Deliverable #2

SE 3A04: Software Design II – Large System Design

March 13, 2023

Tutorial Number: T03
Group Number: G8
Group Members:

- Adam Mak
- Eric Chen
- Justin Ho
- Ahmad Hamadi
- Kevin Ishak
- Jonathan Jiang

IMPORTANT NOTES

- Please document any non-standard notations that you may have used
  - *Rule of Thumb*: if you feel there is any doubt surrounding the meaning of your notations, document them
- Some diagrams may be difficult to fit into one page
  - Ensure that the text is readable when printed, or when viewed at 100% on a regular laptop-sized screen.
  - If you need to break a diagram onto multiple pages, please adopt a system of doing so and thoroughly explain how it can be reconnected from one page to the next; if you are unsure about this, please ask about it
- Please submit the latest version of Deliverable 1 with Deliverable 2
  - Indicate any changes you made.
- If you do NOT have a Division of Labour sheet, your deliverable will NOT be marked

1 Introduction

1.1 Purpose

The purpose of this document is to give an overview on the system and architectural design of the carpool app. The objective is to take requirements mentioned from the **SRS** and transform it into an architecture that describes the app's top-level structure and identifies its components, which acts as a preliminary blueprint for development. The intended audience for this document includes software developers and engineers, systems design architects, and other potential stakeholders who may benefit from an understanding of the system design.

1.2 System Description

The system is organized into a multiple sections, each fulfilling a requirement of the product. The user authentication section shows the systems used to ensure only registered users can use the app. The update account section takes care of any changes a user may need to make to their personal information. The arrival response section handles the display of the fare, peer-passenger rating system, and point allocation/redemption system. The dispatch ride sub-system handles the initialization of carpool events and requests of passengers to participate in pre-existing carpool events. The database system houses the different databases that are used. Finally, the default home view section takes care of the main general screen, connecting to all the other sections. The sections, and the relationships between them are illustrated in the analysis class diagram. Detailed information on any boundary class, entity class, and controller class can be found in the CRC cards section and system architecture sections of the document.

The document also describes the subsystems that the product will use, and the data architecture styles that will be utilized for these subsections Information regarding this can be found in section 3.2.

1.3 Overview

In Section 2 the composition and interaction of domain concepts is outlined via a descriptive model, which will be represented visually in the form of a analysis class diagram.

Section 3 provides a simplified overview of the overall architectural design and a simplified overview of all the subsystems that needs to be present in the final product.

Section 4 goes further in-depth into the identified classes mentioned in Section 2 by representing them as CRC cards.

At the end of the document, there is a record which represents the division of labour of all contributors to this deliverable.

2 Analysis Class Diagram

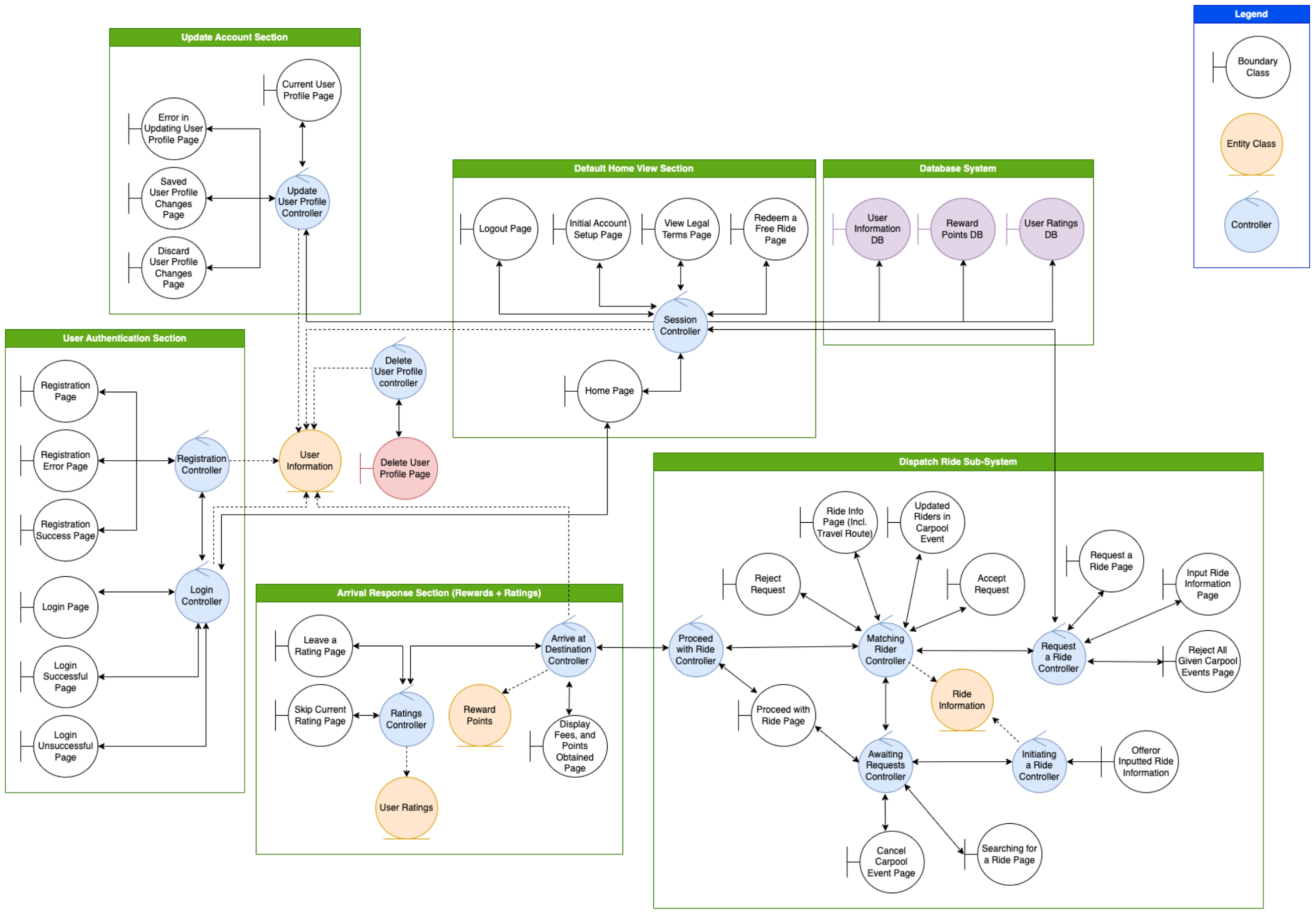
Update Account Section
Current User Profile Page
Error in Updating User Profile Page
Saved User Profile Changes Page
Update User Profile Controller
Discard User Profile Changes Page
Default Home View Section
Logout Page
Initial Account Setup Page
View Legal Terms Page
Redeem a Free Ride Page
Session Controller
Home Page
Database System
User Information DB
Reward Points DB
User Ratings DB
Legend
Boundary Class
Entity Class
Controller
User Authentication Section
Registration Page
Registration Error Page
Registration Success Page
Registration Controller
Login Page
Login Controller
Login Successful Page
Login Unsuccessful Page
Delete User Profile controller
User Information
Delete User Profile Page
Arrival Response Section (Rewards + Ratings)
Leave a Rating Page
Skip Current Rating Page
Ratings Controller
Reward Points
Arrive at Destination Controller
Display Fees, and Points Obtained Page
User Ratings
Dispatch Ride Sub-System
Ride Info Page (Incl. Travel Route)
Updated Riders in Carpool Event
Reject Request
Accept Request
Request a Ride Page
Input Ride Information Page
Proceed with Ride Controller
Matching Rider Controller
Request a Ride Controller
Reject All Given Carpool Events Page
Ride Information
Proceed with Ride Page
Awaiting Requests Controller
Initiating a Ride Controller
Offeror Inputted Ride Information
Cancel Carpool Event Page
Searching for a Ride Page

Figure 1: Analysis Class Diagram (ACD) of the carpooling application.

3 Architectural Design

3.1 System Architecture

- Identify and explain the overall architecture of your system
- Be sure to clearly state the name of the architecture you used (this is the name of the architectural pattern, not the name of your system)
- Provide the reasoning and justification of the choice of architecture
- Provide a structural architecture diagram showing the relationship among the subsystems (if appropriate)
- List any design alternatives you considered, but eliminated (and explain why you eliminated them)

The overall system will have a model view controller (MVC) architecture. Although the system involves storing and accessing data, how the data will be communicated will differ between subsystems (listed in Section 3.2). Here we are focusing more on the interaction between the user and internal workings of the application.

The MVC architecture separates the application into three components:

- Model: The model represents the data and most of the business logic in the application. The model would contain the following components:
  - User information: This includes information such as name, email, phone number, and password. This is encapsulated from the user.
  - Ride information: This includes the details of users who are looking for or offering a ride, such as their starting location, destination, date, time, and number of available seats in the taxi.
  - Reward points: These are associated with a user and are updated every time a user goes on a carpool ride.
  - User ratings
- View: Responsible for displaying the data to the user and receiving user input. It forwards input from the user to the controller and displays any input upon the controller's request to the user. These are a few components that will be included with the view:
  - Login page: This allows users to log in to their accounts.
  - Registration page: This allows users to create a new account.
  - Home page: This shows the user's profile information and allows them to request or offer rides.
  - Ride request page: This allows users to request a ride by specifying their starting location, destination, date, and time.
  - Ride offer page: This allows users to offer a ride by specifying their starting location, destination, date, time, and number of available seats.
  - Match page: This displays a list of potential matches between ride requests and offers (in the viewpoint of a requester).
- Controller: Acts as a mediator between the model and the view. It handles user input, updates the model, and updates the view accordingly. It also deals with some business logic in tandem with the model. The controller would include some of the following components:
  - Authentication controller: This handles user authentication, including login and sign-up.
  - Profile controller: This handles the user's profile information.
  - Ride request/offer controller: This handles the creation and retrieval of ride requests/offers.
  - Dispatcher: This handles the matching of ride requests and offers and updates the view accordingly.

  Note that some of these controllers are split up into sub-controllers in the analysis class diagram.

Having an system that separates components into sections that focus on different aspects of the application prioritizes separation of concerns. Each section has a specific responsibility and can be developed, tested, and maintained without having to worry too much about the other sections.

An organized system also reduces complexity and improves understandability from a developer's perspective. This will in turn improve scalability and quicken the development process.

The presentation abstraction control (PAC) architecture was also considered as it is similar to the MVC architecture. In this context, each PAC agent would represent a subsystem in our application. We found that since the presentation and abstraction do not communicate with each other, it limits flexibility in the implementation and increases the workload and complexity of the control. Furthermore, since the controls in each agent communicate with each other, scaling the application may be tougher as it needs to consider all agents. Therefore, we eliminated the PAC architecture.

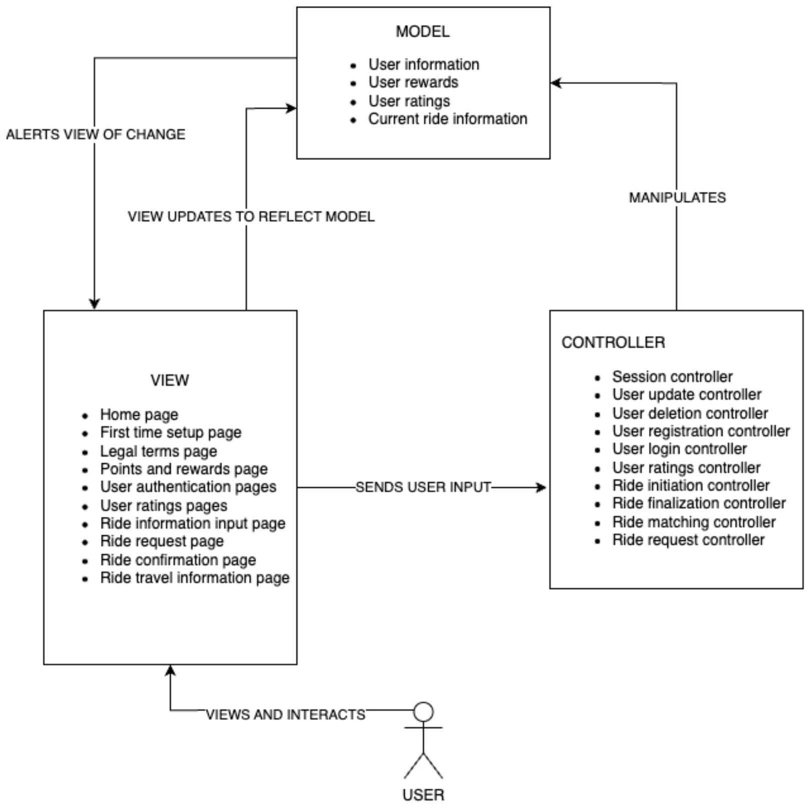

Figure 2: MVC Structural architecture diagram

3.2 Subsystems

There are 2 major subsystems that make up the overall system - the account system and the ride system.

- User Account Management Subsystem
  Purpose: The purpose of this subsystem is to provide a separation of concerns between the business logic of the account management system and the database storage layer. User authentication, registration, modification, information, ratings, and rewards are handled by this subsystem. This also permits functionality such as encryption in the various data requests and modification made by the application required in the implementation of user accounts. This subsystem also makes it easier to test the application as the functional code can be tested in isolation of the database through abstraction using a repository layer.
  Architecture: This subsystem will have a repository architecture style. Account info from all users will be stored in a centralized data store in the form of a relational database. Data will only be stored or pulled upon request from the other components such as the account controller via a DBMS. In the implementation of the app, repositories will be created for user information, ratings, and rewards. Each repository would have methods which allow for the retrieval and modification of data which could then be used by the Ride subsystem in its functionality.

- Ride Subsystem
  Purpose: The primary purpose of this subsystem involves ride matchmaking, where carpoolers are matched with passengers based on location, route, and other potential criteria. The subsystem is also responsible for fare calculation, and is linked to the rating function.
  Architecture: This subsystem will make use of a blackboard architecture design. Regarding the blackboard components, the blackboard itself would contain the data required for the ride matchmaking functions, which would include carpooler and passenger information such as location. The knowledge sources of the blackboard would be the matchmaking methods and algorithms which utilize the data

of the blackboard. The control mechanisms would be the methods which manage the interaction of the knowledge sources with the blackboard. An example of this would be methods that update the blackboard when certain events occur, such as removing carpoolers from the available rides list when their ride is initiated.

4 Class Responsibility Collaboration (CRC) Cards

Class Name: RegisterationPage	
Responsibility:	**Collaborators:**
Display the registration form	
Allow user to select "register" or "log in"	
Send request to register account to **RegistrationController**	**RegistrationController**
Know **RegistrationController**	

Class Name: RegistrationErrorPage	
Responsibility:	**Collaborators:**
Display error message for failed registration	
Provide options to retry registration	
Await with **RegistrationController** to handle registration errors	**RegistrationController**
Know **RegistrationController**	

Class Name: RegisterationSuccessPage	
Responsibility:	**Collaborators:**
Display the Succesful registration form	
Collect user input and pass it to the **RegistrationController** to store	**RegistrationController**
Know **RegistrationController**	

Class Name: RegisterationController	
Responsibility:	**Collaborators:**
Receive user input for registration information (name, email, password, etc.).	**RegistrationPage**
Validate the user input to ensure it meets certain requirements	**RegistrationSuccessPage** , **RegistrationErrorPage**
Create a new user account in the CustomerInfo database with the provided registration	**UserInformationDB**
Allow user account to be logged in with	**LoginController**
Know **UserInformationDB**	
Know **LoginController**	

Class Name: LoginPage	
Responsibility:	**Collaborators:**
Display login form	
Allow user to input login information and select "log in"	
Send request to validate credentials to **LoginController**	**LoginController**
Know **LoginController**	

Class Name: LoginUnSuccesfulPage	
Responsibility:	**Collaborators:**
Display to the user that login was unsuccessful	
Send request to LoginController about the error that occurred during user login	**LoginController**
Know **LoginController**	

Class Name: LoginSuccesfulPage	
Responsibility:	**Collaborators:**
Display to the user that login was successful	
Send request to LoginController to login	**LoginController**
Know **LoginController**	

Class Name: LoginController	
Responsibility:	**Collaborators:**
Redirect user to home page after successful login	**HomePage**, **LoginSuccesful-Page**
Redirect user to login unsuccess-ful page after unsuccessful login	**LoginUnsuccessfulPage**
Receive login request from **LoginPage**	**LoginPage**
Receive registration request from **LoginPage**	**RegistrationController**
Receive user info from **userIn-formationDB**	**UserInformation**
Know **HomePage**	

Class Name: UserInformationDB	
Responsibility:	**Collaborators:**
Store user information	
Send user info to LoginController	**LoginController**
Receive request to create new user	**RegisterationController**
Recieves Updated user informa-tion (such as password, email, etc.)	**UpdateUserProfileController**
Delete user information	**DeleteUserProfileController**
Send location info to arriveAt-DestinationController	**ArriveAtDestinationController**
Send info throughout the session	**SessionController**
Know **SessionController**	
Know **LoginController**	
Know **ArriveAtDestination-Controller**	

Class Name: UpdateUserProfileController	
Responsibility:	**Collaborators:**
Retrieve current user profile information from **UserInformationDB**	**UserInformationDB**
Display current user profile information on **CurrentUserProfilePage**	**CurrentUserProfilePage**
Allow user to edit and submit updated user profile information	
Validate user profile information updates	**SavedUserProfileChangesPage**
Update **UserInformationDB** with user profile information updates	**UserInformationDB**
Redirect to **ErrorInUpdatingProfilePage** if update fails	**ErrorInUpdatingProfilePage**
Redirect to **DiscardUserProfileChangesPage** if user cancels update	**DiscardUserProfileChangesPage**
Collaborate with **SessionController** to check user authentication	**SessionController**
Know **UserInformationDB**	
Know **CurrentUserProfilePage**	
Know **SavedUserProfileChangesPage**	

Class Name: CurrentUserProfilePage	
Responsibility:	**Collaborators:**
Display user profile information	
Allow user receive from **UpdateUserProfileController** and view updated profile information	**UpdateUserProfileController**

Class Name: ErrorInUpdatingtUserProfilePage	
Responsibility:	**Collaborators:**
Display an error message when updating the user profile fails	
Send info to allow the user to **UpdateUserProfileController** to try updating the user profile again	**UpdateUserProfileController**
Know **UpdateUserProfileController**	

Class Name: SavedUserProfilechangedPage	
Responsibility:	**Collaborators:**
Display a confirmation message when the user's profile changes have been saved	
Send info back to **UpdateUserProfileController**	**UpdateUserProfileController**
Know **UpdateUserProfileController**	

Class Name: DiscardUserProfilechangedPage	
Responsibility:	**Collaborators:**
Display a confirmation message when the user's profile changes have been discarded	
Send to **UpdateUserProfileController** that is was deleted	**UpdateUserProfileController**
Know **UpdateUserProfileController**	

Class Name: DeleteUserProfileController	
Responsibility:	**Collaborators:**
Receive a request to delete a user's profile	
Confirm the user's intention to delete their profile	**DeleterUserProfilePage**
Delete the user's profile information from **UserInformationDB**	**UserInformationDB**
Know **UserInformationDB**	

Class Name: DeleteUserProfilePage	
Responsibility:	**Collaborators:**
Display a confirmation message for deleting the user's profile	
Allow the user to confirm their intention to delete their profile	
Send a request to **DeleteUserProfileController** to delete the user's profile	**DeleteUserProfileController**
Display a success message after the user's profile has been deleted	
Know **DeleteUserProfileController**	

Class Name: SessionController	
Responsibility:	**Collaborators:**
Create session for user upon successful login	**LoginController, UserInformationDB**
Check session for user upon each request	
Allow you to traverse between different UI	
Redirect user to the 'HomePage' after successful setup	**HomePage**
Allow user to log out of session	**LogoutPage**
Create initial session for new user upon account creation	**InitialAccountSetup**
Display legal terms for user to review and accept	**ViewLegalTermsPage**
Send/direct user to Allow updating profile	**UpdateUserProfileController**
Store and retrieve user ratings, information and rewardPoints	**UserRatingsDB**, **UserInformationsDB**, **RewardsPointsDB**
Receive to Initiate and track user participation in rides	**RequestARideController**
Receive info to redeem points page after succesful trip	**RedeemAFreeRidePage**
Know **DeleteUserProfileController**	
Know **UpdateUserProfileController**	

Class Name: ViewLegalTermsPage	
Responsibility:	**Collaborators:**
Display legal terms for the user to review and accept	
Send confirmation of acceptance to **SessionController**	**SessionController**
Know **SessionController**	

Class Name: UserRatingsDB	
Responsibility:	**Collaborators:**
Store and manage ratings given by users for drivers and riders	
retreive from ratings information	**RatingsController**
send data to **SessionController**	
Know **SessionController**	

Class Name: HomePage	
Responsibility:	**Collaborators:**
Display home page with relevant information and options for users	
Allow user to view/send their profile information	**SessionController**
Allow user to search for rides	
Allow user to create and manage ride requests and reservations	
Allow user to access their ride history	
Send request to log in to ‘LoginController‘	**LoginController**
Know **SessionController**	
Know **LoginController**	

Class Name: LogoutPage	
Responsibility:	**Collaborators:**
Display confirmation message to the user that they are logging out	
Allow user to confirm or cancel the log out operation	
Send request to end user session to **SessionController**	**SessionController**
Know **SessionController**	

Class Name: InitialAccountSetupPage	
Responsibility:	**Collaborators:**
Display the initial account setup form to the user	
Allow user to input their personal and payment information	
Send request to create user account and store information	**SessionController**
send user to the ‘HomePage‘ after successful setup	**HomePage**
Know **SessionController**	
Know **HomePageController**	

Class Name: RedeemFreeRidePage	
Responsibility:	**Collaborators:**
Display the redeem free ride page	
Allow user to enter their unique code to redeem a free ride	
send current info to **SessionController** to check user authentication	**SessionController**
Know **SessionController**	

Class Name: RideInformationDB	
Responsibilities:	**Collaborators:**
Store, update, query information about rides	
Allow Retreival of Ride Information	**InitiatingARideController**
Send information to **MatchingARideController** to match a ride using the data	**MatchingARideController**
Know **MatchingARideController**	

Class Name: RequestARideController	
Responsibility:	**Collaborators:**
Handle user requests to partake in a ride	
Retrieve ride information from **InputRideInformationPage**	**InputRideInformationPage**
Retrieve user session information from **SessionController**	**SessionController**
Retrieve available rides from **RequestARidePage**	**RequestARidePage**
Notify user of successful ride request	
Notify user of unsuccessful ride request	
Allow user to select all carpool events for rejection	**RejectAllGivenCarpoolEventsPage**
send information to **MatchingRiderController** so it can match rides for current info provided	**MatchingRiderController**
Know **MatchingRiderController**	

Class Name: InputRideDestinationPage	
Responsibility:	**Collaborators:**
Get destination input from user	**RequestInARideController**
Display map and route options to user	
Display estimated ride cost to user	

Class Name: RequestARidePage	
Responsibility:	**Collaborators:**
Display form for user to request a ride	
Send and Handle user request to place order	**RequestInARideController**
Know **RequestInARideController**	

Class Name: RejectAllGivenCarpoolEventsPage	
Responsibility:	**Collaborators:**
Display list of all carpool events for user to reject	
Allow user to select all carpool events for rejection	
Send request to **RequestARideController** to reject all selected carpool events	**RequestARideController**
Know **RequestInARideController**	

Class Name: MatchingRideController	
Responsibility:	**Collaborators:**
Retrieve available ride information from **RideInformationDB**	**RideInformationDB**
Display available ride information on **RideInformationPage**	**RideInformationPage**
Accept a rider request to join a carpool event	**RideInformationDB**, **UpdatedRidersInCarpoolEvent**
Reject a rider request to join a carpool event	**RejectRequest**
Send info to Notify **AwaitingRequestsController** of accepted/rejected rider request	**AwaitingRequestsController**
Send info to collaborate with **ProceedWithRideController** to initiate a carpool event	**ProceedWithRideController**
Match riders with available drivers	**RequestInARideController**
Know **ProceedWithRideController**	
Know **AwaitingRequestsController**	

Class Name: RejectRequest	
Responsibility:	**Collaborators:**
Allow user to reject Remove a ride request from the list of pending requests in **MatchingRideController**	**MatchingRideController**
Send and notify the user who submitted the rejected request	**MatchingRideController**
Know **MatchingRideController**	

Class Name: AcceptRequest	
Responsibility:	**Collaborators:**
Allow user to accept add a ride to request from the list of pending requests in **MatchingRideController**	
Send info to Notify the user who submitted the accepted request to **MatchingRideController**	**MatchingRideController**
Know **MatchingRideController**	

Class Name: RideInformationPage	
Responsibility:	**Collaborators:**
Display ride information to users and allow the to view available rides	
Send information about the ride so users can select and request a ride	**MatchingRideController**
Know **MatchingRideController**	

Class Name: InitiatingARideController	
Responsibility:	**Collaborators:**
Create new ride offer with inputted information	**OfferorInputtedRideInformation**
Send Request to AwaitingRideController to wait forride requests	**AwaitingRideController**
Retreieve ride offer information in RideInformationDB	**RideInformationDB**
Know **AwaitingRideController**	

Class Name: OfferorInputtedRideInformation	
Responsibility:	**Collaborators:**
Retrieve ride information inputted by offeror	
Send the info to **InitiatingARideController** to display success/error page upon successful submission	**InitiatingARideController**
Know **InitiatingRideController**	

Class Name: AwaitingRequestsController	
Responsibility:	**Collaborators:**
Send available ride requests to user on SearchingForARidePage	**SearchingForARidePage**
Initiate and track ride request in progress	**ProceedWithRidePage**
Receive Cancelled carpool event if no matches found	**CancelCarpoolEventPage**
Know **SearchingForARidePage**	

Class Name: SearchingForARidePage	
Responsibility:	**Collaborators:**
Display search form for ride requests	
Allow user to input search criteria and initiate search	
Await to display available ride requests to user	**AwaitingRequestsController**
Find a matching ride offer for user	

Class Name: ProceedWithRidePage	
Responsibility:	**Collaborators:**
Display ride details and options to proceed with ride	
Allow user to select ride and initiate carpooling process	**ProceedWithRideController**
Send Process carpool request	**AwaitingRequestsController**
Know **AwaitingRequestController**	

Class Name: CancelCarpoolEventPage	
Responsibility:	**Collaborators:**
Display ride details and options to cancel carpool event	
Allow user to select carpool event to cancel	
Send Cancelled carpool event goes through the awaitingRequestsController	**AwaitingRequestsController**
Know **AwaitingRequestController**	

Class Name: ProceedWithRideController	
Responsibility:	**Collaborators:**
Send info Collaborate with **MatchingRideController** to match riders with drivers	**MatchingRideController**
Receive Display matched riders and ride information on **ProceedWithRidePage**	**ProceedWithRidePage**
Allow driver to start the ride and update ride status	
Receive Updated ride status and reward points	**ArriveAtDestinationController**
Know **MatchingRideController**	

Class Name: ArriveAtDestinationController	
Responsibility:	**Collaborators:**
Update user's rewards points in **RewardPointsDB** based on the ride information	**RewardPointsDB**
Display the fees charged for the ride and the points earned on **DisplayFeesAndPointsObtained**	**DisplayFeesAndPointsObtained**
Send request for user to leave a rating for the ride on **RatingController**	**RatingController**
Send info with **ProceedWithRideController** to finalize the ride	**ProceedWithRideController**
Know **ProceedWithRideController**	
Know **RewardPointsDB**	
Know **RatingController**	

Class Name: RewardPointsDB	
Responsibility:	**Collaborators:**
Store user's rewards points information	
Receive Updated user's rewards points information based on the ride information	**ArriveAtDestinationController**
Retrieve user's rewards points information for display	

Class Name: DisplayFeesAndPointsObtainedPage	
Responsibility:	**Collaborators:**
Display the fees charged for the ride and the points earned	**ArriveAtDestinationController**
send Update display with the latest fees and points earned to **ArriveAtDestinationController**	
Know **ArriveAtDestinationController**	

Class Name: RatingsController	
Responsibility:	**Collaborators:**
Receive ride rating from user on **LeaveARatingPage**	**LeaveARatingPage**
Send and save the user's rating to **UserRatingsDB**	**UserRatingsDB**
Receive info and collaborate with **ArriveAtDestinationController** to be able to start the rating process	**ArriveAtDestinationController**
send user reuqest to skip rating for current ride on **SkipCurrentRatingPage**	**SkipCurrentRatingPage**
Know **SkipCurrentRatingPage**	
Know **RewardPointsDB**	

Class Name: LeaveARatingPage	
Responsibility:	**Collaborators:**
Display rating form	
Allow user to input rating and select "submit"	
Send rating to **RatingsController**	**RatingsController**
Know **RatingsController**	

Class Name: SkipCurrentRatingPage	
Responsibility:	**Collaborators:**
Display a page allowing user to skip current ride rating	
Allow user to select ”skip” to skip the current rating	
Send request to **RatingsController** to skip current rating	**RatingsController**
Know **RatingsController**	

A Division of Labour

Team Member	**Contribution**
Adam Mak	Purpose, Overview, Architecture design (3.1) and subsystem architectures (3.2), resolving NFR feedback for D1 (added SR-P3) Signed by: Adam Mak
Eric Chen	Analysis Class Diagram (ACD), Resolving BE feedback for D1 and updating changes to LaTeX document. Signed by: Eric Chen
Justin Ho	Analysis Class Diagram (ACD), Resolving BE feedback for D1, updating changes to LaTeX document, section 1.2 (system description) Signed by: Justin Ho
Ahmad Hamadi	CRC Cards, purpose(1.1), Overview(1.2), resolving feedback Signed by: Ahmad Hamadi
Kevin Ishak	CRC Cards Signed by: Kevin Ishak
Jonathan Jiang	Purpose, Structural architectural diagram, Subsystems Signed by: Jonathan Jiang