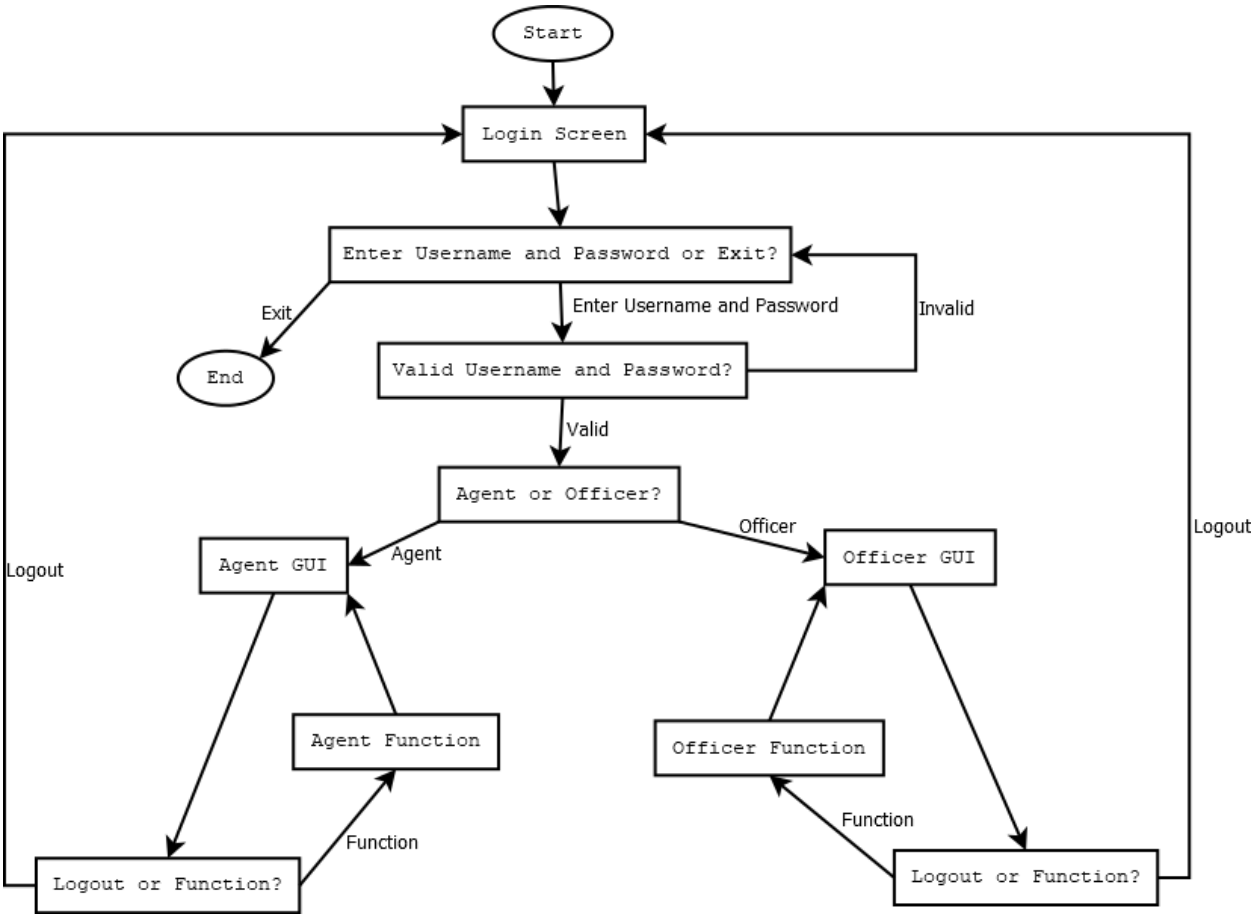
General overview

The goal of this project is to implement sql within other programming languages to develop an application. To achieve these goals, a set of tables was provided with specifications of functionality for the application

To run the program, you will run UI.py. The program will provide a User interface through tkinter. On the user interface, there is a prompt to input a username and a password. If the username or password are invalid, the program will prompt the user to input a valid username or password. Once the user has logged in, the program will display different functions dependent on whether the user is an officer or an agent. An officer will have access to two different functions while the agent will have access to six functions. When clicking on any of the functions, it will display a new UI for the user to input data. Any function with specific input requirements will be displayed as well in the UI. (ie. Any date must be input in the form YYYY-MM-DD)

Detailed Design of Software



What each file does:

backend.py – the main function for all programs that run the queries

agent.py – contains all queries that run the functionality specifications for agents

officer.py – contains all the queries that run the functionality specifications for officers

create.py – contains functions that run simple queries that insert rows into specific tables

validation.py – contains utility functions for confirming whether certain rows or columns in tables exist

AgentGUIs.py – contains the GUI classes for each selection in the Agents menu, each one runs when it's option is selected.

lowerAgentGUIs.py – contains GUI classes for sub screens from AgentGUI's classes

OfficerGUIs.py – contains the GUI classes for each selection in the Officers menu, each one runs when it's option is selected.

UI.py – The main file, run in the command line with python3 and the database as an argument, contains the login GUI and the GUI's for both occupations menus.

Testing Strategy

We developed our own data set, data.sql, by inserting values into our data as we progressed through the functions. When creating the data set, we accounted for the specifications that were required for the applications functionalities (ie: having a car registration already be expired and accounting for that in renew vehicle)

Dynamic testing: as we wrote a function, we would test it as we completed it, and come up with as many test cases as possible to try and find any errors. Once the function worked the way we wanted, we would move on to the next query. If new errors arose in previous functions, we would go back and fix issues and retest the query

Group work strategy:

https://github.com/Derkson/cmputProject1/commits/master

Chris Pontikes worked on the query and general functionality of the program; Josh Derkson worked on the user interface.

Chris Pontikes: 46 commits, 1030 additions, 652 subtractions

Josh Derkson: 18 commits, 1364 additions, 427 subtractions

Most of the work was done together, while both partners were in the same place in order to discuss any issues that arose, very little work was done when both group members were not together.

Files each person worked on:

Chris Pontikes

backend.py, agent.py, officer.py, create.py, validation.py

Josh Derkson

UI.py, AgentGUIs.py, lowerAgentGUIs.py, OfficerGUIs.py

Total Hours working

Chris Pontikes: 28

Josh Derkson: 25