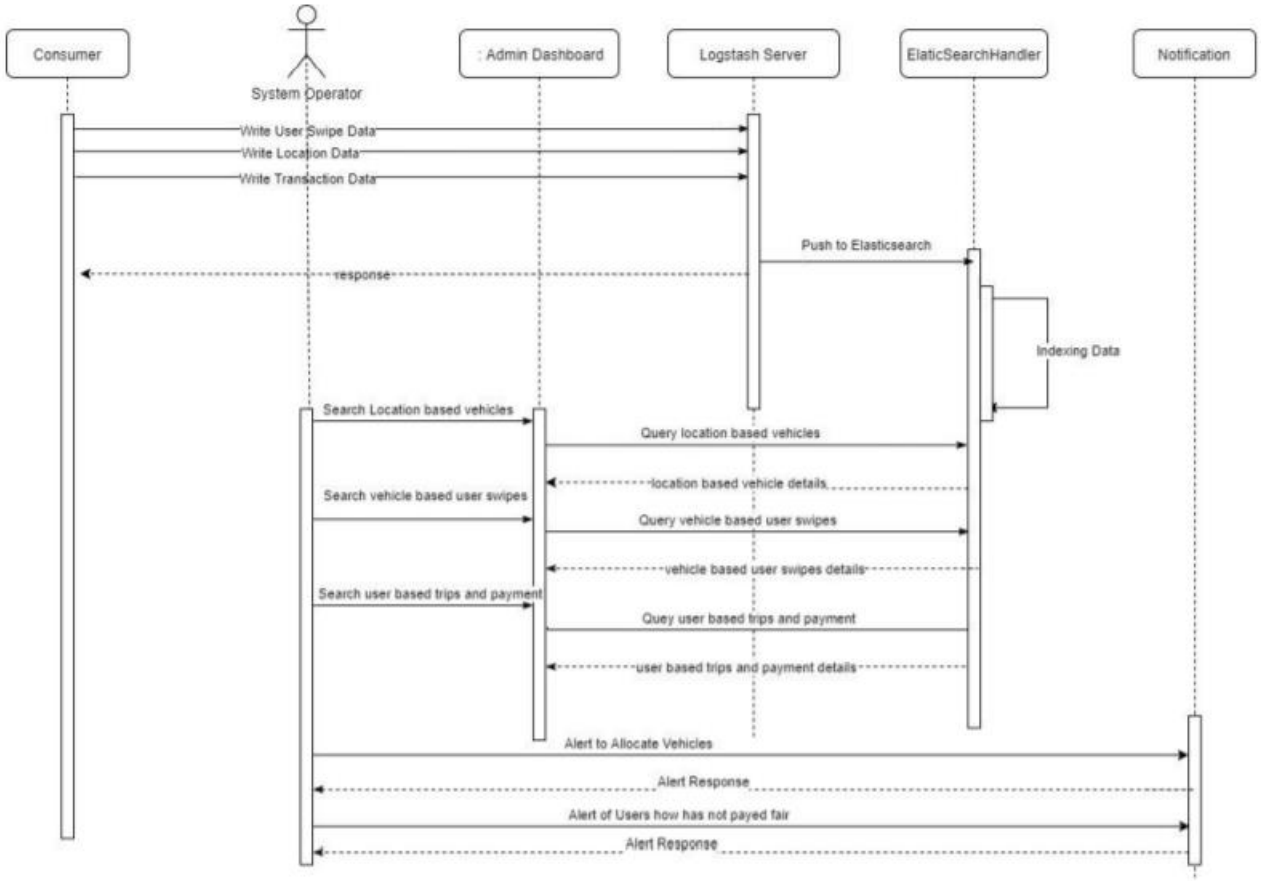
- Assumptions of Relationships
  - User Passenger Card - User is composed of PassengerCard classes. One user has one card. 1-1.
    Ex :When user gets deleted, its parts which is the card here gets deleted.
  - Passengercard CardPackage - PassengerCard can have one Card package. Many
    Card Packages - Amount is defined, according to the amount only we can top up/recharge
  - Passengercard & Payment
    i. In account activation there are no payments for Passengercard 1 – 0..*
  - User & Timetable
    i. Transport Manager, Driver, has 1 or many timetables
  - Route Timetable – One route can have many timetables

Previous sequence diagram



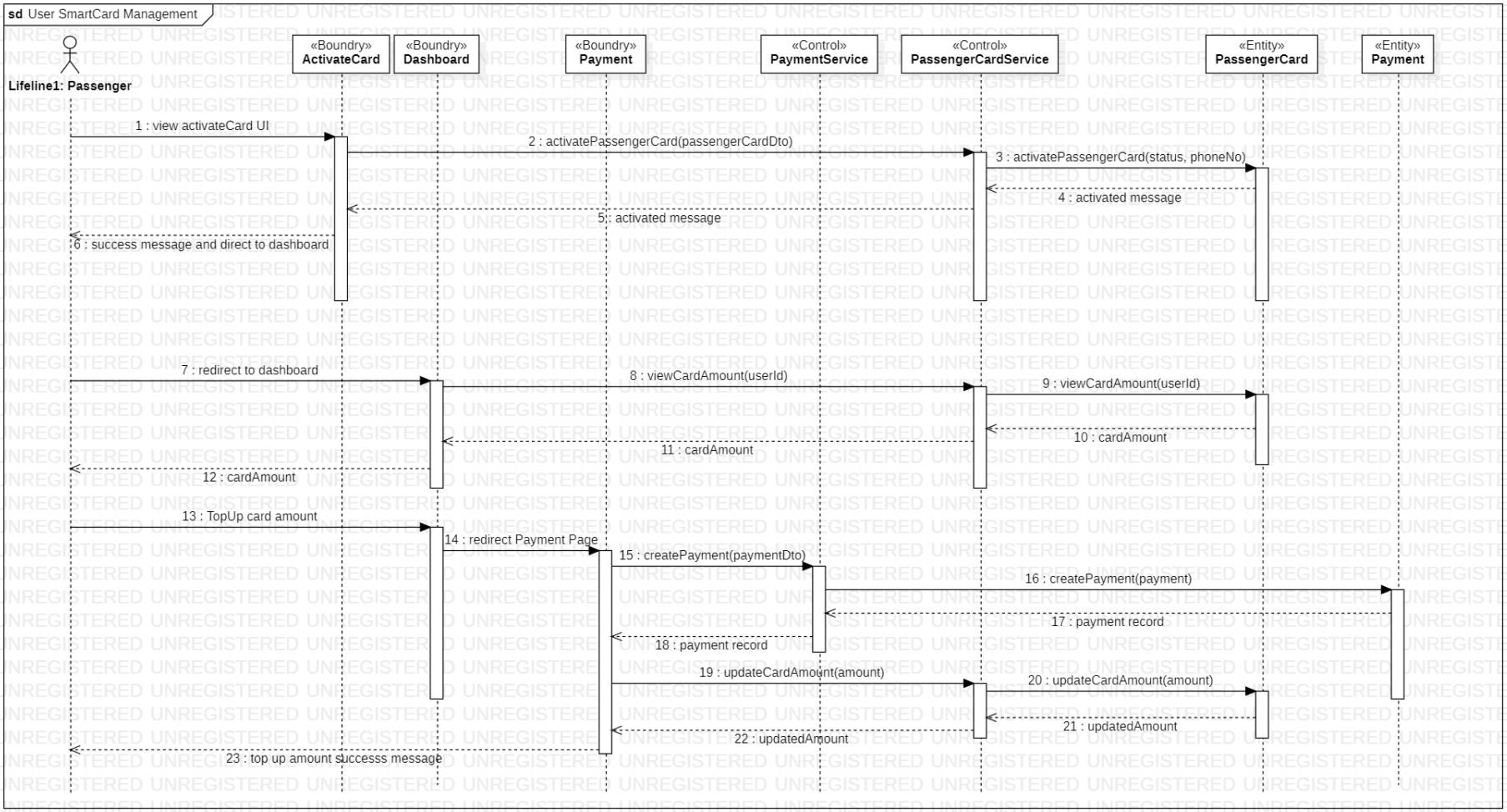

Traveler
Card Scan Machine
Smart Card
Smart Card System
scan card
Check expire Date
invalidate card
identify fare
invalidate card
accept card
invalidate card
show details
card details
mark card status
eject card
scan card
show details
identify distance
calculate payment
deduct credit
mark card status
update details
display bill
eject card & bill

- Real Time Information Management

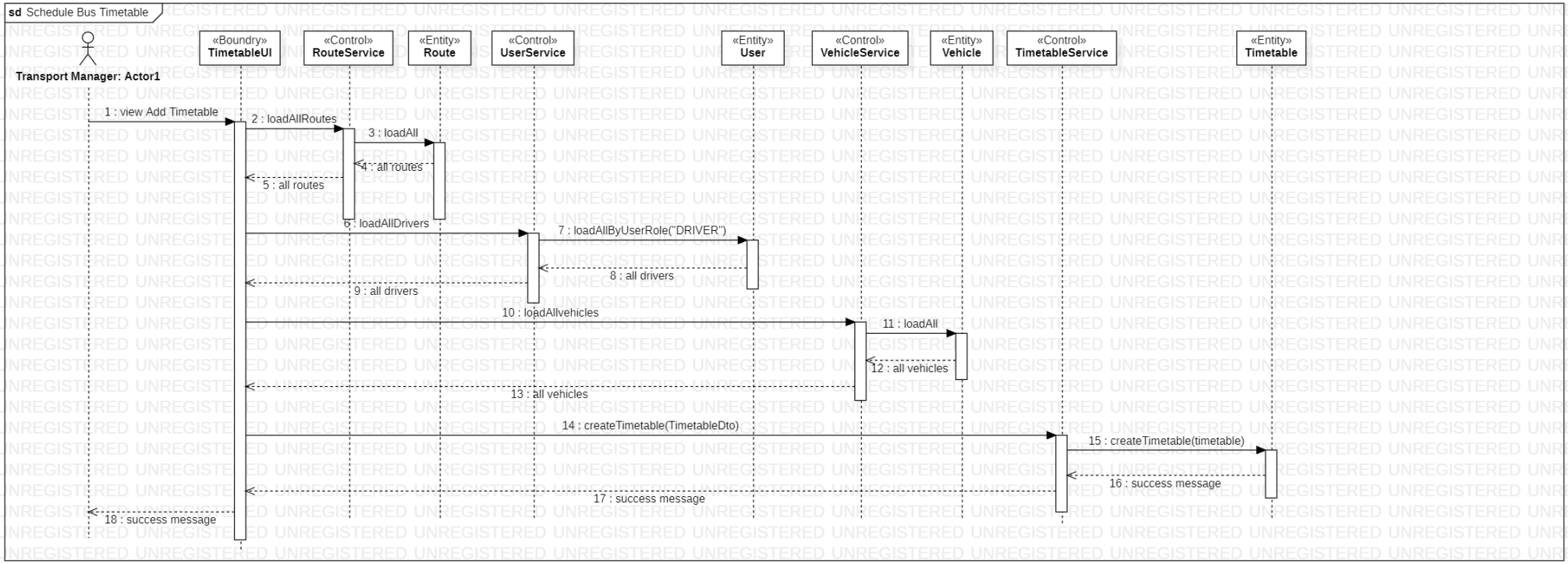


Modified Sequence Diagram

1. User SmartCard Management

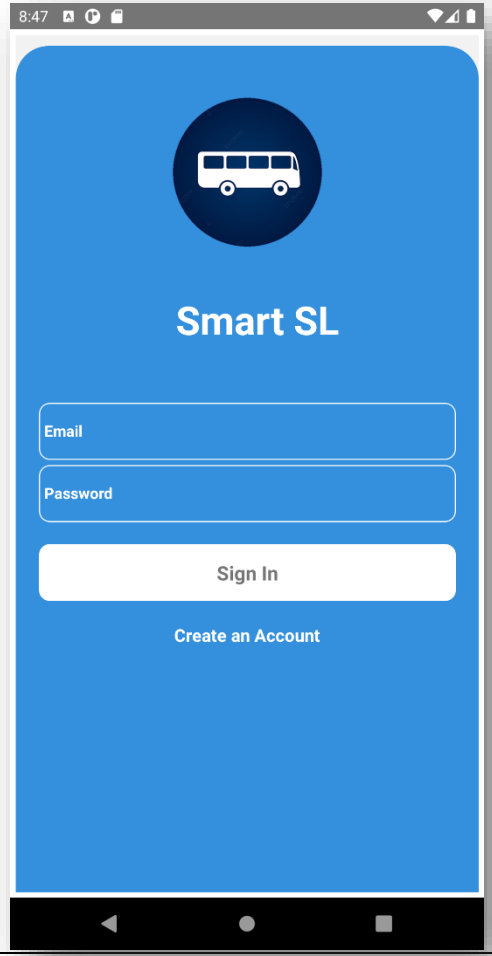


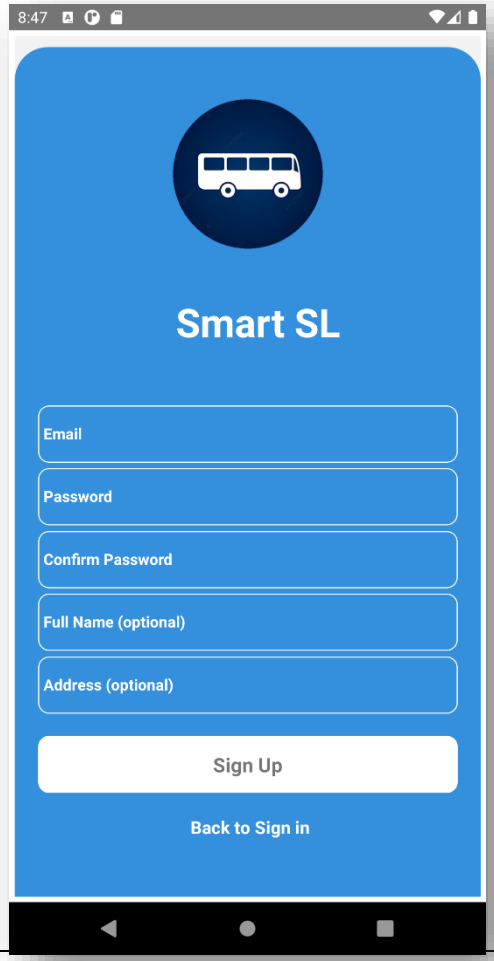
2. Schedule Bus Timetable

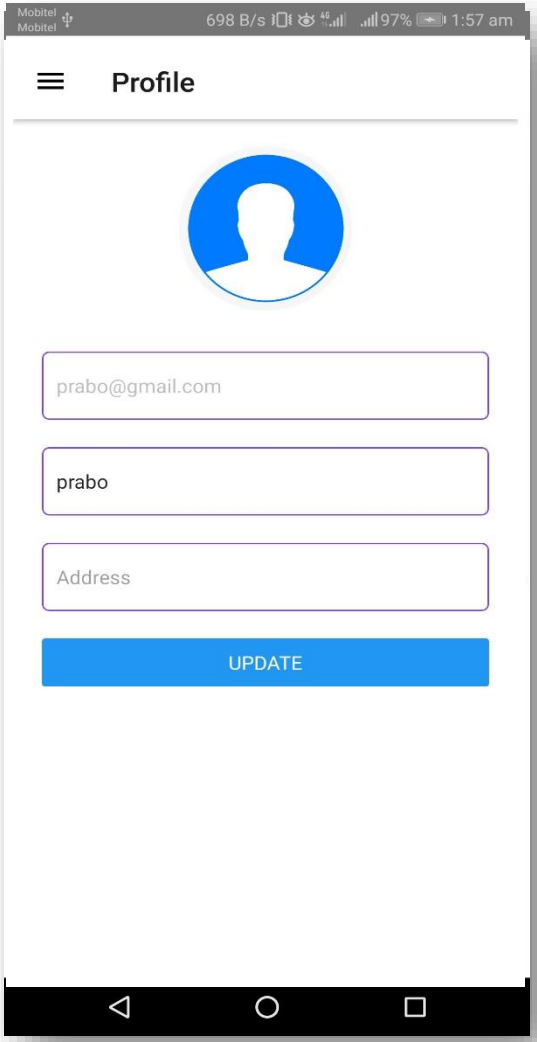


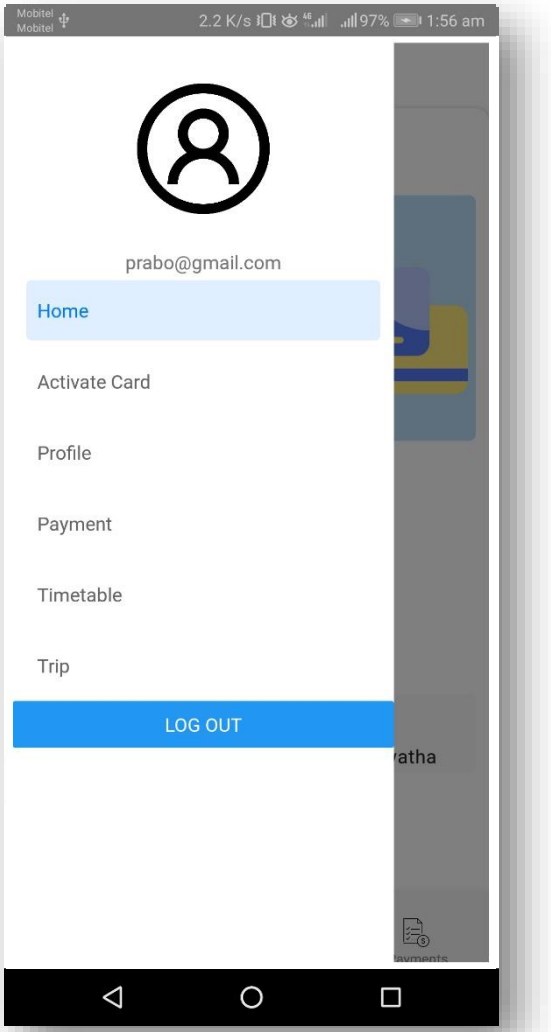
High Fidelity Diagrams with Annotation

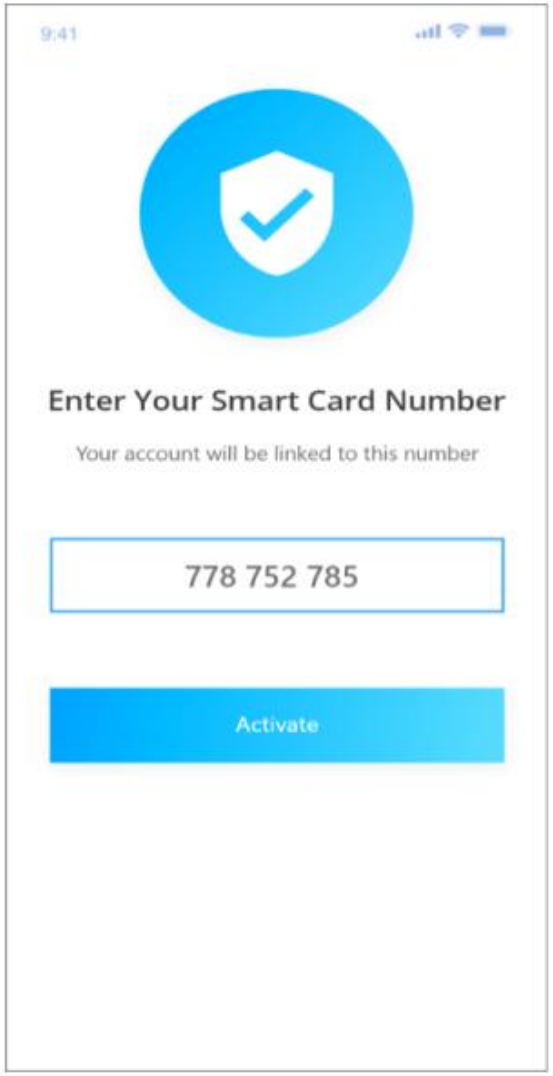
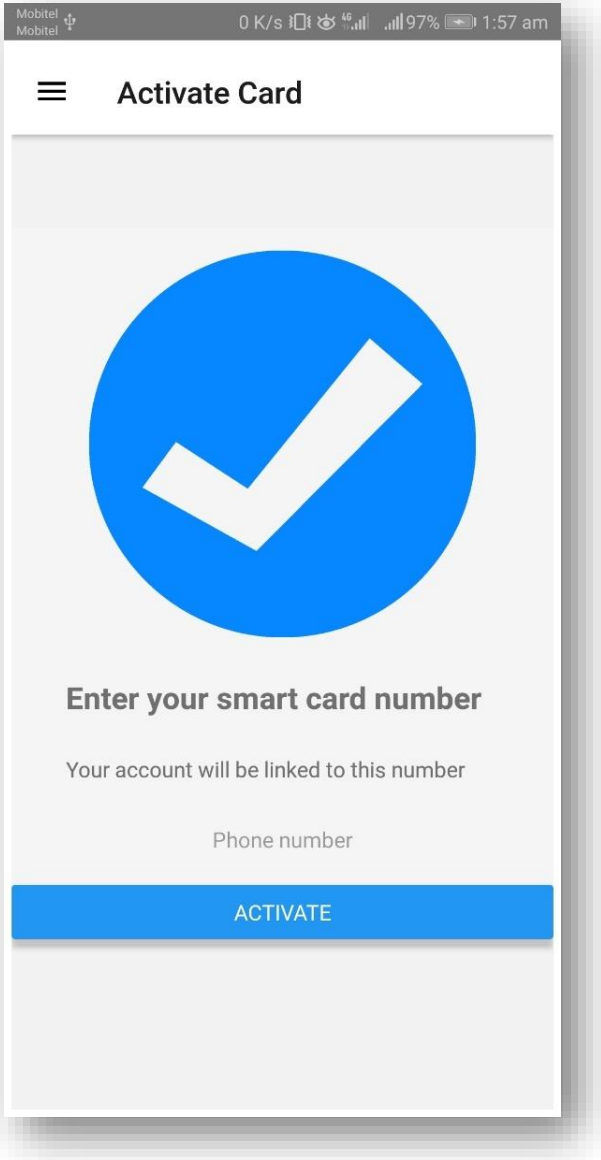
UI justification

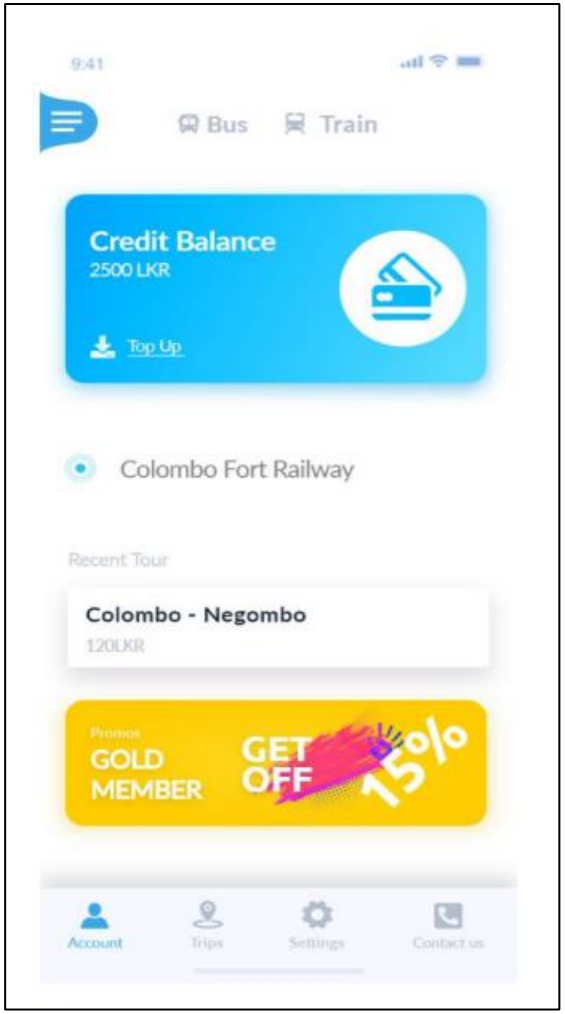
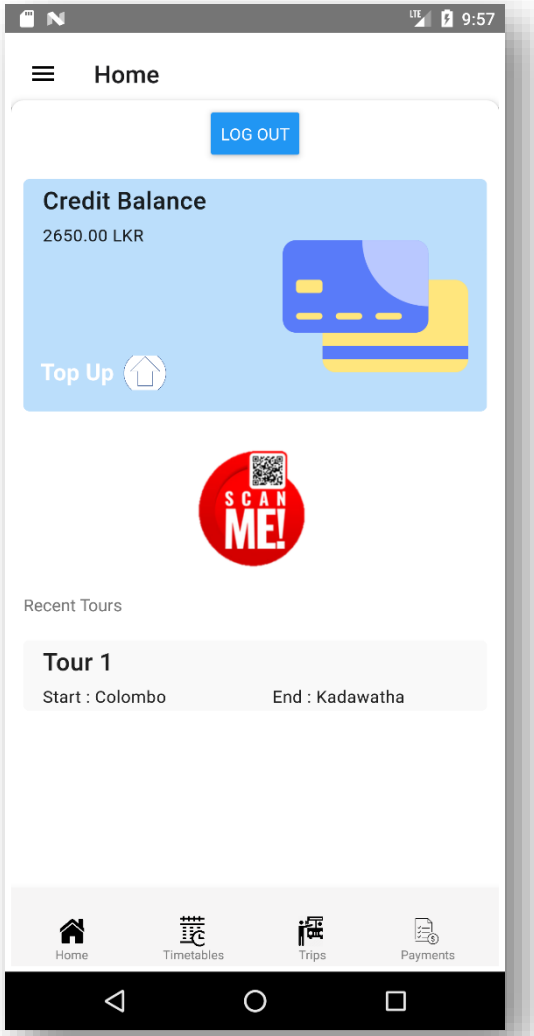
#	Previous UI	New UI	Justification	Status
1	Transport Manager, Ticket Manager		General UI for login	Added

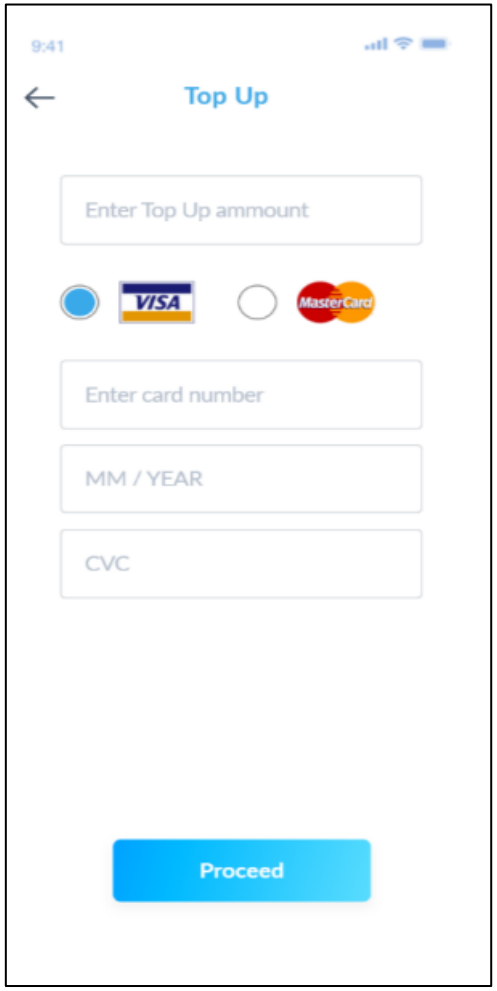
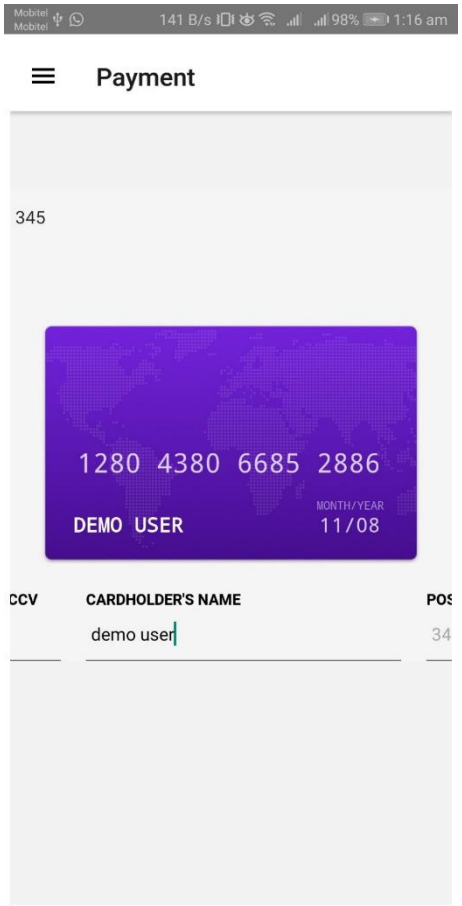
2			Create Account	

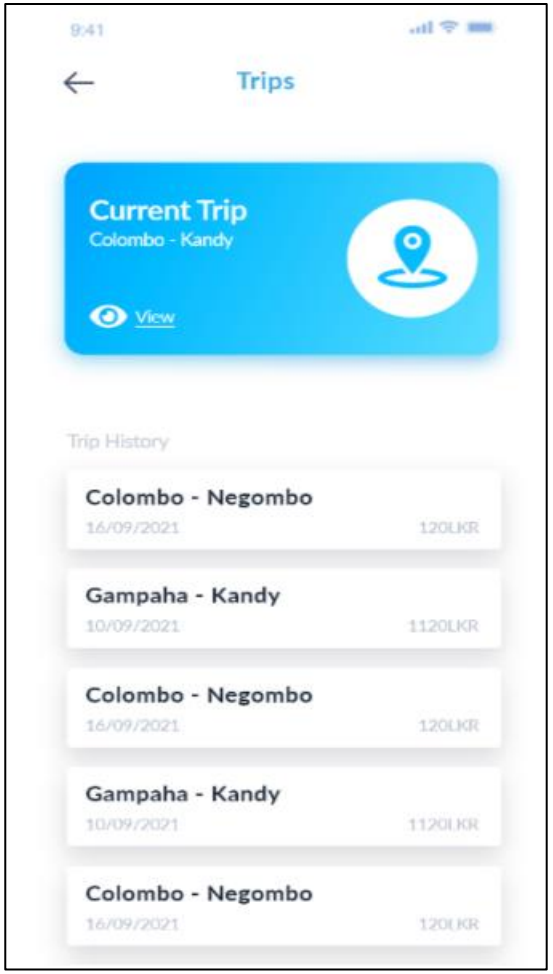
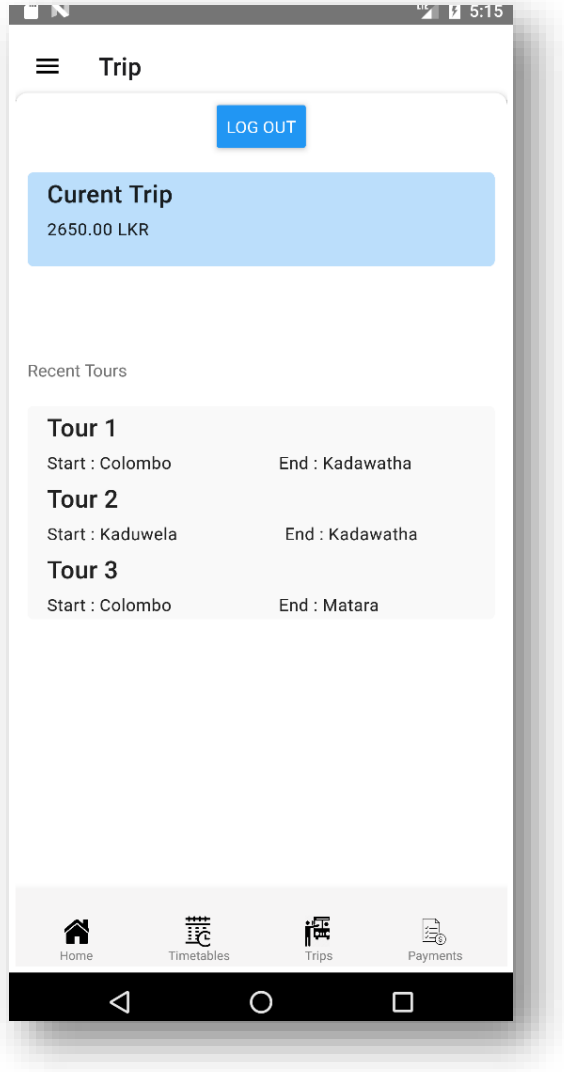
3			User profile	

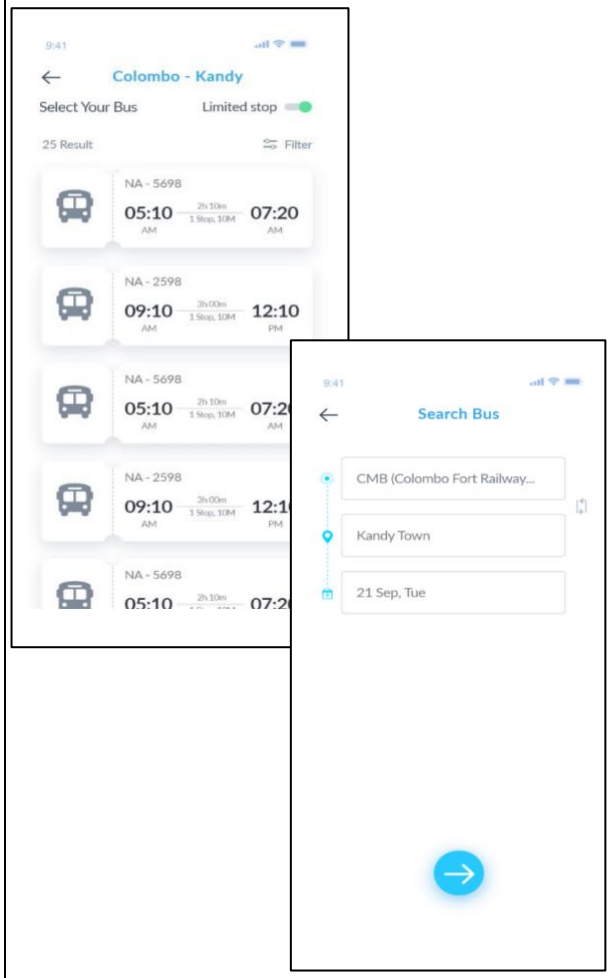
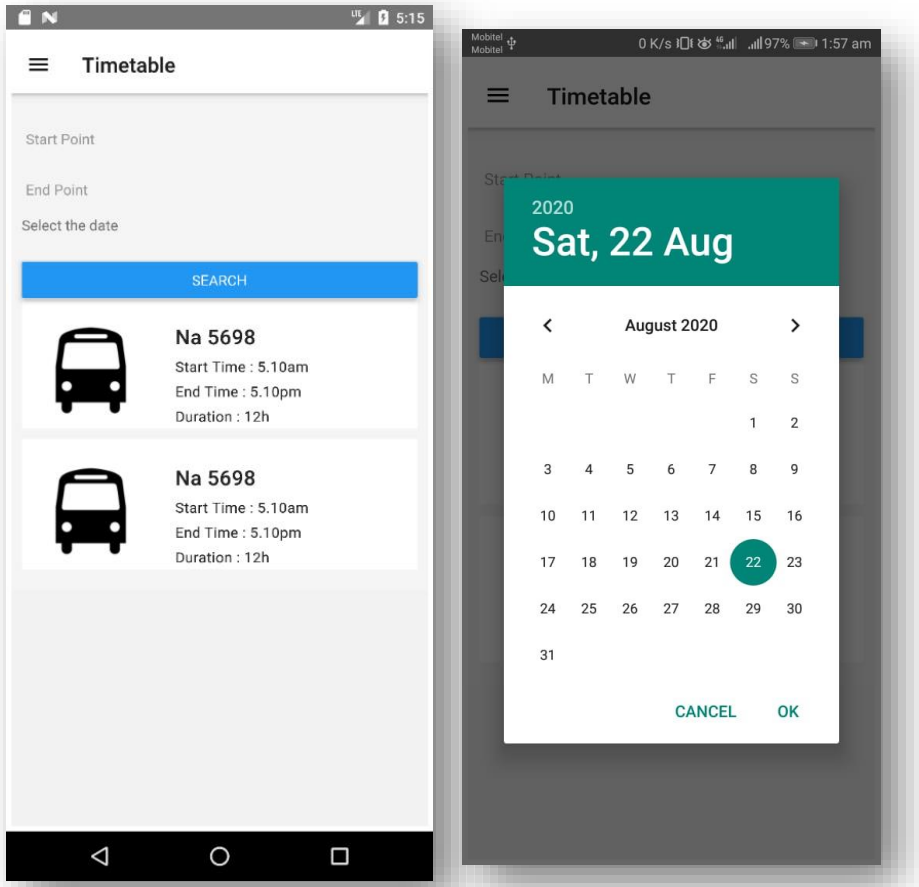
		 Mobitel Mobitel 2.2 K/s 97% 1:56 am prabo@gmail.com Home Activate Card Profile Payment Timetable Trip LOG OUT	Drawer List	

	 9:41 Enter Your Smart Card Number Your account will be linked to this number 778 752 785 Activate	 Mobitel Mobitel 0 K/s 97% 1:57 am Activate Card Enter your smart card number Your account will be linked to this number Phone number ACTIVATE	Same UI	

	 9:41 Bus Train Credit Balance 2500 LKR Top Up Colombo Fort Railway Recent Tour Colombo - Negombo 120LKR Promos GOLD MEMBER GET OFF 15% Account Trips Settings Contact us	 9:57 Home LOG OUT Credit Balance 2650.00 LKR Top Up SCAN ME! Recent Tours Tour 1 Start : Colombo End : Kadawatha Home Timetables Trips Payments	Same UI	

	 9:41 Top Up Enter Top Up ammount VISA MasterCard Enter card number MM / YEAR CVC Proceed	 Mobitel Mobitel 141 B/s 98% 1:16 am Payment 345 1280 4380 6685 2886 DEMO USER MONTH/YEAR 11/08 CCV CARDHOLDER'S NAME demo user POS 34	Same UI	

			View trip history	

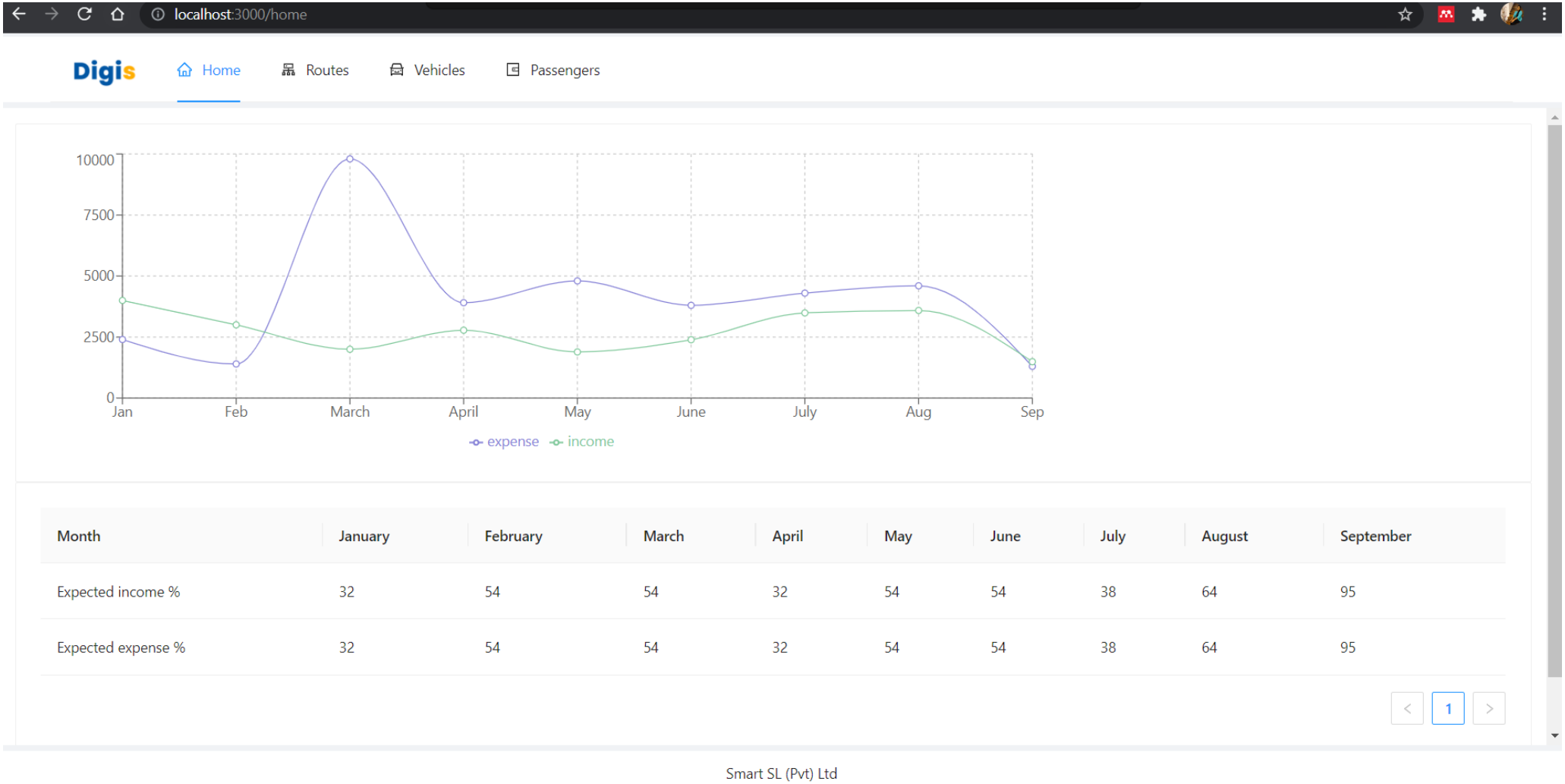
			Search timetable added in the same Timetable user interface	

Web

Login

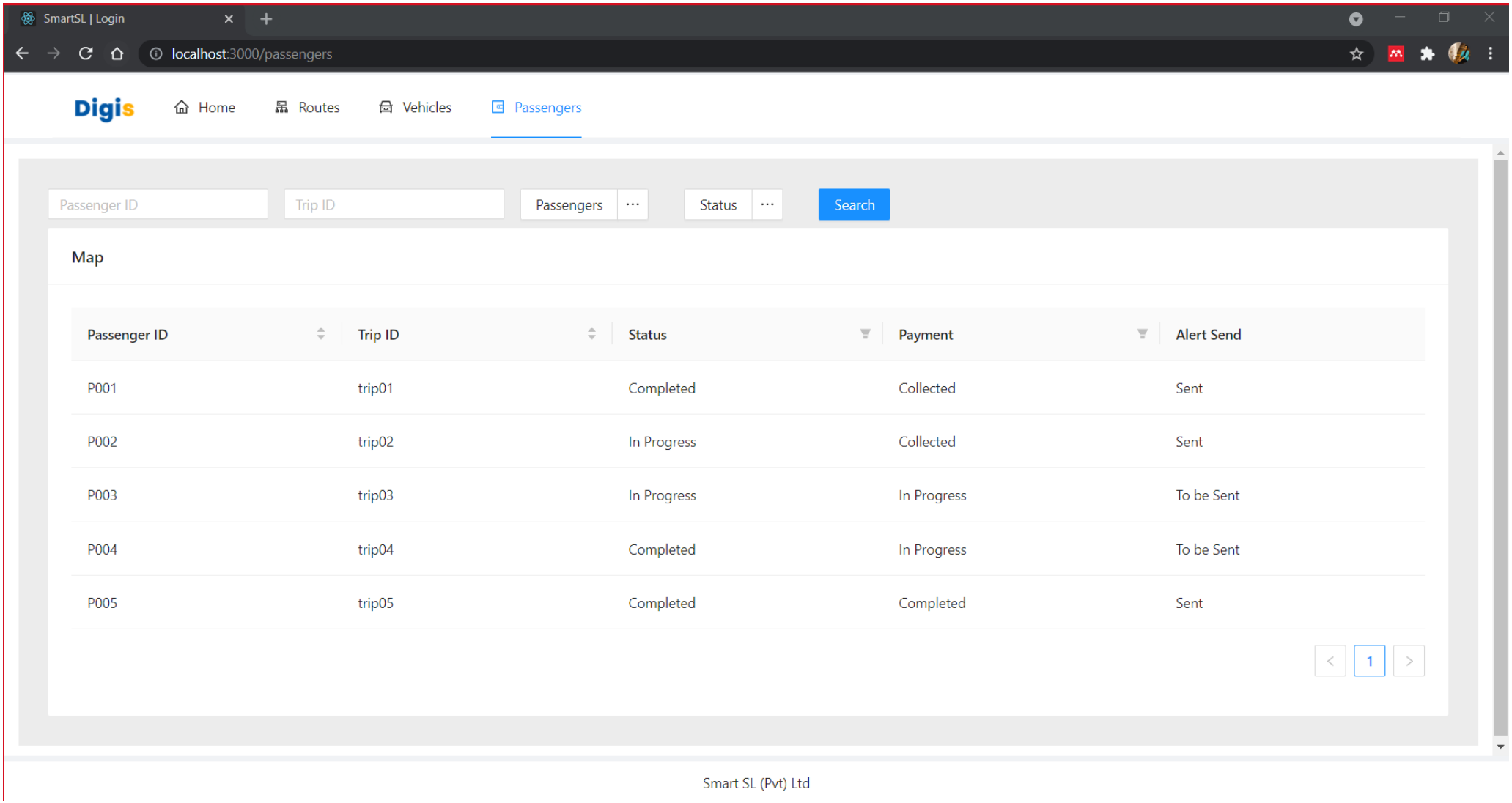

Smart SL
Public Transport Ticketing System
Email/Username
Password
Forgot Password
Login
Already a User?Sign-Up

Home

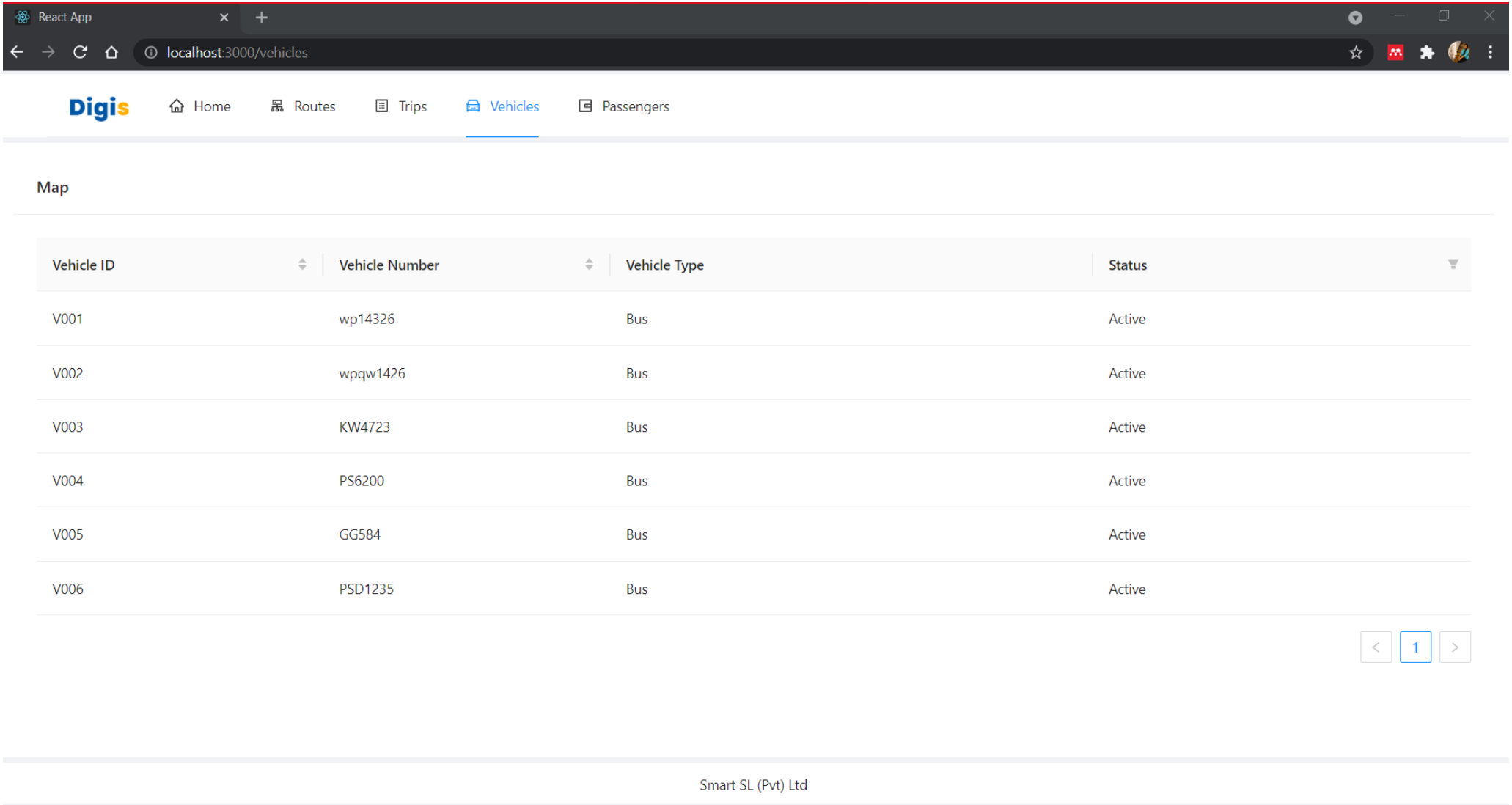


Smart SL (Pvt) Ltd

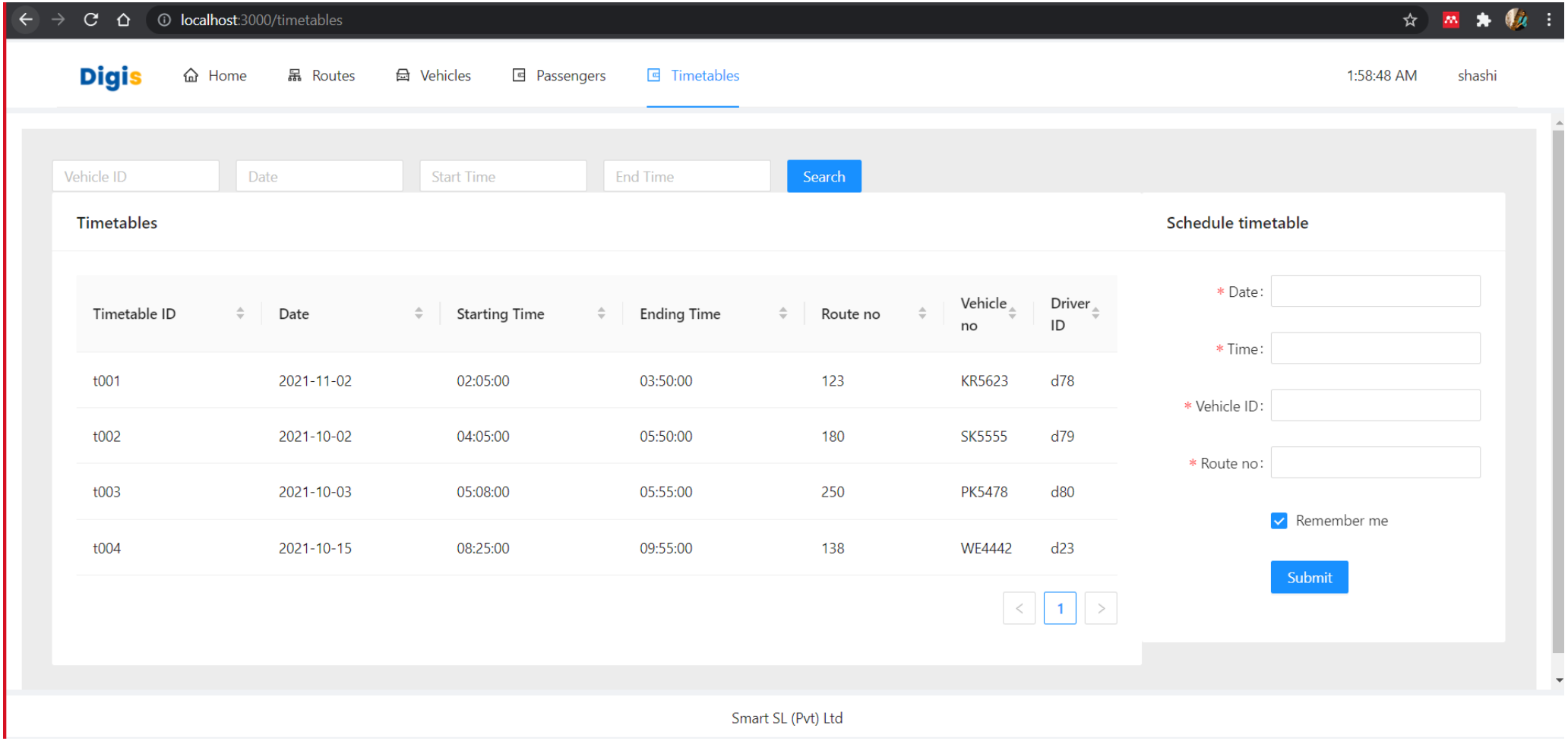
Passengers


SmartSL | Login
localhost:3000/passengers
Digis
Home
Routes
Vehicles
Passengers
Passenger ID
Trip ID
Passengers
Status
Search
Map
Passenger ID
Trip ID
Status
Payment
Alert Send
P001 trip01 Completed Collected Sent
P002 trip02 In Progress Collected Sent
P003 trip03 In Progress In Progress To be Sent
P004 trip04 Completed In Progress To be Sent
P005 trip05 Completed Completed Sent
1
Smart SL (Pvt) Ltd

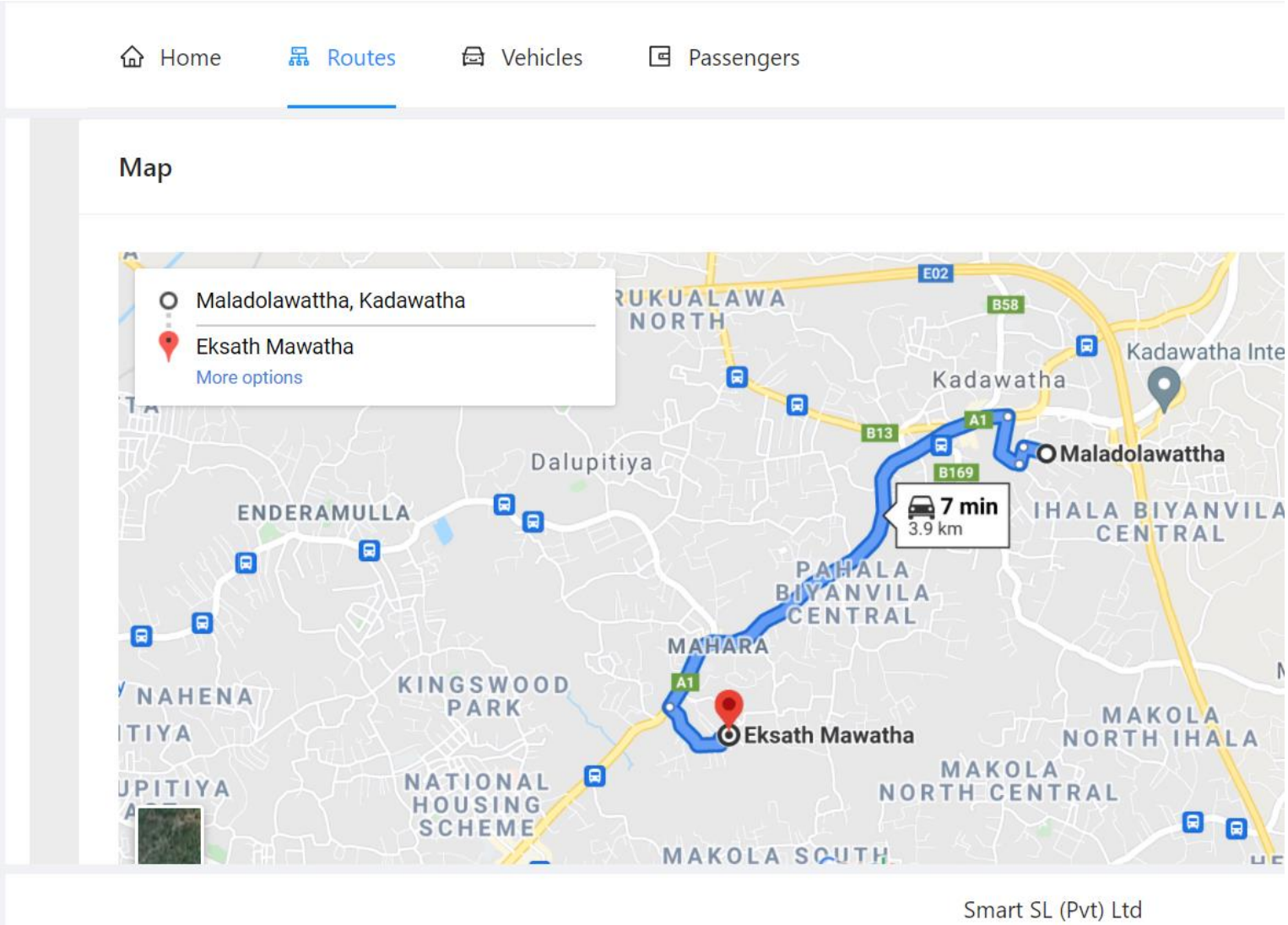
Manage vehicles



Schedule Timetable



Routes


Home
Routes
Vehicles
Passengers
Map
Maladolawattha, Kadawatha
Eksath Mawatha
More options
E02
B58
B13
A1
B169
Kadawatha
Kadawatha Inte
Maladolawattha
7 min
3.9 km
Dalupitiya
ENDERAMULLA
IHALA BIYANVILA CENTRAL
PAHALA BIYANVILA CENTRAL
MAHARA
KINGSWOOD PARK
NAHENA
Eksath Mawatha
MAKOLA NORTH IHALA
MAKOLA NORTH CENTRAL
NATIONAL HOUSING SCHEME
MAKOLA SOUTH
Smart SL (Pvt) Ltd

Design Patterns

1. Singleton

UserServiceImpl.java 2

src > main > java > com > esad > smartsl > service > impl > UserServiceImpl.java > UserServiceImpl

```java
import com.google.common.collect.Lists;

@Service
public class UserServiceImpl implements UserService {

    private static final Logger logger = LoggerFactory.getLogger(UserServiceImpl.class);

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private ModelMapper mapper;

    @Override
    public ResponseDto<UserDto> createUser(UserDto userDto) {

        // get the current date and time
        Date today = new Date();

        userDto.setCreatedAt(new Timestamp(today.getTime()));

        User user = mapper.map(userDto, User.class);

        user = userRepository.save(user);
```

- First a UserRepository is created that manages our User domain objects.
- Next the UserController is created, which uses the UserRepository to return the users.

2. Facade

Using Facade supplies the client with an interface via which the client may access the system

CardPackageFacade.java

src > main > java > com > esad > smartsl > facade > CardPackageFacade.java > ...

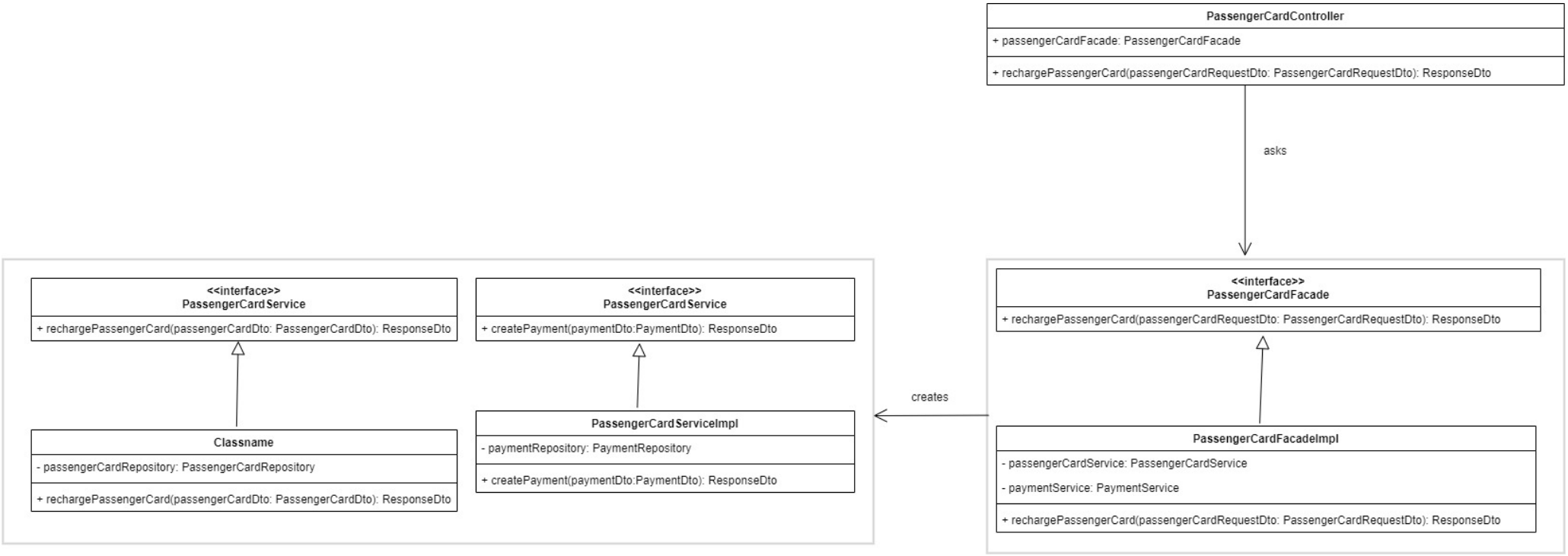
```java
package com.esad.smartsl.facade;

import javax.transaction.Transactional;

import com.esad.smartsl.domain.dto.CardPackageDto;
import com.esad.smartsl.util.dto.responseDto.ResponseDto;
import com.esad.smartsl.util.dto.searchDto.CardPackageSearchDto;

@Transactional
public interface CardPackageFacade {

    ResponseDto<CardPackageDto> createCardPackage(CardPackageDto cardPackageDto);

    ResponseDto<CardPackageDto> updateCardPackage(CardPackageDto cardPackageDto);

    ResponseDto<CardPackageDto> searchCardPackage(CardPackageSearchDto cardPackageSearchDto) throws Exception;

    ResponseDto<CardPackageDto> findCardPackageById(CardPackageSearchDto cardPackageSearchDto);

    ResponseDto<CardPackageDto> deleteCardPackage(CardPackageDto cardPackageDto);

}
```

CardPackageFacadeImpl.java

src > main > java > com > esad > smartsl > facade > impl > CardPackageFacadeImpl.java > {} com.esad.smartsl.facade.impl

```java
package com.esad.smartsl.facade.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.esad.smartsl.domain.dto.CardPackageDto;
import com.esad.smartsl.facade.CardPackageFacade;
import com.esad.smartsl.service.CardPackageService;
import com.esad.smartsl.util.dto.responseDto.ResponseDto;
import com.esad.smartsl.util.dto.searchDto.CardPackageSearchDto;

@Service
public class CardPackageFacadeImpl implements CardPackageFacade {

    @Autowired
    private CardPackageService cardPackageService;

    @Override
    public ResponseDto<CardPackageDto> createCardPackage(CardPackageDto cardPackageDto) {
        ResponseDto<CardPackageDto> responseDto = new ResponseDto<CardPackageDto>();

        responseDto = cardPackageService.createCardPackage(cardPackageDto);

        return responseDto;
    }
```

1. Call to the CardPackageFacade through the controller.
2. Using CardPackageFacade implementation we call service layer.


PassengerCardController
+ passengerCardFacade: PassengerCardFacade
+ rechargePassengerCard(passengerCardRequestDto: PassengerCardRequestDto): ResponseDto
asks
<<interface>>
PassengerCardFacade
+ rechargePassengerCard(passengerCardRequestDto: PassengerCardRequestDto): ResponseDto
PassengerCardFacadeImpl
- passengerCardService: PassengerCardService
- paymentService: PaymentService
+ rechargePassengerCard(passengerCardRequestDto: PassengerCardRequestDto): ResponseDto
creates
<<interface>>
PassengerCardService
+ rechargePassengerCard(passengerCardDto: PassengerCardDto): ResponseDto
<<interface>>
PassengerCardService
+ createPayment(paymentDto:PaymentDto): ResponseDto
Classname
- passengerCardRepository: PassengerCardRepository
+ rechargePassengerCard(passengerCardDto: PassengerCardDto): ResponseDto
PassengerCardServiceImpl
- paymentRepository: PaymentRepository
+ createPayment(paymentDto:PaymentDto): ResponseDto

3. Strategy

We generate objects that represent multiple strategies and a context object whose behavior changes according to its strategy object in the Strategy pattern. The strategy object modifies the context object's execution algorithm.

NoBusFareStrategy.java

src > main > java > com > esad > smartsl > strategy > NoBusFareStrategy.java > {} com.esad.sr

```java
package com.esad.smartsl.strategy;

/**
 * @author Lahiru Ramesh 2021-10-29
 */
public class NoBusFareStrategy implements FareCalculationStrategy{
    @Override
    public double calculate(double distance) {
        return 0;
    }
}

```

```java
public class LuxuryBusFareStrategy implements FareCalculationStrategy{

    @Autowired
    private BusFareStageRepository busFareStageRepository;


    @Override
    public double calculate(double distance) {
        double basePrice = distance*0.010;
        double distancePrice =  busFareStageRepository.getBaseRate(distance);
        return basePrice + distancePrice;
    }
}
```

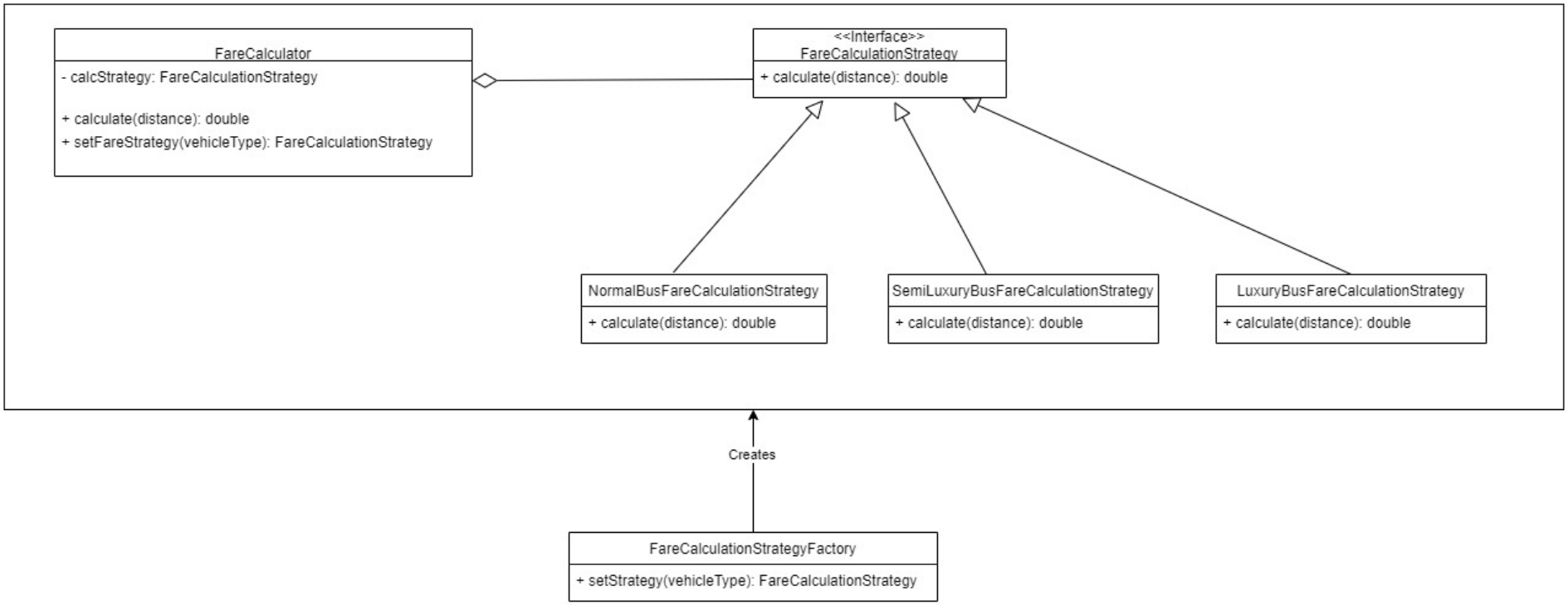
4. Factory

We build objects using the Factory pattern without disclosing the creation code to the client and then refer to the newly formed objects through a common interface.

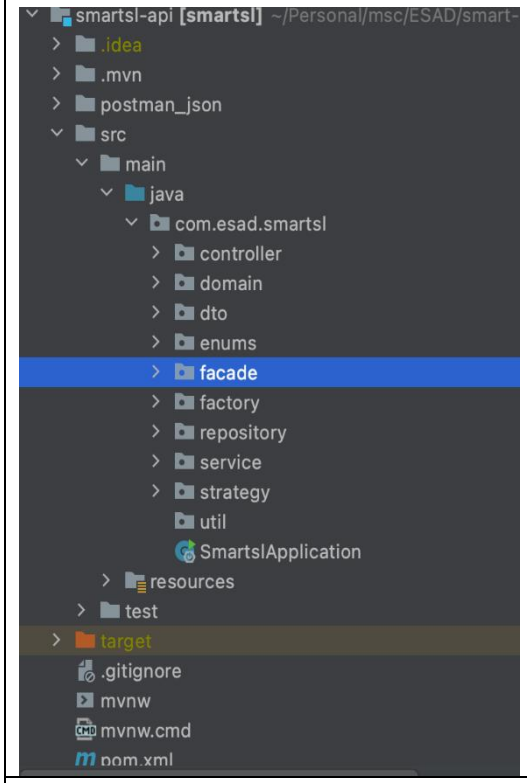
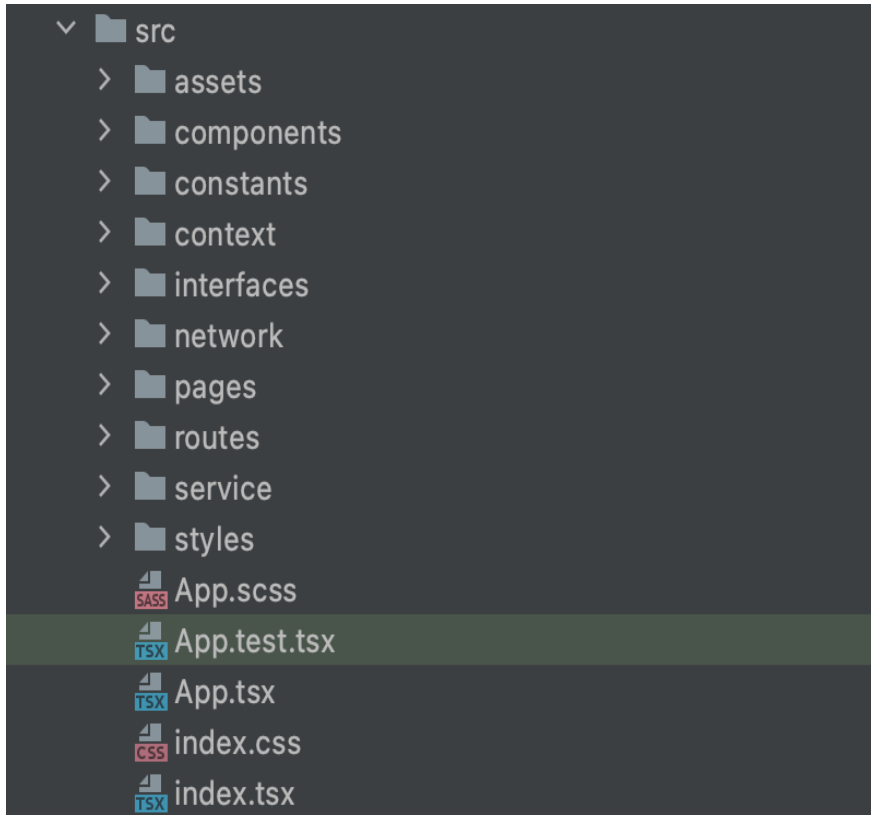
FareCalculationStrategyFactory.java

src > main > java > com > esad > smartsl > factory > FareCalculationStrategyFactory.java > ...

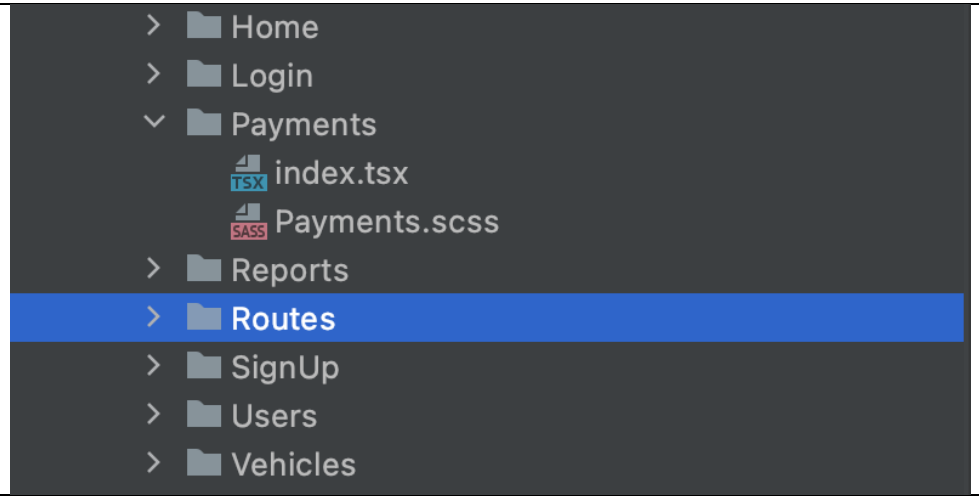
```java
package com.esad.smartsl.factory;

import com.esad.smartsl.strategy.*;

public class FareCalculationStrategyFactory {

    static FareCalculationStrategy fareCalculationStrategy;

    public static FareCalculationStrategy getStrategy(int vehicleType) {
        switch (vehicleType) {
            case 1: fareCalculationStrategy = new NormalBusFareStrategy();
            case 2: fareCalculationStrategy = new SemiLuxuryBusFareStrategy();
            case 3: fareCalculationStrategy = new LuxuryBusFareStrategy();
            default: fareCalculationStrategy = new NoBusFareStrategy();
        }
        return fareCalculationStrategy;
    }
}
```


FareCalculator
- calcStrategy: FareCalculationStrategy
+ calculate(distance): double
+ setFareStrategy(vehicleType): FareCalculationStrategy
<<Interface>>
FareCalculationStrategy
+ calculate(distance): double
NormalBusFareCalculationStrategy
+ calculate(distance): double
SemiLuxuryBusFareCalculationStrategy
+ calculate(distance): double
LuxuryBusFareCalculationStrategy
+ calculate(distance): double
Creates
FareCalculationStrategyFactory
+ setStrategy(vehicleType): FareCalculationStrategy

Coding Standards & Best practices

Back-end	Front-end
1. Modular project structure  	1. Modular project structure 
2. Keep controllers clean and simple	3. Keep components small and function specific

```java
public class CardPackageController {

    private static final Logger logger = LoggerFactory.getLogger(CardPackageController.class);

    @Autowired
    private CardPackageFacade cardPackageFacade;

    @PostMapping("/create-card-package")
    public ResponseEntity<ResponseDto<CardPackageDto>> createCardPackage(@RequestBody CardPackageDto cardPackageDto) {

        return new ResponseEntity<>(cardPackageFacade.createCardPackage(cardPackageDto), HttpStatus.OK);

    }

    @PostMapping("/update-card-package")
    public ResponseEntity<ResponseDto<CardPackageDto>> updateCardPackage(@RequestBody CardPackageDto cardPackageDto) {

        return new ResponseEntity<>(cardPackageFacade.updateCardPackage(cardPackageDto), HttpStatus.OK);

    }
```



4. Business Logic in services Layer

```java
import com.esad.smartsl.enums.OrderDirection;
import com.esad.smartsl.enums.ResponseMessage;
import com.google.common.collect.Lists;

@Service
public class PassengerCardServiceImpl implements PassengerCardService {

    private static final Logger logger = LoggerFactory.getLogger(PassengerCardServiceImpl.class);

    @Autowired
    private PassengerCardRepository passengerCardRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private ModelMapper mapper;

    @Override
    public ResponseDto<PassengerCardDto> createPassengerCard(PassengerCardDto passengerCardDto) {

        // get the current date and time
```

5. Keep postman for API documentation

<table>
<tr>
<td>
smart sl
user
POST create user
POST update user
POST search user
POST find user by id
user role
SmartCard
Payments
Routes
Vehicles
Auth
TimeTables
POST create route
</td>
<td></td>
</tr>
<tr>
<td>
6. Follow SOLID Principles

1. Single Responsibility
2. Open Close Principle
3. Interface Segregation Principle
</td>
<td></td>
</tr>
</table>