Deliverable #2

SE 3A04: Software Design II – Large System Design

Tutorial Number: T01
Group Number: Group 5
Group Members: Arash, Daniel, Matthew, Waleed, Willie

1 Introduction

1.1 Purpose

This document describes a Taxi Carpooling mobile application, integrated with an existing taxi company. The application will allow customers to share any existing taxi ride within any of the taxi company's vehicles, offering a carpool option to any nearby riders along the user's trip.

The purpose of this document is to specify the class and system architecture of the application overall, as well as all the subsystems that the system consists of. It analyses each class entity of the application, and describes the purpose and relationship with one another, how it interacts to make this application functional for production use.

The intended audience of the document is primarily the development team that is responsible for the design and implementation of the application. It is also useful for client's project management team, so they can get a basis of the technical aspects that the app will entail. This document can also be used by those in QA or responsible for testing of the application to ensure it meets the specified architectural requirements.

1.2 System Description

The system is a taxi carpool matching system that allows users to give others an opportunity to carpool with them. This is done by using the application to "offer a carpool" to other application users that would like to also reach the same destination. Main riders "offer a carpool", and other users must "request a carpool" to be able to be considered additional riders. Within the carpooling vehicle, riders will be able to access an additional gambling feature that allows riders to wager their fares with other riders, this way the challenger can either reduce or increase their fare.

To allow users to request and match with each other for carpooling, each user must have their own account and profile to distinctively identify one another, and maintain past ride history. The system contains an account login and registration, profile management, ride offer and requests, taxi map, and gambling game.

1.3 Overview

This document includes a class analysis diagram, an overall structural software architecture and the CRC cards that will be used for further object-oriented analysis. The class analysis diagram includes a breakdown of all the boundary, identity and control classes of the system that were derived from the use cases. Furthermore, the diagram also displays the connections between each class and its attributes. The subsystems of the application are listed below to outline distinct features and its relations to their functions.

2 Analysis Class Diagram

Boundary Classes

- Map Page
  - Implements Google Maps API
- Route Input Page
- Login Page
- Login Fail Page
- Start Registration Page
- Registration Error Page
  - If wrong username and password
- Registration Success Page
- View Profile Page
- Delete Profile Page
- Edit Profile Page
- Edit Profile Page Error
  - If we are changing username to someone else's username or password to something not strong)
- Fare Display Page
- Rating Riders Page
- Request Carpool Page
- Search Rides Page
- View Ride Matches Page
- Scan QR code Page
- Offer Taxi Carpool Page
- Accept/Deny Request Page
- Start Gambling Game Page
- Accept/Deny Game Page
- Result of game page
- Challenge Inelligibility
  - Not old enough, must be legal age to gamble (checks profile)

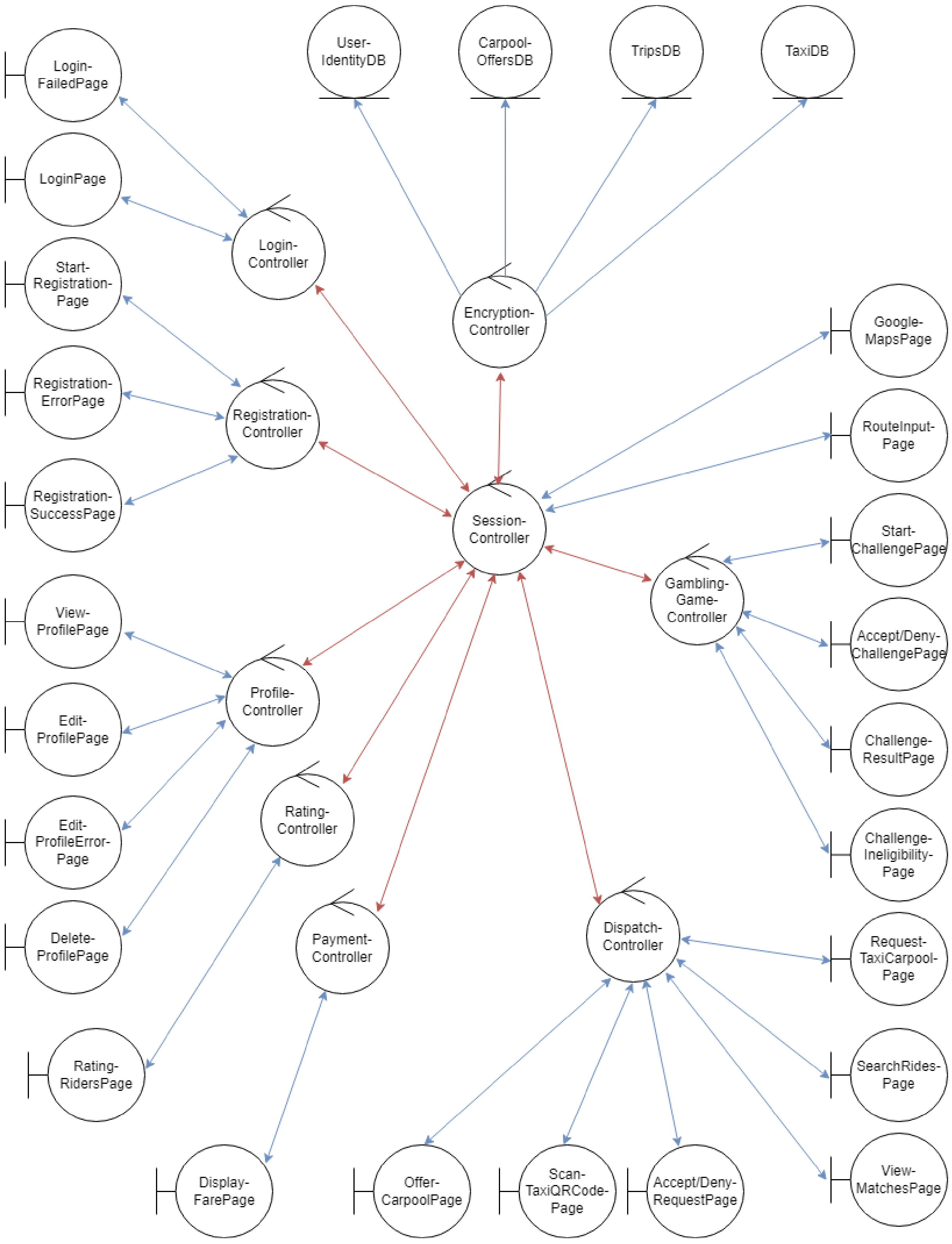
Entity Classes

- User identity
  - Username
  - Profile picture?
  - Average Rating
  - List of Ratings
- User information
  - Legal name
  - Email
  - Phone number
  - Password
- User IdentityDB
  - Stores all users
- Trips DB

    - Stores trip info
        - Route
        - Payment total
        - Carpool passengers
- OffersDB
    - Stores the offers made by main riders

Controllers

- Session controller
- Dispatch controller
    - Offer Controller
    - Request Controller
    - Functions
        - Request taxi carpool
        - Cancel request
        - Offer taxi carpool
        - Cancel offer
        - Add user to carpool
- Registration controller
    - Functions
        - Create new user
        - Log in user
- Encryption controller
    - Functions
        - Encrypt
        - Decrypt
- Ride controller
    - Functions
        - Calculate payment
        - Rate rider
        - Gambling game
        - Dispatch
        - Map
- Rating controller
- Payment controller
- Gambling Game controller
    - Functions
        - Play game
- Log in controller
- Profile controller
- Map controller


User-IdentityDB
Carpool-OffersDB
TripsDB
TaxiDB
Login-FailedPage
LoginPage
Login-Controller
Encryption-Controller
Start-Registration-Page
Registration-ErrorPage
Registration-Controller
Registration-SuccessPage
Google-MapsPage
RouteInput-Page
Session-Controller
Start-ChallengePage
Gambling-Game-Controller
Accept/Deny-ChallengePage
View-ProfilePage
Profile-Controller
Edit-ProfilePage
Challenge-ResultPage
Rating-Controller
Edit-ProfileError-Page
Challenge-Ineligibility-Page
Delete-ProfilePage
Payment-Controller
Dispatch-Controller
Request-TaxiCarpool-Page
Rating-RidersPage
SearchRides-Page
Display-FarePage
Offer-CarpoolPage
Scan-TaxiQRCode-Page
Accept/Deny-RequestPage
View-MatchesPage

3 Architectural Design

3.1 System Architecture

The system will implement the Model-View-Controller (MVC) architecture. This architecture provides a separation between 3 components, the model, the views and the controller:

- The Model component represents the application's data and business logic and allows for interaction with the data. It includes the data storage where it can store and retrieve data corresponding to the instructions of the controller. It is also responsible for notifying the view after an internal state change (within the model).
- The View component represents the user interface (how the user interacts with the application). It sends messages that the user has made an interaction from a user input, to the controller (ex. A click of a button). It receives notifications from the model and updates accordingly. Furthermore, it presents the updated data that is retrieved from the Model to the user.
- The Controller component handles the user interactions and will mainly act as an interface between the model and view components to process all the requests. It receives messages from the view, retrieves data from the model, and updates that data based on the user inputs received from the messages.

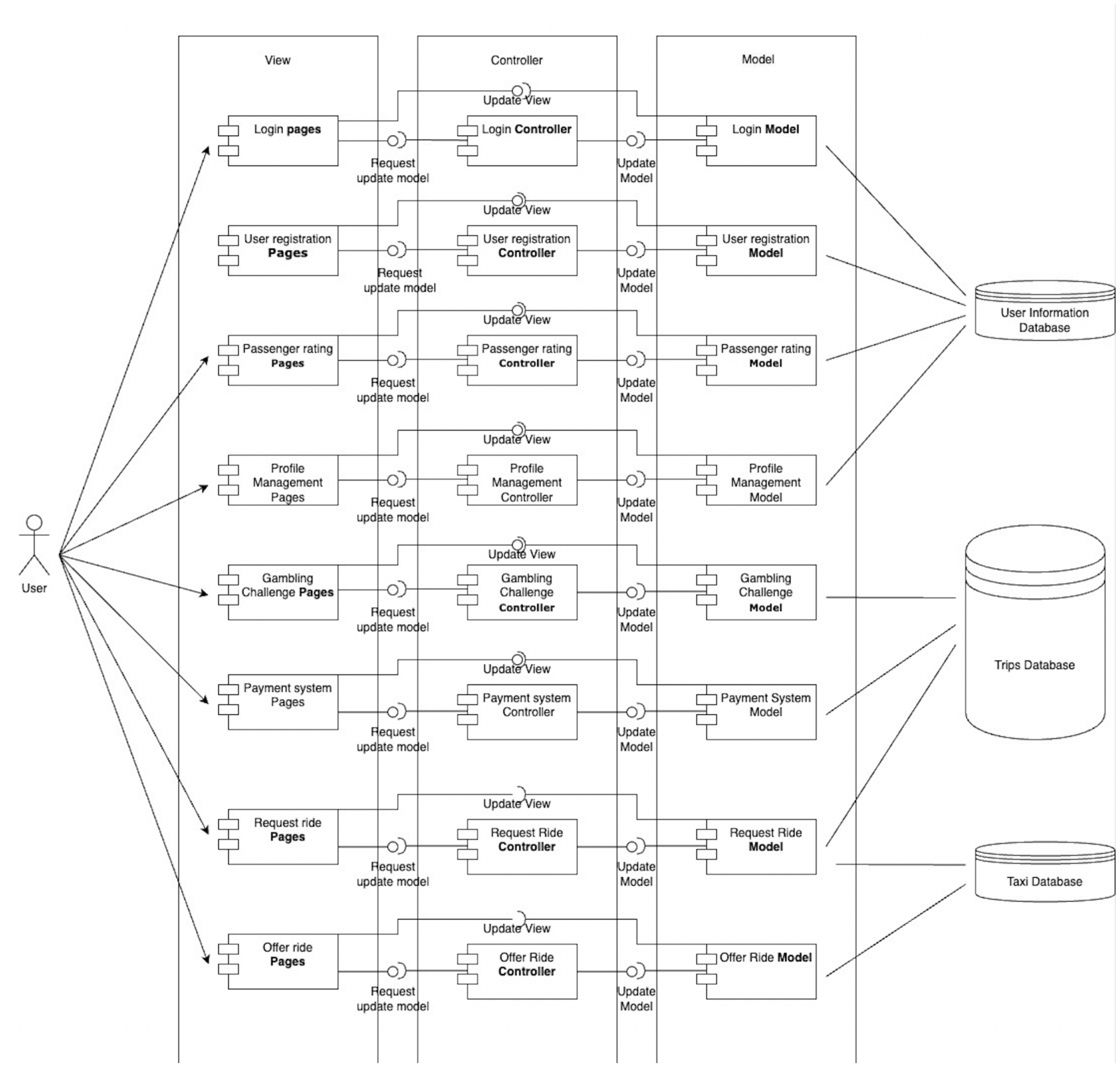
The MVC style architecture was chosen to be the best architecture to represent the overall system because it is a user-friendly interactive software architecture that implements a separation of concerns allowing for the system to be manageable, maintainable and scalable.

Firstly, this architecture promotes manageability and maintainability, by separating the application's data, business logic and user interface. Since we can isolate each section, we can work on each component independently. Developers can modify a component while all other components remain unaffected. Since the carpooling system we are building is separated into several modules (logging in, offering carpool, additional gambling component), it is important for us to choose a style where the components have the property of being independent of one another. This will allow us to independently manage and maintain different parts of the system. We will be able to make additional changes (ex. adding new features) without introducing new bugs to the system, this will be a significant help in managing and maintaining our system.

Secondly, this architecture is highly scalable as it has components that can be independently scaled. The model, view and controller each can be scaled: the model can be scaled to handle large volumes of data, the view can be scaled to have a more pleasant user interface and lastly, the controller can be scaled to handle view messages and data transactions from the model more efficiently. For our carpooling app, each component needs to be able to handle a larger work load, as more users are added to the system. By using this architecture for this application, we can scale the model, view and controller components individually to handle issues like increasing downtime, causing performance issues and harming the data's integrity.

In order to implement this architecture class diagram in our designed analysis class diagram we have designed the model below. In the diagram below, the user will be interacting with the interfaces that are mentioned below, where each of the interfaces will represent the pages in our class analysis diagram that the user will be interacting with. The Controller will correspond to the controllers in our class analysis diagram. And lastly, the model will correspond to each of the controller-databases that we have mentioned in our class analysis diagram which is mainly responsible for interacting with the database and updating the database.

Structural Architecture Diagram of MVC:



Prior to choosing the MVC model, we looked for alternative models such as the repository and blackboard architecture style. However, both are more data centered, it was difficult to pick one to model our overall carpooling system as there is a high dependence on the data store and does not align with the system's underlying purpose.

In the repository architecture style, it is used for a system that is mainly focussed on managing the persistence of data. This may result in a less flexible and modular system, as there is a high dependency between the data structure of the data store and it's agents. Whereas, in the application, a more independent system allows for less reliability and increased maintainability in case an issue with the data structure presents itself. Thus, in order to have a low coupled system, it is better to not go with the repository architecture.

In a blackboard architecture style, it is most commonly used for retrieving specified knowledge and rule sources to compute a response from the system. In contrast, the purpose of the application is to simultaneously handle multiple device's interactions and communication between each other. A blackboard architecture would be insufficient for the purpose of this application, as it supports applications designed for informational purposes, hence why we did not go with this architecture.

3.2 Subsystems

1. **Carpooling subsystem**

   A single, user (main rider) offers their taxi as a carpool taxi, and other passengers request, match with, and joins the main rider in their taxi. Also handles ride cancellations. Taxi directions and ride duration is re-computed with each new rider.

2. **Gambling challenge subsystem**

   Any single user (designated player) of the current riders in the taxi carpool may request a gambling challenge to all other riders. Each other rider may accept or deny this request to join into the gambling pool, where a randomized, even probability is rolled between the designated users vs. all accepted players. Winner's fare payment is decreased, and loser's fare payment is increased.

   Relationship: Only exists corresponding with carpooling system; can only exist if a carpool ride exists

3. **Taxi fare payment subsystem**

   Fare costs for each carpool rider are finalized and displayed. Costs are calculated depending on each rider's duration of the ride, and the number of passengers they shared with during their ride.

   Relationship: Only exists corresponding with the carpooling system, and also affected by the gambling challenge subsystem, if a gambling challenge has occurred.

4. **Passenger rating subsystem**

   Upon ride completion, each rider will be prompted with the option to provide a rating of any other carpool riders along their trip.

   Relationship: Only exists corresponding with carpooling system, subsystem only available after a carpool has been completed

5. **User registration/login subsystem**

   Users opening the app for the first time are prompted with a login screen, where new users can register a new account, and returning users can log in to their existing account.

6. **Profile management subsystem**

Profile information such as personal details, past trips, ratings, etc. are linked to each user. Users can modify details such as name, email, favourite destinations, preferred ride settings, etc.

Relationship: User registration/login subsystem; each profile is linked with a user account

4 Class Responsibility Collaboration (CRC) Cards

Login-Page	
Responsibility:	**Collaborators:**
Allows user to enter email	
Allows user to enter password	
~~Sends the email and password to the Login-Controller~~	~~Login-Controller~~
Handle click-event of "Login" button	Login-Controller

Login-Failed Page	
Responsibility:	**Collaborators:**
Displays login error	
Handle click-event of "Try Again" button	~~Login-Controller~~

Login Controller	
Responsibility:	**Collaborators:**
Handles Login Event	Session Controller
~~Receives entered login and password from the user~~	
~~Sends the enter login info to the session controller~~	~~Session-Controller~~
Send receive request from User-Identity DB	Session-Controller

Start-Registration Page	
Responsibility:	**Collaborators:**
Displays text boxes to enter attributes	
Can entername from user input	
Can enter email from user input	
Can enter password from user input	
Handles click of “Register” button	Registration-Controller
Sends registration information to the registration Controller	

Registration-Error Page	
Responsibility:	**Collaborators:**
Displays Registration Error	
Handles click of “Registration Page” button	Registration-Controller

Registration-Success Page	
Responsibility:	**Collaborators:**
Displays Registration Success	
Handles click of “Home/Menu” button	Registration-Controller

Registration-Controller	
Responsibility:	**Collaborators:**
Sends the registration information to the user identity DB through different controllers	Session Controller, Encryption Controller , User-Identity DB
~~Knows User-Identity DB (from last line)~~	
Handles email/phone confirmation	Encryption Controller
Enters home/menu of App	Session Controller

Offer-Carpool Page	
Responsibility:	**Collaborators:**
Takes input from the user to activate the offer carpool mode	
Sends the input from the user to the Offer Controller	Offer Controller
Displays the option to offer carpool	

User Information DB	
Responsibility:	**Collaborators:**
Knows Password	~~Encryption Controller, Session Controller, Registration Controller~~
Knows profile Picture	~~Encryption Controller, Session Controller, Profile Controller~~
Knows email/phone number	~~Encryption Controller, Session Controller, Profile Controller~~
Knows username	~~Encryption Controller, Session Controller, Registration Controller~~

User Identity DB	
Responsibility:	**Collaborators:**
Stores name	Encryption Controller, Session Controller, Registration Controller
Stores profile picture	Encryption Controller, Session Controller, Profile Controller
Stores ratings list and average	Encryption Controller, Session Controller, Ride Controller, Rating Controller
~~Sends all the information to the encryption controller~~	Encryption Controller

Carpool Offers DB	
Responsibility:	**Collaborators:**
Knows previous carpool offers made	~~Encryption Controller, Session Controller, Ride Controller, Dispatch Controller~~
Knows previous carpool offers' invoice	~~Encryption Controller, Session Controller, Ride Controller, Dispatch Controller~~
Knows previous carpool drivers	~~Encryption Controller, Session Controller, Ride Controller, Dispatch Controller~~

Trips DB	
Responsibility:	**Collaborators:**
Knows past trips details	~~Encryption Controller, Session Controller, Ride Controller~~
Knows past trip’s invoice	~~Encryption Controller, Session Controller, Ride Controller, Payment Controller~~
Knows past trip’s drivers	~~Encryption controller, Session Controller, Ride Controller~~

Encryption Controller	
Responsibility:	**Collaborators:**
Send write/edit/delete/retrieve request to User-Identity DB	User-Identity DB
Send write/edit/retrieve request to Carpool-Offers DB	Carpool-Offers DB
Send edit/edit/retrieve request to TripsDB	TripsDB
Send authenticated data to Session Controller	Session Controller

Session Controller	
Responsibility:	**Collaborators:**
Receives user information from the profile Controller	Profile Controller
Receives user information from the registration Controller	Registration Controller
Authenticates the user	
Receives authenticated encrypted information from the encryption controller	Encryption Controller
Sends trip information to the encryption Controller	Ride Controller

Map Controller	
Responsibility:	**Collaborators:**
Receives pick up location input from Route-Input page	Route Input page
Receives drop off location input from Route-Input Page	Route Input page
Receives input from the page	Google-Maps Page
Sends the pickup/drop off location to the Google Maps page	Google-Maps Page
Sends the current/pick up/drop off location to the Ride Controller	Ride Controller

Google-Maps Page	
Responsibility:	**Collaborators:**
Displays current location	
Displays pick up location in map	Map Controller
Display drop off location	Map Controller
Displays estimated arrival time	Map Controller
Displays route to destination	Map Controller

Route-Input Page	
Responsibility:	**Collaborators:**
Allows user to input pick up location	
Allows the user to input drop off location	
Sends user's input to map Controller	Map Controller

Ride Controller	
Responsibility:	**Collaborators:**
Receives the challenge results from the Gambling-Game Controller	Gambling-Game Controller
Receives the profile information from the Session Controller	Session Controller
Receives the encrypted data from the Session Controller	Session Controller
Receives the pickup/drop off location from the Map Controller	Map Controller
Receives rating information from the Rating Controller	Rating Controller
Receives Invoice information from the Payment Controller	Payment Controller
Receives offers and request for carpool from the Dispatch Controller	Dispatch Controller
Sends all the trip information to Session Controller	Session Controller
Sends user information to the dispatch Controller	Dispatch Controller
Sends user information to the Rating Controller	Rating Controller

Gambling-Game Controller	
Responsibility:	**Collaborators:**
Receives inputs from the accept/deny challenge page	Accept/Deny Challenge Page
Calculates the Results for the challenge	
Sends the result to the Challenge-Result page	Challenge Result page
Receives inputs from the challenge-ineligibility page	Challenge Ineligibility page
Evaluates that whether someone is ineligible to issue a challenge	Challenge Ineligibility page
Sends the Challenge results to the ride controller	Ride Controller

Start-Challenge Page	
Responsibility:	**Collaborators:**
Display description of gambling game	
Display confirmation to start a game request	
Sends the user input confirmation to the Gambling-Game Controller	Gambling-game controller
Display roll to choose challenge winner (from challenger perspective)	

Accept/Deny-Challenge Page	
Responsibility:	**Collaborators:**
Accepts user’s input to accept or decline the challenge	
Sends the user input to the Gambling-game controller	Gambling-game-controller
Display roll to choose challenge winner (from competitor's perspective)	

Challenge-Result Page	
Responsibility:	**Collaborators:**
Display whether the user has won (unique to challenger vs competitor’s perspective)	
Display close Challenge page button	
Receives the game result from the Gambling Game Controller	Gambling Game Controller

Challenge-Ineligibility Page	
Responsibility:	**Collaborators:**
Display a message that states ineligibility to play game (unique to ineligible users attempting to send a challenge request)	
Sends the user’s “I agree” confirmation to the Gambling Game Controller	Gambling-game controller

Dispatch Controller	
Responsibility:	**Collaborators:**
Receive carpool request from Request Controller	Request Controller
Receive carpool offer from Offer Controller	Offer Controller
Send update request to Ride controller	Ride Controller

Offer Controller

Responsibility:	Collaborators:
Receives code from Scan-Taxi-QR-Code Page	Scan-Taxi-QR-Code Page
Receives offers from Offer-Carpool Page	Offer-Carpool Page
Sends offer to Dispatch Controller	Dispatch Controller
Receives requests from Dispatch Controller	Dispatch Controller
Sends requests to Accept/Deny-Request Page	Accept/Deny-Request Page
Receives request responses from Accept/Deny Request Page	Accept/Deny Request Page
Sends request responses to Dispatch Controller	Dispatch Controller

Offer-Carpool Page

Responsibility:	Collaborators:
Allows user to enter taxi information (destination, # passengers)	
Handle click event of “Offer Carpool” button	Offer-Controller

Scan-Taxi-QR-Code Page

Responsibility:	Collaborators:
Allows user to scan QR code using camera	
Handle sending code to Offer-Controller	Offer-Controller

Accept/Deny Request Page

Responsibility:	Collaborators:
Allows user to view incoming carpool requests	
Allows user to accept or deny request	
Knows Offer-Controller	
Handle response of request	Offer-Controller

Request-Controller	
Responsibility:	**Collaborators:**
Receives the search date filters from the Search-Rides Page	Search-Rides Page
Sends the matched results to the View-Matches Page	View-Matches Page
Finds matches that fit the time Constraint	
Receives Confirmation to search for a match	Request Taxi-Carpool Page
Sends the Finalized Matched result to the Dispatch Controller	

Request-Taxi-Carpool Page	
Responsibility:	**Collaborators:**
Allows user to enter destination with criteria	
Sends the click event of "Request Taxi-Carpool" button to Request Controller	Request-Controller

Search-Rides Page	
Responsibility:	**Collaborators:**
Searches for rides with the specified criteria	
Sends available options to the Request Controller	

View-Matches Page	
Responsibility:	**Collaborators:**
Allows user to view a list of potential matches	
Displays information about each match	
Receives Selected Match from Received Controller	Request-Controller
Sends the user's selected matched driver to the request controller	

Rating Controller	
Responsibility:	**Collaborators:**
Receive rating data from Rating-Riders page	Rating-Riders Page
Sends rating data to Ride-Controller to be stored in User-Identity DB	Ride-Controller, Ride Controller, Session Controller, Encryption Controller, User Identity DB

Rating-Riders Page	
Responsibility:	**Collaborators:**
Displays options to rate other riders from trip	Rating controller
Send user input of rating to rating controller	Rating controller

Payment Controller	
Responsibility:	**Collaborators:**
Can process payment through API to cover taxi fare	
Can determine whether payment was successful	
Can calculate fare from RideController	RideController

Display-Fare Page	
Responsibility:	**Collaborators:**
Receives fare from Payment Controller	Payment Controller
Displays fare that user has to pay	

A Division of Labour

Waleed: Contributed to the class analysis diagram, contributed to the software architectural design (worked on sections: 1.2, 2). Completed section 3.1. Completed the refined class analysis diagram (added missing elements and relationships, and increased readability).

Signature:

[illegible]

Willie: Contributed to Architecture Design with subsystems description documentation (Section 3), introduction to system overview and purpose (Section 1), and responsibility and collaborators of CRC cards.

Signature:

[illegible]

Daniel: Contributed to section 1 introduction and Class Responsibility Collaboration (CRC) Cards

Signature: daniel a

Matt: Contributed to Analysis Class Diagram (Section 2) and CRC Cards (Section 4).

Signature:

× Matthew Miller

Arash: Contributed to the system architecture design MVC diagram (section 3), and the Class Responsibility Collaboration (CRC) cards (section 4).

Signature:

[illegible]