Report

CMPUT291 Mini-project 1

Jarrett Yu, Michael huang, nicholas serrano

2018

# Overview

## Introduction

The program is a basic ride-sharing program that allows users to provide rides to other users and uses a database for most of its important functionality. A user may sign in or register and given that they are successful, can access the main program. From there they can select a sub menu, one for each specified function.

## Setup and Operation Instructions

The program requires at least JRE 1.7 to be installed as well as SQLite3. To run the program use the command “java Main.jar [arg]” where “arg” is the path to the database file. If no argument is provided the program will default to the local directory and use tables.db.

To navigate menus the prompts on the command line should be followed. The instructions will be indicated within quotations, brackets, or pipes.

# Design

## Design Overview

We decided to split the code such that there was a starter class, which started the code; a manager class, which managed the other classes; a login class which dealt with initial connections to the database; a helper class which contained some frequently used functions; and screen classes, one for each functionality required. This allowed for easy concurrent code development as each group member could work on a module without interfering with other parts of the code. A variable that records states is used for the management of screens and will break the main loops of classes given specified states. Prepared statements are used to prevent SQL injection attacks to prevent the need to write our own classes for input sterilization.

## Helper Classes

JDBC\_Connection is the class where we store various shared functions that are involved with connections. The functions here are used for abstracting the creation of a connection as well as functions for sending and receiving messages. The method connect() is made so that if no database is specified in the command line a default database is used from the local directory.

## Driver and Manager Classes

These classes consist of the following: Main, Menu\_Main, and Menu\_Login. The main function contains the main function which accepts 0 or 1 arguments. The argument provided is the path to the database and if none is provided will default to the local directory. Main will handle initializing the connection to the database and the creation of a Scanner object which will be used for all inputs. It then calls and instance of Menu\_Login. A public static variable is initialized here for the program to track its state.

Menu\_Login manages user logins and registrations and operates under to modes, one for each of the previously mentioned functionalities. If a user successfully logs in or successfully registers the menu will call an instance of Menu\_Main which will be the main screen and manage each part of the program.

Menu\_Main manages each sub screen and allows the user to choose which part of the program they want to access. It is built such that each sub screen only needs to connect to the main menu for full functionality of the program which simplifies collaboration and reduces dependency problems.

## Sub Screen Classes

// do stuff for module 1

Menu\_Search handles the second point in the required functionalities described in the assignment specifications which is searching for existing rides and giving the option for a user to send a message requesting to be added to a ride. This class uses a finite state machine to track what should be printed by the program and what inputs from the user it should respond to. The class’s main loop is split into 2 primary functions which is the querying step and the printing step. There is a function dedicated to receiving user inputs, one to searching the database, one to printing a page (5 results), one to process post-query inputs, and one to send a message. If a function operates successfully it changes its state to the next function needed, else it reverts and requests the user tries again. The function links back to only the main menu by breaking out of its primary runtime loop.

Menu\_PostRq handles the fourth required functionality in the assignment spec. This module asks the user to enter all relevant information and generates a unique rid, then inputs into the requests table. This module returns to main menu after an input is made to the database.

Menu\_ManageRq handles the fifth required functionality in the assignment spec, which is to search for existing requests when given a location code or a city, and be able to message a user from the returned row; or the user will be given a list of their own requests and they will be able to delete a request. This class also uses a finite state machine to track what the program should do. The class’s main loop is split into the search function or the delete function. The search function has a dedicated state to message another user; while the delete function has a state to delete a request from the database. If the program encounters an error, it will be push to the main class menu, or the search or delete menus. The function links back to the main menu only by breaking out of its primary runtime loop.

# Testing

We built a small set of test cases which contained 4 members, 6 rides, 1 request and some various pieces of data. This set of data is good for testing searches since it gives more than 5 rides for a search term. A single enroute is also included to test if searching will cover enroute options.

For testing logging in and registering we created a new user as well as tried using a basic ‘1’=’1’ attack in the email field to test that our sql injection prevention was working. Login’s normal functionality would be tested by logging in when testing other functions.

Two notable bugs we ran into was a instruction stack bug and a SQL write-lock bug. The instruction stack bug one where we had an ever-increasing stack of classes causing a situation where the main menu would appear even after selecting the quit option. This was resolved by changing the way classes would return to the main menu class. The SQL write-lock bug was where sqlite would refuse an insertion statement due to a lock exception. We found that this was due to how we tested our code in Eclipse and stopped the program without properly exiting the program. This then bypassed the Connection.close() call. Since sqlite still thinks the connection is open it doesn’t allow another process to request the write-lock and when we tested the code again, it would refuse the request.

# Production Organization

We split the project so that one person had to work on the functionalities of one of the sub screens, as well as build the main menu system. They also worked more on the report since the implementation of the main menu was overall simpler than the sub screens. The other two group members instead worked on two sub screens each. Communication was through in-person speech or though a group-chat. Most of the time work was independent unless a member wanted to discuss a system or change part of the design.

Jarrett worked on the initial code which involved the driver class, login class, and the main menu class as well as the search sub menu. These driver classes were made earlier in the project to allow easy collaboration for the sub menus. He also wrote some helper functions for connecting, sending messages, and reading messages. Since writing the driver classes was generally easier than coding a sub menu he also handled writing most of the report with the design of each sub menu and testing of the sub menu up to the member that coded it. Estimated time working ~13 hours.

Michael worked on the sub menus for posting, sharing, and deleting ride requests. He also revised the driver classes after recommending the usage of prepared statements to counter sql injection.

Nick worked on the sub menus for offering and managing rides.