Registry App

General Overview and User Guide

Registry App is a command-line application that allows registry agents and traffic officers to perform their work on computers. This would be a massive upgrade from the traditional pen and paper approach and would drastically increase the work performance of the users. On top of this, database administrators can easily manage backup and control access, which makes ensuring the availability and security of private citizen data easier. Also, with a computerized system, all agents and offices across the province can access the same data, making mailing hard copies of paperwork unnecessary. This not only saves time but also trees. Hopefully, this would help in slowing down global warming that is increasingly becoming an issue of our modern society.

After starting the application, the user can log in with their username and password. For creating accounts, please contact the database administrator. An appropriate menu will appear after a successful login. Registry agents and traffic officers can then perform their work. The command-line interface is a wizard-like layered menu which will guide users in performing their tasks. This application features validations that prevent user mistakes from corrupting the database, however, we still recommend users to double-check their inputs since mistakes such as typos in a person's name cannot be automatically detected.

Python 3.5 is required to run this application, to start, execute:
`python3 registry.py <path to database file>`

Software Design

This application heavily relies on Sqlite3, and most of the computation is offloaded to the Sqlite3 library. Since Sqlite3 does not do type checking, input validation needs to be handled by the application itself. Besides this, most of the code is implementing the wizards and layered menus. Multiple files were used to adhere to the design practice of separation of concerns, the DRY, or Don't Repeat Yourself principles. Shown below are the major components of this application:

- registry.py
- registry_agent.py
- registry_officer.py
- input_util.py

Since we are offloading most computation to Sqlite3, our application is mostly linear and procedural. Although we did use classes and objects, they are really just singletons without local states. Most of the code are while loops implementing different levels of menus. Below are detailed descriptions for each component.

registry.py: This is the entry point of this application, it handles login and the main menu. It also parses command-line arguments to handle opening the appropriate database.

registry_agent.py: Similar to the component above, this component handles all functionalities for registry agents. This includes all submenus and wizards.

registry_officer.py: Similar to registry_agent.py, this component handles submenus and wizards for functionalities for traffic officers.

input_util.py: Handles common input validation. This includes reading integer, date, string, and name.

The major data flow between the files occurs from registry.py towards registry_agent.py and registry_officer.py. The database cursor and connection object are passed in the constructors of the classes and set as a private class variable. This is so that when registry.py requires the functionality of registry_agent.py and registry_officer.py, it doesn't have to provide the connection and cursor every time. As mentioned above, registry.py is the entry point. It provides the user with the main menu, and when the user selects an option, it will call the appropriate function from registry_agent.py or registry_officer.py. All these files make calls to input_util.py for common input validation.

Security is our first priority, so when a user types in a password, it is hidden to protect the user's privacy and to ensure the security of the system. The parameter binding feature of Sqlite3 is used instead of naive string substitution, this ensures that SQL injection attacks are not possible. Passwords are stored unencrypted in the database as requested by the database administrator. We highly recommend the use of a salted hashing function like Scrypt to secure passwords in the database before deploying this application into production.

Testing Strategy

Since most computation is handled by Sqlite3, we decided to do manual testing. Manual test cases were created from reading the assignment specifications, and other test cases arose from clarifications and some creativity.

Our test cases follow BDD, or the Behavior Driven Development principle, where we test our application based on expected behaviour. We did this throughout development as well as final testing before submission. Passing and failing tests are tracked through GitHub and fixes are double-checked by a different team member.

About 40 test cases were developed for registry agent related functionalities and 15 test cases for traffic officer-related functionalities. Also, about five test cases were created to test the login flow and the main menu. This resulted in us finding about 6 bugs, which mostly had to do with input validation.

Work Break-down

We use a git repo hosted on GitHub to manage our code and track our progress, and Facebook Messenger for other communication needs. We all spent approximately the same amount of time on this project, about 10 hours each. The tasks are distributed as follows:

- Project setup
    - Arun Woosaree:
        - Setup repository and commit hooks
        - Refactoring to remove usage of Python features unavailable on the lab machine due to outdated packages
        - Makefile, Python virtual environment, code quality tooling
    - Hai Yang Xu:
        - Automation scripts and project report Google Doc
    - Tamara Bojovic:
        - Python boilerplate and login interface
- Project design documentation and report
    - Arun Woosaree:
        - Review and editing
        - Adding notes about security
    - Hai Yang Xu:
        - Initial draft
    - Tamara
        - Review and editing
        - Adding notes to software design
- Registry agents functionalities
    - Arun Woosaree:

        - Register a birth
    - Hai Yang Xu:
        - Process a bill of sale
        - Get a driver abstract
    - Tamara Bojovic:
        - Register a marriage
        - Renew a vehicle registration
        - Process a payment functionalities
- Traffic officers functionalities
    - Hai Yang Xu:
        - Issue a ticket
        - Find a car owner functionalities
- Testing
    - Arun Woosaree:
        - Create manual test plan
        - Carry out manual testing
        - Fixing bugs
    - Hai Yang Xu:
        - Fixing bugs
    - Tamara Bojovic:
        - Carry out manual testing
        - Fixing bugs