CMPUT 291 Mini-Project 1 Report

Overview

The purpose of this project was to learn how to use SQL in a host programming language. A system is designed that can store enterprise data related to a ridesharing platform.

Mini-Project-1 is a python command-line application that interfaces with and SQLite database using the sqlite3 python package. Using this program, users can offer a ride, search for rides, book members or cancel bookings, post ride requests, and search and delete ride requests. All commands are executed through the command-line. Users are appropriately notified when input is required or when their actions are invalid or incomplete.

The database schema for this project is defined as:

- members(email, name, phone, pwd)
- cars(cno, make, model, year, seats, owner)
- locations(lcode, city, prov, address)
- rides(rno, price, rdate, seats, lugDesc, src, dst, driver, cno)
- bookings(bno, email, rno, cost, seats, pickup, dropoff)
- enroute(rno, lcode)
- requests(rid, email, rdate, pickup, dropoff, amount)
- inbox(email, msgTimestamp, sender, content, rno, seen)

User guide/Detailed design

Note: Our official user guide and documentation for mini-project-1 is located at: https://mini-project-1.readthedocs.io/en/latest

Shell - Shell uses the Python Cmd module to create a line-oriented command interpreter. The shell.py file represents the main command loop of the entire application. This allows the user to interface with the application as if it was in its own shell. The shell is opened using the command "mini-project-1 [-h] [-r] [-i|-d] <db name>" where the -h flag will output the command usage help, the -r flag is used to register a new user (more information in section 3 below), the -i flag is used to initialize a new database given by <db name>, and -d opens an existing database given by <db name>. Once opened, the shell will require the user to log in (more information in section 3 below). More information for these can be found in section 3 below. Once logged in the shell exposes all methods to the user:

- **book_member** – Used to book members on rides that a user offers. The user is able to book a member by specifying the member's email, the number of seats booked, the cost per seat, the pickup location code, and the drop-off location code. The system warns the user if the ride is being overbooked. Once booked, the booked member will be sent a message notifying them of the booking. Used as "book_member <email> <seats> <price> <pickup> <dropoff>" where email is the member to book's email, seats is the number of seats booked, price is the price per seat, pickup is the pickup location code, and dropoff is the drop-off location code.

- **exit** – Exits the application shell. Used as a single command with no arguments.

- **offer_ride** – Allows the user to offer a new ride by providing the date of the ride, the number of seats offered, the price per seat, a luggage description, the source location, and the destination location. Optionally, the user can add a car number and a list of enroute locations. Used as "offer_ride <date> <seats> <price> <luggage> <src> <dst> [--cno #] [--enroute [list]]" where date is an ISO 8601 formatted datetime string for the date of the tide, seats is the number of available seats, price is the price per seat, luggage is a description of the luggage, src is the source location, dst is the destination location, --cno # is n optional integer representing the car number of the car performing the ride, and --enroute [list] is a space-separated list of enroute locations.

- **search_requests_city** – Allows a user to search all ride requests that are in the database based on a provided city name (case-insensitive). Used as "search_requests_city <city>" where the city is the name of the city to search by.

- **select_request** – Allows a user to select a ride request entry. Once selected, the user is asked if they would like to message the owner of the ride. This message is sent to their inbox. Used as "select_request <rid>" where rid is the request ID.

- **cancel_booking -** Used to cancel bookings on the rides that the user offers. Once deleted, a message is sent to the member whose booking was deleted. Used as "cancel_booking <bno>" where bno is the booking number of the booking to delete.

- **help** – Prints documentation about a specific method. Used as "help <topic>" where the topic is one of the methods in this list.

- **post_request** – Posts a new ride request with a date, a pickup location code, a drop-off location code, and the amount willing to pay per seat. Used as "post_request <date> <pickup> <dropoff> <price>" where date is a ISO 8601 datetime string, pickup is the location code for the desired pickup location, drop-off is the location code for the deisted drop-off location, and price is the amount a user is willing to pay per seat. Checks are made to ensure that the ride request is valid before it is added to the database. Errors in this process are communicated to the user and not committed to the database.

- **search_request_lcode** – Allows a user to search all ride requests that are in the database based on a provided location code (case-insensitive). Used as "search_requests_lcode <lcode>" where lcode is the location code of the pickup location to search by.

- **delete_request** – Deletes a ride request that the user has made. Used as "delete_request <rid>" where rid is the request ID of the request to delete. If the ride request does not belong to the user trying to delete, the request is not deleted.

- **login** – Used to log in after the user has logged out. If the user is currently logged in, the user will be notified that they are logged in. Used as a single command with no arguments.

- **register** – Used to register a new user. Used as a single command with no arguments. Once the command is executed, the user is prompted to specify an email, their name, their phone number, and a password.

- **search_rides** – Allows the user to search and retrieve all rides based on up to 3 keyword arguments (case-insensitive). A ride is matched if it matches all specified keywords where the keywords correspond to location codes for pickup, drop-off, or enroute locations. If there are more than 5 matches, only 5 are shown at a time. When being displayed messages, the user is able to select one of the presented matches and message the member who posted the ride. This message is added to the poster's inbox. Used as "search_rides <term1> <term2> <term3>" where term1, term2, and term3 are keyword location codes.

- **list_bookings** - Allows the user to list the bookings for their rides. Used as "list_bookings".

- **list_requests** - Allows the user to list their ride requests. Used as "list_requests".

- **logout** - Used to log out the user. Used as "logout" with no arguments.

- **show_inbox** - Displays the users inbox. Used as "show_inbox".

Common – The common module contains utility functions that are used internally in the project to perform common actions. Examples of such functionalities include validating that an input conforms to the database schema, defining exceptions, and formatting prices or dates.

Account creation and login - To register a new account. The user must specify the -r flag when running the "mini-project-1" function. When this is performed, the user is prompted to create a new account. Once created, the user will then move to the login screen. The login screen is opened by default when the -r flag is not specified. The user is prompted to login with their username (email) and password. The login prompt loops until a valid username/password combination is specified.

Testing strategy

Our primary testing strategy was to employ white-box unit testing using the pytest module. Our unit testing would validate the operation of all functions (internal and external) by comparing them to expected outputs.

Our testing efforts can be found in the https://github.com/CMPUT291PROJECT1F18/Mini-Project-1/tree/master/test/unit subdirectory of our code repository.

To keep our master branch stable, a continuous integration service was employed. Our CI was run for every pull request to ensure that our code maintained its quality and stability. Past and present testing results can be seen on our Travis CI build history at:
https://travis-ci.org/CMPUT291PROJECT1F18/Mini-Project-1

Breakdown of work

Nathan Klaptstein (nklapste)
- Hours: ~20 hours
- Progress: Provided most of the groundwork and organization for the project. Defined the project structure and user controls. Developed functionality for login, registering a new user, showing the user inbox, help documentation, and system logging. Also contributed by writing unit tests for the system.

Ryan Furrer (rfurrer)
- Hours: ~16 hours
- Progress: Developed functionality for booking a member, offering a ride, and searching rides. Also contributed by fixing bugs and writing unit tests for the system.·

Thomas Lorincz (tlorincz)
- Hours: ~16 hours

- Developed functionality for searching ride requests (by city name and location code), deleting ride requests, selecting a ride request and sending a message, listing requests, and listing bookings. Also contributed by writing the project report.

Work coordination

We primarily coordinated through GitHub. An organization was set up for the project (CMPUT291PROJECT1F18) and each task or bug was turned into an issue ticket These tickets were assigned to members and each pull request was associated with an issue. To measure progress, and when an issue was done, the assigned member would close the issue. This workflow increased productivity immensely and did not result in errors (all users were responsible with their git practices).

Communication was performed regularly over email and text messages. Once a week, at least two group members would work together on the project (the other user usually worked remotely if they were unable to join in person).

Design decisions differing from requirements

Python was used for its quick development time and rich libraries. Using libraries such as the built-in cmd library and the datetime library pendulum helped speed up development time by avoiding re-creating the wheel. Documenting tools such as sphinx were used to create automatic documentation on the project. Testing tools such as pytest and their respective addons pytest-cov provided useful unit testing tools and test statistics.

It was not explicitly specified whether one or two functions should be used to handle searching for a ride request (by location code or city). So, two functions were created to simplify the implementation of this functionality.

The register/login functionality was difficult to perform with our command-line shell (using Python Cmd module). The specification describes a login/register screen or prompt that the user must interact with. In our implementation, registration is performed by specifying the optional “-r” flag when calling “mini-project-1” entry point function. Once registered, the shell moves into the login function which will loop until a valid username/password combination is entered. If the user already has an account, they will move into the login function after calling the “mini-project-1” entry point function without the “-r” flag.