EECS 448 Final Project

**Encrypted Account and Password Management Tool**

Gehrig Keane: 2727430

Kurtis Slagle: 2742806

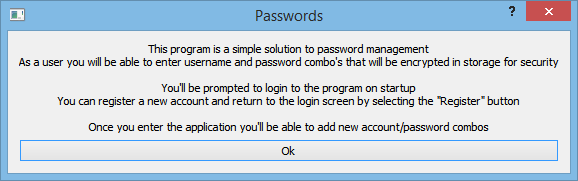
Philip Wood:2631449

**Updated Proposal Items**

* The user class mentioned in the proposal was phased out with the implementation of the database. It isn’t necessary to maintain a class based on database information when the database manages entries and the like for us. Reading in database entries and storing them in an object would increase overhead and areas for gaps in security.
* We’ve selected SQLite as the database methodology for this assignment, Qt (the graphics framework we are using has an SQLite API for database interaction without network support. SQLite was selected due to its friendly interaction with Qt and its simple easy to use command line functionality.
* Qt was the selected GUI implementation which uses C++ source, python was not used.
* The projects has not been packaged into a closed source executable due to licensing issues.

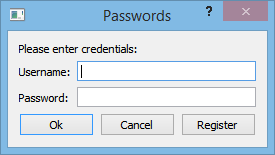
**Discussions on GUI design**

**Initial Information Screen**



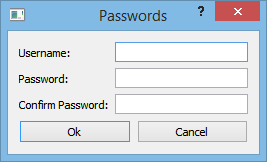
When the program begins, the user will be given a prompt informing them that the program is a password management system, as well as the basic aspects of how it works. The user can either press OK to continue or the red X in the upper right corner. Either will result in the user being sent to the Login Screen.

**Login Screen**



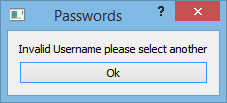
The login screen allows pre-existing users to login to their account by typing in their correct username and password. Once they do so, pressing the OK button will process the username and password provided, checking to see if it matches an entry in the database. If the match is successful, then the user will be sent to their Password Manager page. If not, then the user will be sent to the Incorrect Password page. Pressing Cancel, or the red X in the upper right, on the Login Screen will exit the program. Pressing Register on the Login Screen will send the user to the Register User page.

**Register User**

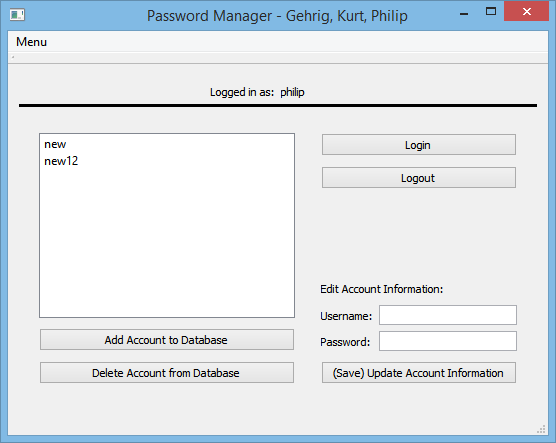


This screen will allow the user to type in a username and password for a new user account. If the user types in a username that is already in the database, then the Invalid Username page will be displayed. Once the user has created a new username and password they will be sent back to the Login Page.

**Invalid Username**



**Password Manager**



The Password Manager page allows the users to view and manage their personal passwords. At the top of the screen the program identifies whose account we are logged in as; in this example we are logged in as Philip. At any point the user can select the Login or Logout buttons to switch users and logout of the current account that they are in. This will result in the user being sent back to the Login Screen. To save a new password, the user may input a username and password into the two fields at the bottom right of the screen and then press either the Update Account Information button of the Add Account to Database button. This will update the list field on the left of the screen with the new username added on, and add the new username and password to the database. At any time the user may select a username in list field on the left of the screen. The username and password for the list item selected will be displayed in the username and password fields at the bottom right of the screen. Once a list item is selected, the user can also choose to delete that item from their account by selecting the Delete Account from Database button. The list will then be updated, and the selected element will no longer be present. The user may exit the program at any time by selecting the red X in the upper right of the window.

**User Stories and Tasks**

A user’s day to day life requires recalling multiple account names with unique passwords in a timely fashion. Often menial accounts for services that aren’t commonly used must be recorded.

User Story 1:

* As a user looking to manage his/her account-password pairs, I’d like to log in to the management software securely. Authentication of the login credentials must be encrypted and queried with the software’s database.

Task 1:

* The login dialog associated with logging into the account management software must authenticate the login credentials with a universal encryption scheme (RSA encryption). This requires encrypted database queries in the authentication function.

User Story 2:

* As a user new to the account management software, I’d like to be able to register at the login prompt to start managing my own account-password pairs. Registering a new account with the software requires querying the database in a secure encrypted fashion, to augment the user base.

Task 2:

* Registering an account with the account management software requires a new dialog with a username field and two password fields for password confirmation. Upon acceptance any invalid proposed username and password combinations would result in an instruction dialog clearly informing the user of their error.

User Story 3:

* After I’ve logged in to the account management software, I’d like to be able to add/delete/edit account-password pairs. Addition/Deletion/Edition of a user’s data must be handled securely and all information in the database must be encrypted for the sake of privacy.

Task 3:

* Upon logging into the account management software the account-password pairs must be displayed to the user in a clean fashion such that the user can select different accounts and edit them appropriately. This may include adding a new account, deleting old accounts, or updating the information for current accounts. On the backend encrypted queries communicated with the database.

User Story 4:

* As a user who’s logged into the account management software, I’d like to be able to logout of the software.

Task 4:

* The user must be able to logout of the account management software and efficiently save the modification to their account.

**Testing Processes and Traceability Matrix**

Task 1:

* Inputs: User Credentials
  + Test for validity
    - Invalid credentials do not gain access
  + Credentials sent to database are encrypted before sending
* Outputs:
  + Prompt if user entered invalid credentials
  + Console output logging the encrypted database queries to ensure only encrypted data is transmitted

Task 2:

* Inputs: New User Credentials in the “Register” Menu
* Outputs:
  + A dialogue box informing the user of any error
  + Users that already exist cannot be re-registered

Task 3:

* Inputs: Selected account from database or new account credentials
  + Account Selected From List
    - Selected account details shown to user in plain text without error
    - Account details altered and alterations persist
  + New credentials entered (account entered into database)
    - The account created in database, encrypted, and exists between executions
  + Account deleted
    - An account is selected and deleted. The deleted account never returns without being added again by the user
  + All database queries are encrypted. Data is only decrypted when being displayed on the screen

Task 4:

* Log Out
  + Click “Logout”. Software does not show any user data anymore, and a user must log in or register to use the software

**Requirements Traceability Matrix**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RTM | Defect | Critical Failure | Repaired Failure |  |  |
|  |  |  |  |  |  |
| Test Case | Scenario | Steps | Test Data | Result | Defects |
| 1 | Log In | 1) Open Application 2) Enter Login Credentials | id = kurt pw = kurt | Pass |  |
| 2 | Add Account to Database For Existing Users | 1) Enter Account Details 2) Click "Add Account To Database" | id = hello2 pwd = 123123 | Pass |  |
| 3 | Maintain New Account Data after logout | 1) Log out 2) Log In 3) Check for hello2 account |  | Pass |  |
| 4 | Check Account Persistence Between Application Launches | 1) Log In 2) Add Account 3) Log Out 4) Log In 5) Check for New Account | id = hello2 pwd = 123123 | Pass |  |
| 5 | Register New User and Add Account Info |  |  |  |  |
| 5a | Register New User | 1) Register New User | id = new pwd = new | Pass |  |
| 5b | Add Account Info For New User | 1) Log in as new user without  any into in database 2) Add account info | id = 444 pwd = 444 | Pass |  |
| 6 | Register Users after Log in and Log out of other user | 1) Log In 2) Log Out 3) Register New User | id = Hello pwd = Hello | Failure | Crash\_01: Program crash upon second account creation <New Account is successfully created and saved> |
| 6 | Register Multiple Users In One Sessions or after Log in | 1) Register New User 2) Log In 3) Log Out 4) Register New User | id = Hello pwd = Hello | Pass | Crash\_01 Repaired |
| 7 | Attempt to register existing User | 1) Click Register 2) Enter credentials for existing user 3) Expect register failure |  | Pass |  |
| 8 | Attempt to read database without decryption | 1) Access database with a database viewer without decrypting data | Expect incomprehensible Information | Pass |  |
| 9 | Delete accounts from database | 1) Log in 2) Delete existing account 3) Log Out 4) Log back in 5) Ensure account is deleted | <delete any existing account info> | Pass |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Major Tasks | Days |  |  |  |  |  |  |  |  |  |  |  |  |  | Personal Responsibilities |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |  |
| Post - Documentation |  |  |  |  |  |  |  |  |  |  |  |  |  |  | KS, PW |
| Login - Authentication |  |  |  |  |  |  |  |  |  |  |  |  |  |  | GK |
| Encryption/Decryption |  |  |  |  |  |  |  |  |  |  |  |  |  |  | KS |
| Account Management |  |  |  |  |  |  |  |  |  |  |  |  |  |  | GK, PW, KS |
| User Interface |  |  |  |  |  |  |  |  |  |  |  |  |  |  | GK, PW, KS |
| User Management |  |  |  |  |  |  |  |  |  |  |  |  |  |  | GK |
| Database Interface |  |  |  |  |  |  |  |  |  |  |  |  |  |  | GK, PW |
| Initial Draft of Concept |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Tweaks | KS = Kurt Slagle | | | |  |  |  |  |  |  |  |  |  |  |  |
| Partial | GK = Gehrig Keane | | | | |  |  |  |  |  |  |  |  |  |  |
| Completed | PW = Phillip Wood | | | | |  |  |  |  |  |  |  |  |  |  |

**Accuracy, efficiency, friendliness, and visual communications**

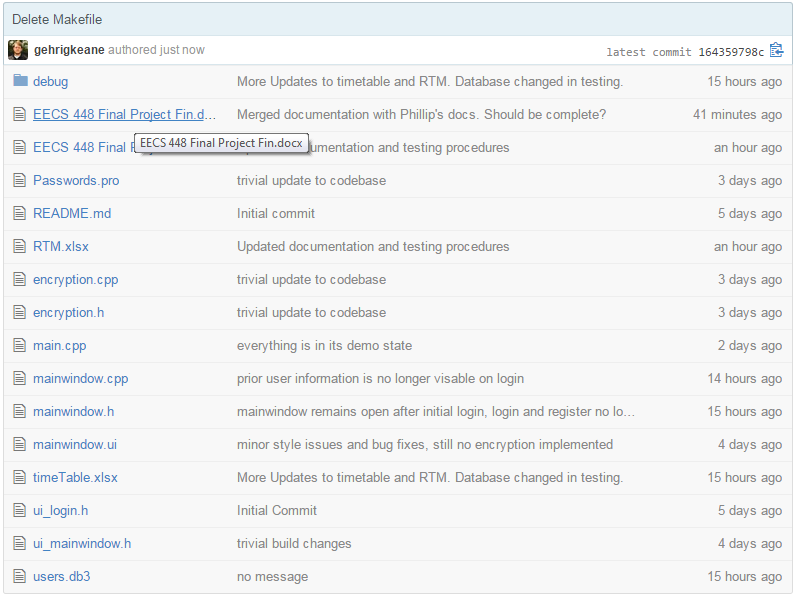
The implementation of the account management suite designed in our final project provides accuracy, efficiency, friendliness, and effective visual communication in the following ways:

* Accuracy: the major point of interest for accurate calculation within our software falls on the encryption scheme. Our software accurately takes standard strings encrypts and stores, then decrypts and displays said strings without loss of the user’s information. These operations are performed in real time with the user actions as will be discussed below, but even so our storage and retrieval function flawlessly as far as accurate data retention. Furthermore, due to the method of SQLite implementation if the program crashes queries often finish uninterrupted providing another level of information security for users and their data.
* Efficiency: the efficiency associated with our permanent storage solution and the queries associated with said permanent storage are the most pertinent to discussions of efficiency. Our database is held in a db3 file which is managed by SQLite queries, meaning queries on a largely populated data set of users and account-password pairs will never take longer than tens of milliseconds even when queries are performed in real-time per the user’s direction. With a data set the size of our current database file queries are generally less than one millisecond.
* Friendliness: We’ve designed a dialog class (the invalidDialog function) for the sake of friendly user communication that displays a simple dialog with a text message and an ok button. We feel this often instructional dialog helps users that aren’t familiar with the software gain insight into their mistakes or lack of understanding when working with their account. These instructional dialogs inform the user if their passwords don’t match when registering, it also informs the user if the entered login username isn’t found in the database, et cetera. Furthermore, we believe the UI in the MainWindow class clearly and in a simple fashion displays all the information associated with a user’s account.
* Visual Communication: As afar as visual communication is concerned, we tried to make the MainWindow class as intuitive as possible. Criterions for this process include adding redundant buttons such that the user can find an option that may or may not be apparent under a different name. Exempli gratia: the login and logout buttons are functionally identical, but the user may not be familiar with the program and thus may not notice this fact in our programs functionality. Additionally, the save and add buttons function much in the way previously stated, and again this helps new users work with the application. Visually we tried to minimize the application and reduce items that may be cumbersome to the eye. Finally, the use of layouts (Built into the Qt form implementation) allow scaling for clear display when the window is resized thereby increasing the scope of intuitive visual communication.

**GitHub Usage**

*https://github.com/gehrigkeane/EECS-448-Final-Project*

There were a total of ~47 commits and no major forks for the sake of simplicity.



**Concluding Remarks and Future Extensions**

**Concluding Remarks:**

The final implementation of our projects successfully fulfills all of the requirements that we defined. It allows users to log into their specific account as well as register new accounts. As intended, users are able to view and edit all passwords within their account. Users may switch between accounts by simply logging out, and logging into another account. All information stored to the database has been encrypted, and thus will remain secure. Information in the database is successfully decrypted so that the user may see their information when they choose to do so. As a whole, our program is an all in one package that provides a safe and easy way to store passwords and usernames.

**Future Extensions:**

In the future we will look to provide further options for users when managing their account. These may include:

* Allowing the user to provide context for each password outside of the just the username and password
* Integrate as a web extension to allow easier access to our application
* Include auto completion based on the website that the user is currently accessing

**References**

http://doc.qt.io/

This resource provides the Qt reference guide with simple example and implementations of Qt functionality

https://github.com

Github was used to host the project files for collaborative development of the source files as well as the documentation and build assets

http://zetcode.com/db/sqlite/

zetcode provides tutorials on common programming paradigms, in our case we used zetcode’s tutorial as an SQLite query reference.

http://stackoverflow.com/

stackoverflow was useful in solving specific questions with regard to syntax and Qt-SQLite implementations