CMPUT 291 Mini-Project Design Document
Editors: Jose Ramirez, Curtis Goud

1 - General Overview

This application is a high-level program that regulates messages for ride members. The application offers the following functionality for users:

- Offer a ride:
  A user can offer an available ride for all members to book on
- Search for rides:
  A user can search for rides within specified locations that the user will input
- Book members or cancel bookings:
  An existing member can book on existing rides, or cancel any rides that they are booked for
- Post ride requests:
  If a member is unable to search for a ride that they want. They can simply request one
- Search and delete ride requests:
  A member can view/find any of their requested rides and delete them if they choose to

The code is organized in a basic M-V-C layout with the dataConn.py acting as the Model, commandClass.py the Controller, and project.py the view. The model handles all SQL interactions, which is sent to the controller. The controller handles logic of the SQL queries and sends/receives data to the model and view. Lastly, the view is our command line interface that the user interacts with.

In order to be able to use this application, the user must be a registered member in the database. **If the user is not a registered member, there will be a REGISTER option from the login prompt**.

The application makes many attempts at handling invalid input, but due to the nature of command line programs, complete domain coverage is impossible. The program WILL operate under its required specifications but may have unexpected functionality if used improperly beyond the specifications.

2 - Function Design

This section will explain the functionality of the applications available functions and describe the specifics as the required inputs, and their returned information. For the purpose of this

documents, we will focus on the data collection given by only the functions that interact with the database of the application. We will not go through the functions within the *main* application.

Pre-requisite information:

- A data class called “dataConn.py” holds all the functions related to inserting and querying data from the main database that the application is connected to. This will act as the “Model” of the program.
- The “commandClass.py” file is the controller for the “project.py” file. It will allow the main program to call the controller, with the controller holding functions that will handle data into the “dataConn.py” file.
- The “project.py” file is the “View” for the entire program. It handles all of the outputting of the required information requested by the user.
- The project is launched using the command “python3 project.py” database arguments will be provided inside of the program

System Structure

The structure of the entire application is as follows:

- The **login** for the application
- After **login**, the main menu will be visible to the user and they can select which main function they want. The functions are described as follows: *offerRideUI(), searchRideUI(), bookMembersUI(), postRideRequestUI(), searchDeleteRequestUI()*

offerRideUI:

The *offerRideUI()* function will allow the user to offer a ride to that will be visible to all other members to search for. The user will be required to fill in the field requirements in order to insert the ride into the database. The *commandClass.py* will be called to handle the inserting function into the database.

searchRideUI:

The *searchRideUI()* function will allow the user to search for rides throughout the database with the locations that the user is looking for. The user will be required to provide 1-3 location keywords that will be passed to the search function from the *commandClass.py* file. The *commandClass.py* file will then use functions from the *dataConn.py* file and return all the resulting rows of the search query.

bookMembersUI:

The *bookMembersUI()* function will allow the user to book members for any of the rides that they offered. The user will be able to view all of their offered rides (*commandClass.py* oddered rides function will return all resulting rows) and select their offered ride and book a member to it. The user must provide the required fields to book a member, which are then

passed to the *commandClass.py* file, which will use the insert booked member function from *dataConn.py*.

postRideRequestUI:

The *postRideRequestUI()* function will allow the user to to post a ride request by provide some given criteria. The *commandClass.py* will take in the required fields and insert them into the database from the insert functions called from *dataConn.py*. The ride request now exists within the database.

searchDeleteRequestUI:

The *searchDeleteRequestUI()* function will allow the user to view all of their requested rides, and search for other requested rides. When the user wants to view their own requests, they *commandClass.py* file will call the database and return all the requested rides the user has made (from using the query function in *dataConn.py*). The user can then select a request, and delete it (once again calling the *commandClass.py* which calls a function from *dataConn.py* that deletes the ride request from the database). When the user performs a search they are able to give single keyword to find matching ride requests with that keyword matching the substring of a location column. The user is then able to message the user of one of the returned requests.

3 - Testing strategy

The testing involved within the application requires the creation of some test data and some test functions for the *dataConn.py* file. The reason for this is that most of the main functionality for this application will come from this file, as it is the one which interacts directly with our database. Our testing strategy consisted of black-box testing techniques, we tabulated a checklist for each functional element of the program (those being the specification given on the assignment) and created a set of use cases for these components. We tried our best to achieve domain coverage when it came to plausible user input. The program does handle invalid inputs however there may be some cases this leads to unspecified functionality.

Bugs were generally only found in the UI where the user is given free reign over the input. We did our best to restrict the user from entering anything unspecified, and aborted the operation if they did so.

4 - Group work breakdown strategy

For the group-work strategy, there are just 2 of us (Curtis Goud, and Jose Ramirez). We have split up the work by the following:

- Jose was in charge of coding the UI elements of the entire system and handling prints. He will also help with creating the queries for the desired system information and handling data output to the application UI. The UI is a command line interface.

- Curtis was in charge of coding the *dataConn.py* file for getting all of the data information from the connected database. He also creates the queries required for the correct data output. He also is in charge of creating the data controller throughout the entire application.
- Both of us would be in charge of testing the data validity, the correct UI output, and controller implementation.

The amount of time taken on the entire project is as estimated:

- Curtis: 10 hours
- Jose: 10 hours

We did our best to divide the work in a way to allow us maximum parallel development. We split up the application into an M-V-C layout where the dataConn.py would be the Model, project.py as the view and command.py the controller.

This allowed Jose to fully implement UI functionality while Curtis was working on implementing the dataConn.py SQL functions. We eventually worked until we met in the middle and simultaneously worked on the command.py controller. There were some instances where we would make edits and look over each others work as we went to ensure compatibility. Both of us were able to fully implement the features assigned to us. Together we stayed on track using Github for code collaboration, and VoIP and text to communicate with each other.