# Design Document

Logan Yue
Jolene Poulin
Thomas Maurer

**General Overview:**

Our program runs entirely in a terminal. The core navigation and central program interface of our program is contained in the file main.py. This file loads the database and imports all of the other functions from the remaining files. Then the login function is called from login.py to allow the user to login or create an account. Upon login the user has several options for calling functions from one of the other files. The offering and searching ride requests happens in the functions contained in rides.py. Booking members and cancelling bookings is contained in the file manage_bookings.py. Posting ride requests is handled by a function in post_ride_requests.py, searching ride requests is handled by a function in searchRequests.py and deleting a request you've posted is handled in deleteRequest.py.

**Design:**

Login: The login function is a basic command line interface, like the rest of the program. It takes a valid user-input username and searches the database for a matching member. If a matching member is found, they are prompted for their password and logged in upon successful password entry. The member's unread messages are then displayed. If a matching member is not found, the user is prompted to create a new account. This account creation process requires a name, phone number, and password. Each of these entries are checked and validated to prevent SQL injections and to ensure a proper format. The user is then logged in and shown the main screen.

Main: This is the main interface and connecting file of our program. It handles program navigation through command line arguments and calls the corresponding functions in the other files. It also stores the current user and database variables at runtime.

Offer a Ride: This functionality is handled by the offer_ride function in rides.py. It takes the current user and database name as function parameters and then allows the user to create a ride in the database. It also allows the user to link enroute locations and a car to this ride.

Search Rides: This functionality is handled by the search_rides function in rides.py. It also takes the current user and database name as function parameters. It then allows the user to search all rides in the database based on 1-3 keywords and then message the driver of one of these rides if the user chooses.

Search Ride Requests: When a user types ‘search_requests’, they are directed to the request search functionality. Here, they can enter a non-case-sensitive location or an exact location code to search for requests. Partial matches are not return. The system displays up to 5 requests at a time. If a user chooses to message another member about a request by pressing ‘m’, they are asked for a request number, a ride number, and a message. Each of these are validated to prevent SQL injections and ensure a proper format before being inserted to the ‘inbox’ table.

Delete Ride Request: By typing ‘delete_request’, the user is shown a list of all the requests they have made. The user can then type a valid integer representing the request number to delete it. The integer is checked for SQL injections before deleting the request. The user is then returned to the main screen.

Post Ride Request: This functionality is handled by the PostRideRequest function in post_ride_request.py. It takes the current user’s username and the controller as input and takes values from the user necessary to create a ride request; Date, dropoff lcode, pickup lcode, and cost per seat. The function then takes this information and uses it to insert a row into the requests table in the database

Manage Bookings: This functionality is handled by the manage_bookings function in the manage_bookings.py file. This opens up a second navigation page that allows the user to execute the following functionalities:

View Rides: The user may enter ‘view rides’ to show their list of rides with number of available seats five at a time

View Bookings: The user may enter ‘view bookings’ to view all bookings on rides that they have offered. From here, the user may enter a booking number of a valid booking in order to cancel a booking on their ride. The system will then notify the relevant user by posting a message to their inbox

Book a member on a ride: The user may enter a ride no. of a ride that they have posted in order to book a member on it. They then enter the user’s email, a cost per seat, the number of seats, a pickup location and a drop off location at which point, the system will post the booking and notify the specified member py posting a message to their inbox

**Usage:**

Login: Users are asked to enter an email address of a valid format (example@123.com) in order to either login or create a new account. If the address exists in the database, the user is prompted to enter a password. The password is hidden for security purposes and the users has 3 tries to enter a correct password before the program ends and asks you to try again later. Passwords are case sensitive and cannot be longer than six characters. If the address is not in the database, the user is prompted to confirm the creation of a new account, or exit the program. If the user confirms, they are asked to enter their name (valid string format), their phone number (as XXX-XXX-XXXX, all digits), and to create a password. Once the new account

creation process is complete, they are logged in as that user. After a successful login, all new messages that the user has received are displayed.

Main: Upon startup you will be asked to login. This is explained above. Once logged in, you will be asked what action you would like to perform. Possible actions include: “help, exit, offer, search_rides, search_requests, delete_request, post_request, manage_bookings” Each of these calls one of the below functions, except for help, which displays help information, and exit, which closes the program.

Offer a Ride: If the user types offer they will be prompted to supply information about the ride they would like to offer. The majority of this is straightforward and will just ask for an input of a certain type. Then the user will be asked to provide a location for source and destination. Typing in a lcode will instantly select that location. Otherwise, typing in a keyword will search all locations for that keyword and allow the user to select from one of them. Upon completion of all necessary ride information the user will be asked if they would like to add a car to the ride. If yes they will need to enter the car number and the program will ensure they own that car before adding it to the ride. They will also be asked if they would like to add any enroute locations to the ride and if yes they will be prompted to add locations just like before.

Search Rides: If the user types search_rides into the main control they will be asked to enter a keyword for searching rides. They will be prompted for up to 3 keywords with the option to use only one or two if they prefer. These will then be compared against all rides and enroute locations and any matching rides will be returned. The user then has the option to type the rno of the desired ride and message the poster of that ride.

Search Ride Requests: If the user types search_requests, they are directed to the request search functionality. The user can enter a location, such as Edmonton (non case sensitive) to find all requests posted that have a matching pickup location. Users can also enter a complete location code. Partial matches are not displayed. Up to five matching results are displayed at a time, and the user has the option of messaging another member regarding a request. Pressing ‘enter’ displays up to 5 more requests, while typing ‘m’ allows the user to send a message. If the user decides to send a message, they are prompted to enter the request number (a valid integer), a ride number (a valid integer), and a message (a valid string). This message is then sent and the search request functionality ends.

Delete Ride Request: If a user wants to delete a request they have posted, they can do so by typing “delete_request”. This displays all the requests they have posted. Upon entering a valid integer corresponding to a request, the request is deleted and the user is returned to the main screen.

Post Ride Request: If the user types post_request, they can post a ride request upon inputting the relevant information (pickup, dropoff, price, date) as it is requested by prompts

Manage Bookings: If a user wants to view their rides, book members on their rides, view bookings for their rides, and remove bookings for their rides, they can do so by entering manage_bookings. From there, there will be a prompt to enter ‘view rides’ (to view their posted rides), ‘enter a ride number’ (to book a member on a ride) or ‘view bookings’ (to view bookings for their rides). If the user input ‘view bookings’, then from there they may enter a booking number in order to cancel the specified booking.

**Testing Strategy:**

The majority of our testing strategy was a visual confirmation: test the function manually, then inspect the database to make sure the expected change took place. We began by populating a test database with some test data so we could test the functionality of our programs. Then each person built and tested their functions on their own, for any cases described in the program spec and some small error cases. Lastly we combined the files in main.py and used this to test the functionality of every part again.

**Work Break Down:**

We used Facebook Messenger to communicate and stay on track with one group meeting near the end to ensure all our functions worked together. We also created a GitHub repo to keep track of our files. We broke the project down into six parts; the 5 points and login. Each team member took two of these parts:

**Jolene Poulin:**

Worked on: Login and Search & Delete Ride Requests. (Files login.py, searchRequests.py, and deleteRequest.py)

Approximate time spend: 8 hours

Progress made: all functions in the files listed above, group work during the meeting for testing and merging all our files, further refinement and error checks

**Tom Maurer:**

Worked on: Offer a Ride and Search for Rides. (File rides.py)

Approximate time spent: 9 hours

Progress made: created functions for offer and search rides alone. Created basic structure of main file. Group work for testing and debugging.

**Logan Yue:**

Worked on: Booking Members & Cancel Bookings and Post Ride Requests. (Files post_ride_request.py and manage_bookings.py)

Approximate time spent: ~10 hours

Progress made: Completed functionality for posting ride requests (section 4) and managing bookings (section 3) as well as testing.

We all worked on the main file, adding to it as we completed our respective functions.