CS307 – Software Engineering
Design Document

Sentinel Data Vault

Team 19

Jiho Choi, Zhaoji Jiang, Adam Petty, Dingfun Sun,Thomas Worns

TABLE OF CONTENTS

PURPOSE

Today, people have countless numbers of device passwords, website logins, and a wide variety of other hard-to-memorize personal information. Keeping and remembering all of these is an extremely difficult task, especially considering that in order to be secure, each one of these items must be lengthy and complex. Our system, the Sentinel Data Vault, is a data manager that will store and track all of these items of personal information in a clean and simple interface, requiring only the vault login to quickly access a user's stored information. Our team will implement this system to eliminate the need to memorize a multitude of information, and allow swift, secure access to a user's personal information.

1. **Account Management**
    a. Users can create new accounts for the data vault, which includes a user name, login email, password, and password hint. Entering the login email and password upon application launch will grant a user access to the entirety of their stored information within the data vault so that it becomes the only login info they will ever have to memorize.
    b. If a user forgets their data vault password, they can see a hint that they've set during account creation, or be given the option to reset their password via the user login email.
    c. Users can completely and securely delete their account(s) and its associated data.

2. **Data Management**
    a. Users can store different types of information in the data vault, which includes: website logins, device passwords, application passwords, credit/debit/gift cards, Wi-Fi networks, licenses, passport, ID cards, PINs, SSNs, account numbers, phone numbers, garage/door entry codes, receipt/confirmation numbers, serial numbers, shipment tracking numbers, travel tickets, coupons, insurance card, and prescriptions. Each data entry will have a type, a name, and the relevant information.
    b. Users can permanently delete data entries from their vault.
    c. Users can access a password generator that will take as into account desired length, special characters, case, punctuation, and character redundancy. The generator will also display the strength of the generated password.
    d. Users can see the strength of passwords entered into data entries for specific types (ones that have password fields). The data vault will track the strength of all entered passwords, calculate the average, and report to the user an overall security rating.

3. **Security**

a. All data entries will be encrypted, and data types deemed highly sensitive (such as financial information or SSNs) will be more strongly encrypted. Users will indicate this when they create an entry, where they will set a security level toggle between normal or extreme encryption. For users that want the strongest security on all entries, a settings option will be available to make this the default.
b. Users can set up reminders for individual passwords to be changed to encourage stronger security, as well as the option to adjust the time between reminders.
c. Users will have protection against brute force attacks with the option to lockout their account from further access after a user-specified number of failed login attempts. Each successive lockout will last for an increasing amount of time, eventually maxing out at a week-long lockout (this can be adjusted by the user). When a lockout occurs, the user will be notified via email. Optionally, the user can set it so that their entire account is securely deleted after a certain number of lockouts (to be determined by the user).
d. Users can turn on two-factor authentication upon login to gain access to their account to further increase security. After entering their normal vault login and password, a 6-digit code will be sent to their account email. This code must also be entered into the vault at login in order to gain access to stored information.

4. Sharing

a. Users can create a shared folder and specify who is allowed to access its data entries. All users with permission to open the folder will be able to see all its content, but not edit the entries. Only the user who creates the shared folder is able to set the permissions.
b. Users can share a single data entry of any type with another user via email. The email will be sent to a user's data vault login email address and contain the encrypted information along with a decryption key. The receiving user will enter these data into their own vault, which will decrypt and display the shared entry. Only the user specified can decrypt the data.

5. Backups

a. Users can back up all their stored information in an encrypted file. This file can also be used to import and export their account information between computers, such as when getting a new one or setting up their data vault on a second machine.
b. Users can specify the location of their encrypted backup file using their operating system's file browser.

6. User Interface

a. UI will be intuitive and user-friendly, utilizing mouse navigation and a

graphical interface similar to that of the common file browser, with a tri-column layout of a category sidebar, entry list, and entry preview (see UI mock-up Design Details section).

b. Users can easily find and view their stored data entries. They can sort and search data entries according to name and type.
c. Users can copy out information from data entries into the OS clipboard for easy use of info.
d. When viewing data entries, users will have a hide/show toggle for highly sensitive data fields, such as passwords, PINs, or codes. The toggle will alter the fields between text and black dots.
e. Users can enter information using a built-in virtual keyboard, giving them the option to avoid physically typing on their keyboard for data entry.

DESIGN OUTLINE

The Sentinel Data Vault will be a desktop Java application that allows users to securely store sensitive personal information. The user will interact with the system via a file browser type GUI. Our system will implement a multi-layered design architecture. The system needs a user interface layer, an encryption layer, an application logic layer, and a data storage layer.

1) **The User Interface Layer**
    a) Primary user interaction point
    b) Browser will display stored information
    c) UI features will correspond to actions the user can perform
2) **The Application Logic Layer**
    a) This layer will move data between encrypted storage and decrypted data going to the user interface
    b) This is the only layer that actively handles unencrypted data for a non-display purpose
    c) This layer communicates with encryption and data storage layers through request handling
3) **The Encryption Layer**
    a) This layer will be the only point where data is encrypted and decrypted
    b) Any data that is to be stored or is coming from storage will pass through this layer
    c) This layer communicates with the App Logic layer and the Data Storage layer
4) **The Data Storage layer**
    a) This layer is our secure database that stores all data
    b) Data will include passwords, logins, and other sensitive data
    c) This layer will communicate with the App Logic layer and the Encryption Layer.

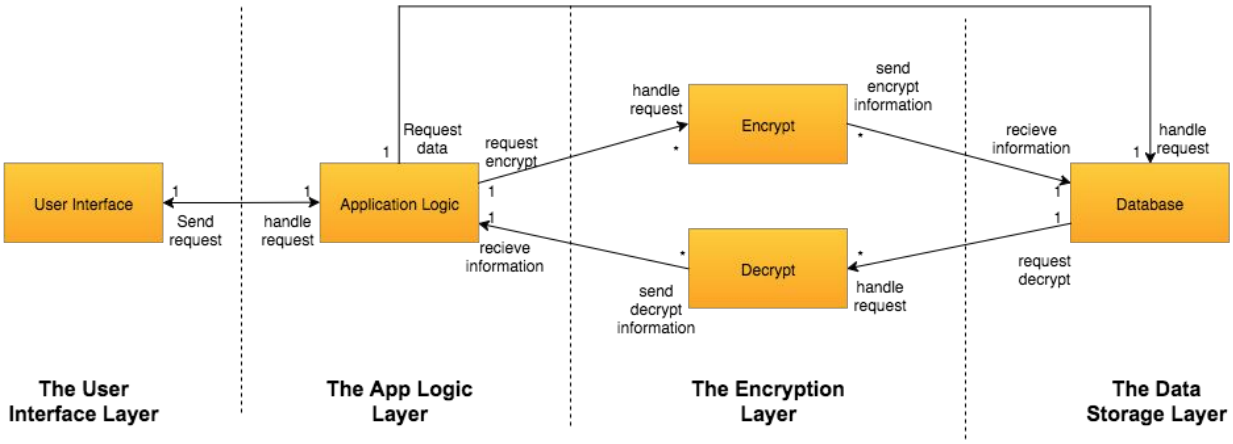


Figure 2.1 – High level overview

DESIGN ISSUES

Issue 1: What language should we write the system in?

Option 1: C / C++

Option 2: Python

Option 3: Java

Decision: We choose Option 3 because Java offers the greatest cross-platform flexibility, especially for generating a GUI, and provides opportunities to create modular, reusable code. Java also has large existing libraries and APIs directly applicable to our system. Lastly, as a team, we have the most experience using Java. C/C++ would be far too time restrictive to accomplish our system goals within our time frame.

Issue 2: What cryptography API should we use for encrypting and decrypting user data?

Option 1: RSA encryption

Option 2: Blowfish encryption

Option 3: Java Cryptography Architecture (JCA)

Decision: We choose Option 3 because the JCA is most easily implemented with Java because it's built for it. It's also the language that we are writing our system in. JCA provides the asymmetric cryptography features that work well with our design, along with a variety of encryption techniques and algorithms (rather than one single method).

Issue 3: How should we let users sort and search for their data?

Option 1: Search/Sort user data by date modified

Option 2: Search/Sort user data by data type

Option 3: Search/Sort user data by owner

Option 4: Search/Sort user data by entry name

Option 5: Search/Sort user data by login/password

Decision: We choose Options 1, 2, and 4 because these represent the most important information a user would need to quickly access or find desired data. Option 3 and 5 are not included because of privacy concerns; we don't want to allow logins and passwords to be visible. Option 5 is also not possible because passwords are encrypted.

Issue 4: What design pattern should we use to implement the Graphical User Interface?

Option 1: Navigation (drop-down) menus

Option 2: Layered menus

Option 3: Thumbnail / Icon viewer

Option 4: File Browser

Decision: We chose Option 4 as our GUI design pattern because it is the most familiar and intuitive layout to a majority of users. A file browser design also allows searching, sorting, and viewing potentially long lists of items to be easily displayed as compared to cluttered menu layers and thumbnails or unwieldy drop-downs. Option 4 will also let us display more necessary information on the screen at once.

Issue 5: Where should the user's data be stored?

Option 1: Store all user data locally

Option 2: Store data on a server

Option 3: Store data and public key on a server and store private key and decryption algorithm on the user's computer.

Option 4: Store data exclusively in external drives

Decision: We chose Option 1 as our data storage plan. This is the best option because it eliminates the vulnerabilities of a network connection and it ensures that the user will have complete control over their data in terms of where the data is stored and the security of an external server (Options 2 and 3). Option 4 is a possibility, but requires the a user always have the external drive in tandem with their computer. Furthermore, Option 1 allows for the possibility of storing backups on external drives.

Issue 6: How should the data be stored in the system?

Option 1: Storing data in an array

Option 2: Storing data in an SQL Database

Option 3: Storing data in a tree structure

Option 4: Storing data in a hash table

Decision: We chose Option 2 for our database design. Option 2 makes the most sense for us because SQL is a very powerful database language and has an API for Java, making it simple to use and implement. Using an array is impractical for both efficiency and storage size reasons. Using trees or a hash table could potentially be very fast, but building and maintaining the proper structure for them would be too great an undertaking in terms of time and complexity.

Issue 7: How should the user be allowed to share data to non-local users?

Option 1: Sharing via email

Option 2: Sharing via SMS (text messaging)

Option 3: Sharing via server

Decision: We decided on Option 1, sharing by email. Sending emails is much simpler to implement than setting up SMS and sending texts. Also, SMS would require storing a personal phone number for all users, whereas all users already use a trusted email as their vault login. A folder with permissions would only permit users on the same computer to share information, however we liked this idea as a separate form of local sharing and decided to implement it. Option 3 obviously requires the construction of an entire server/client setup for merely a single purpose. Additionally, we already decided we were not going to implement a server for our system.

Issue 8: How should users share data with local users?

Option 1: Sharing via folder

Option 2: Sharing via permissions

Option 3: Sharing via an in app user inbox for sharing of data entries

We decided on Option 2. Option 1 doesn't make much sense because sharing via folder increases the amount of data stored in the database. Option 3 doesn't make a great deal of sense if the users are already in the application because they would have to swap screens between viewing and decrypting the password. Sharing via permissions allows for a user to be notified that they now have access to new data and the user can then access the data accordingly.

Issue 9: How should a user login to the data vault?

Option 1: Enter a vault password to gain access to the application, and then use an account login (email) to access a user's specific account and data.

Option 2: At application launch, a user enters their vault account login (email, password) to gain access to their account and data.

Decision: We decided on Option 2 because Option 1 needlessly requires an extra password for the application, which, while more secure, is redundant and conflicts with our goal to allow users to only need one password to access their vault information. Also, the option of having two-factor authentication will give the ability to add extra security to the login process.

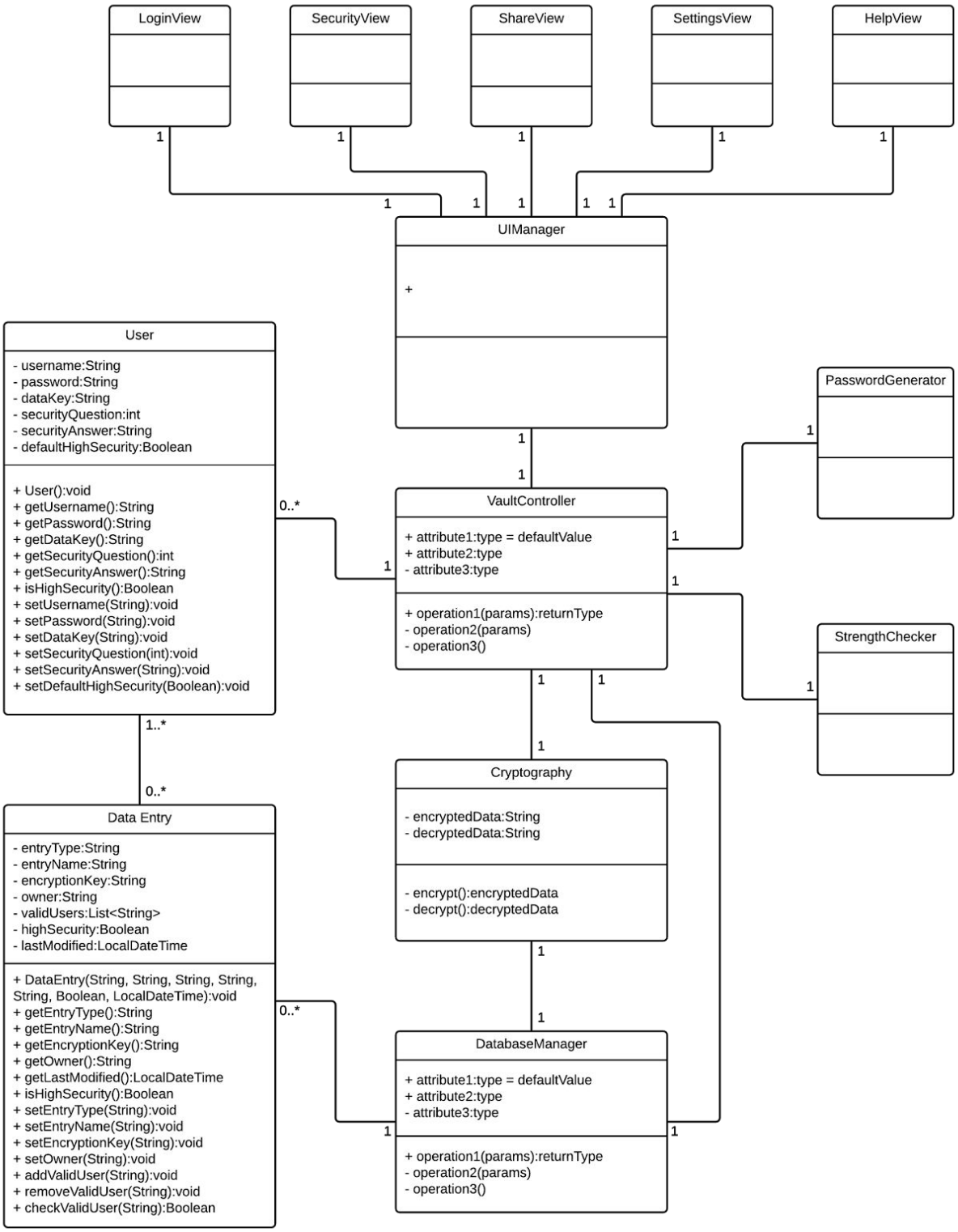
Issue 10 How many users should be allowed? How many data entries should be allowed per user?

> **Option 1:** Limited number of users (10>) and limited number of data storage entries (50>).
>
> **Option 2:** Limited number of users (50>) and limited number of data storage entries per user (100>).
>
> **Option 3**: No limit to number of users, but number of data entries per user is capped at 50.
>
> **Option 4:** Number of users capped at 20, but unlimited number of data entries per user (up to disk space size, user will be warned after a configurable file size is exceeded, for example, user will be warned after file size exceeds 5GB.)
>
> **Option 5:** No caps on number of users, or data entries per user, user will be warned after a certain (user configurable) file size is exceeded.

Decision: We decided on Option 5. This option makes the most sense because the data entries are small and limiting them arbitrarily seems to defeat the purpose of our system considering that users could potentially have a very high number of data entries. We will notify the user to ensure they do not fill their hard drive by accident, but in the end, it's the user's hard drive to do what they please with.

DESIGN DETAILS

a. Class Design of User-Data System



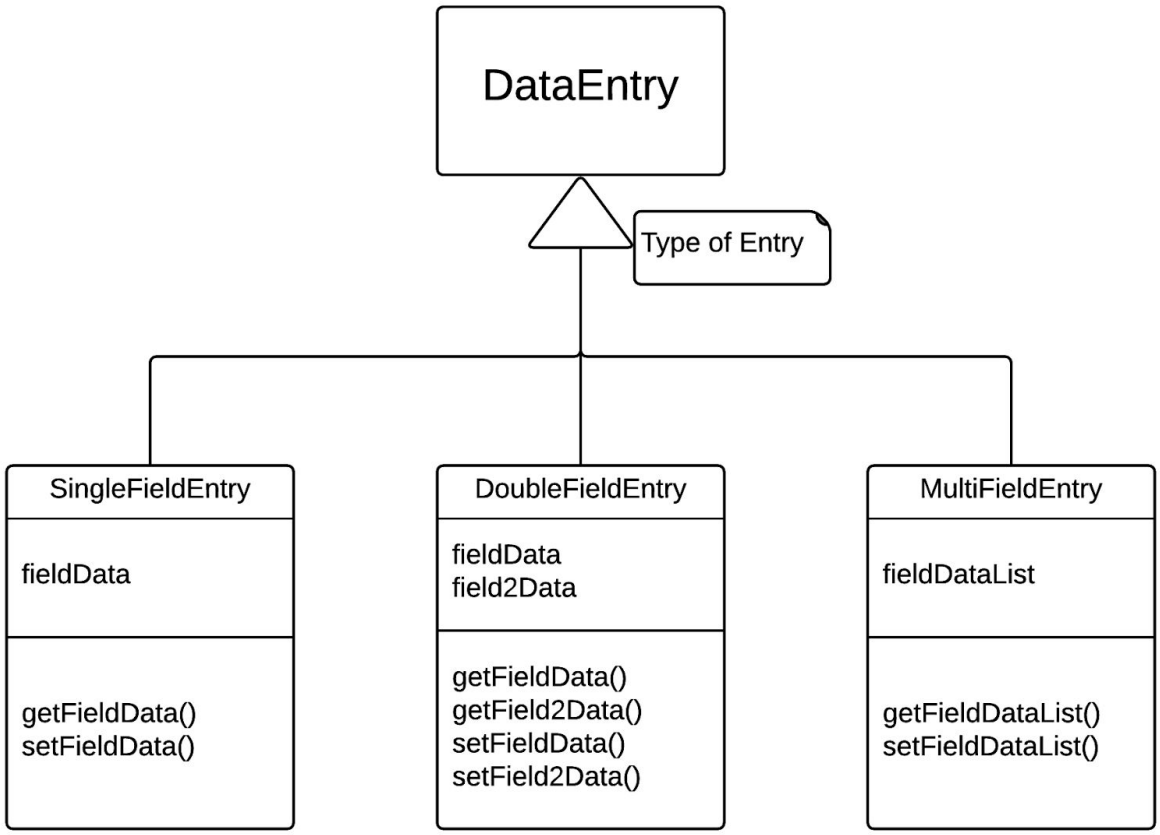


Figure 4.2 – Abstract DataEntry Class Diagram

b. Description for Class Design

User Class:

- Entity that summarizes the system's users.
- Stores user accounts information: username (login email), password, and login security question/answer.
- Contains getter and setter methods for all fields in the class
- Contains the encryption key used for decrypting a user's information.

DataEntry Class:

- The DataEntry class will be an abstract class to all of our data entry type subclasses, such as singleFieldEntry, doubleFieldEntry, and multiFieldEntry.
- This abstract class represents a basic data entry the user has entered into the system. The fields of the data entry abstract class are the bare minimum information fields that an entry can have.
- Subclasses will add fields for specific data and handlers for that data.

- Contains getter and setter methods for all fields.

VaultController Class:

- This class is the heart of the back end
- This class links the database, cryptography, and UI classes
- This class handles requests from the UI and makes requests of the database
- This class passes data to the cryptography class for encryption, and receives data from the cryptography class after it's been decrypted
- Handles the updating of data entries and user information

PasswordGenerator Class:

- This class generates a secure password based on user-specified parameters, including number of characters, special characters, character case, character redundancy, and punctuation.

StrengthChecker Class

- This class checks the strength of a password using regular expressions.
- Assigns a security score to the password based on the result of the strength checks.

Cryptography Class:

- This class is responsible for encrypting and decrypting sensitive data. Separate methods will perform encryption and decryption.
- Data being passed into the database will be run through the encryption method and data being passed out of the database will be passed through the decryption method.
- Data being shared will not be passed through the decryption process and will be retrieved in its encrypted state.

UI Manager Class:

- This class handles all requests and data from the other UI classes
- This class will pass requests for data and data to the Controller class
- This class will create the other UI classes when requested.

MainView Class:

- This class creates and displays the file browser interface
- This class will pass entered data to the UI manager class

LoginView Class:

- Displays the login screen to the user
- Allows access to the help screen and passes user entered data to

the UI manager class

ShareView Class:

- The user will be able to share data through this class
- This class will display the screen that allows the user to share data
- This class will pass requests and data to the UIManager class

SettingsView Class:

- This class will display the user's settings and allow for changes to the settings
- This class will pass requests and data to the UIManager class

HelpView Class:

- This class will display a preset help screen to the user, giving advice on how to use the application as well as security tips

SecurityView Class:

- This class will display a section that the user can use to generate a password
- This class will display a section that the user can use to check the security of a password

DatabaseManager Class:

- This class handles all the requests for data from the Controller class
- This class passes all data out through the cryptography class for decryption
- Any data entering this class will have passed through encryption in the cryptography class

c. System Interaction Diagrams

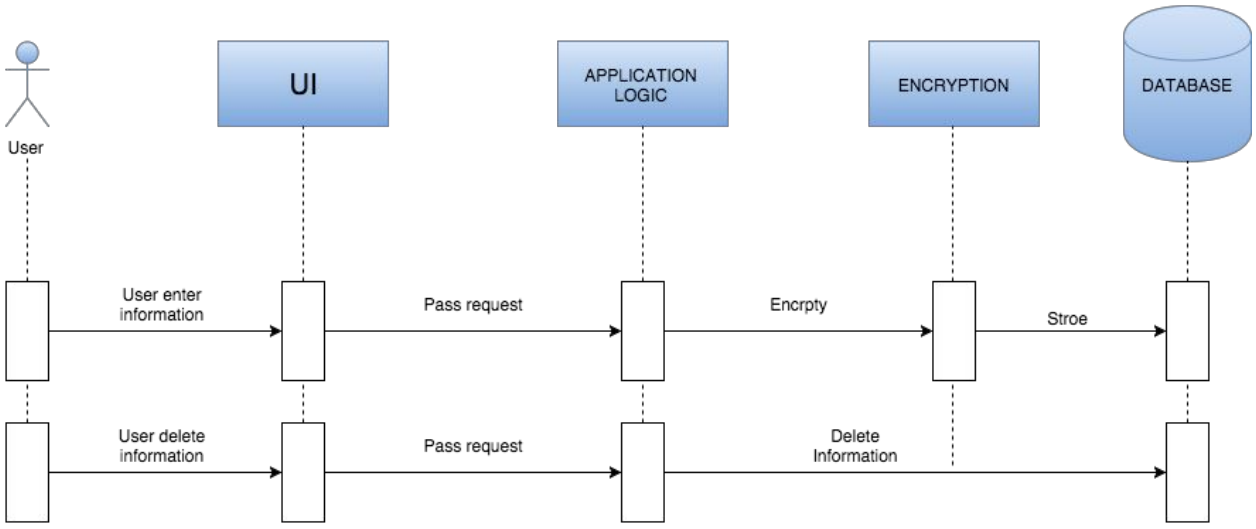


Figure 4.3 – Sequence of events when user adds or deletes information.

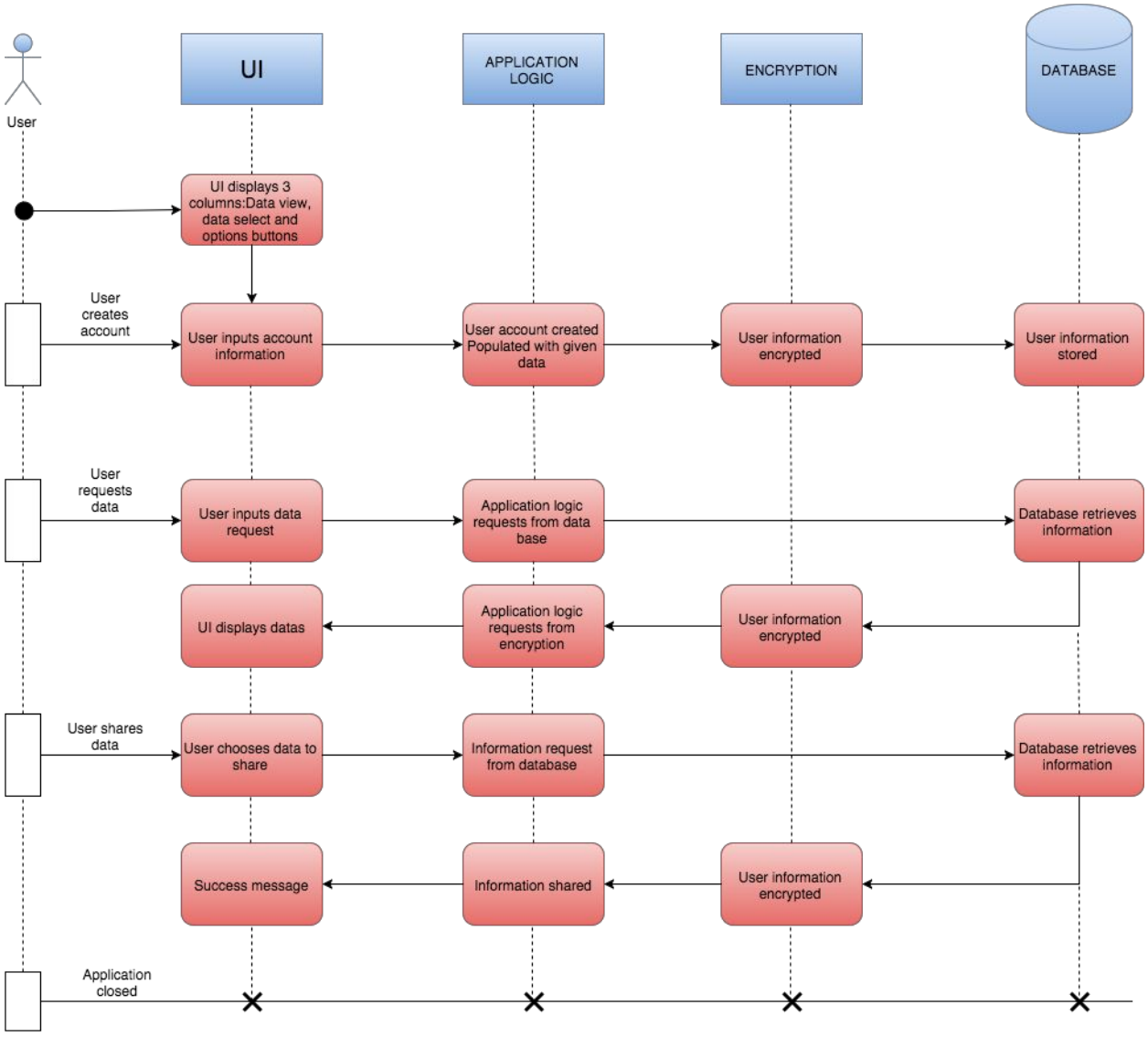


Figure 4.4 – Sequence of events when user creates an account, requests data, and shares data.

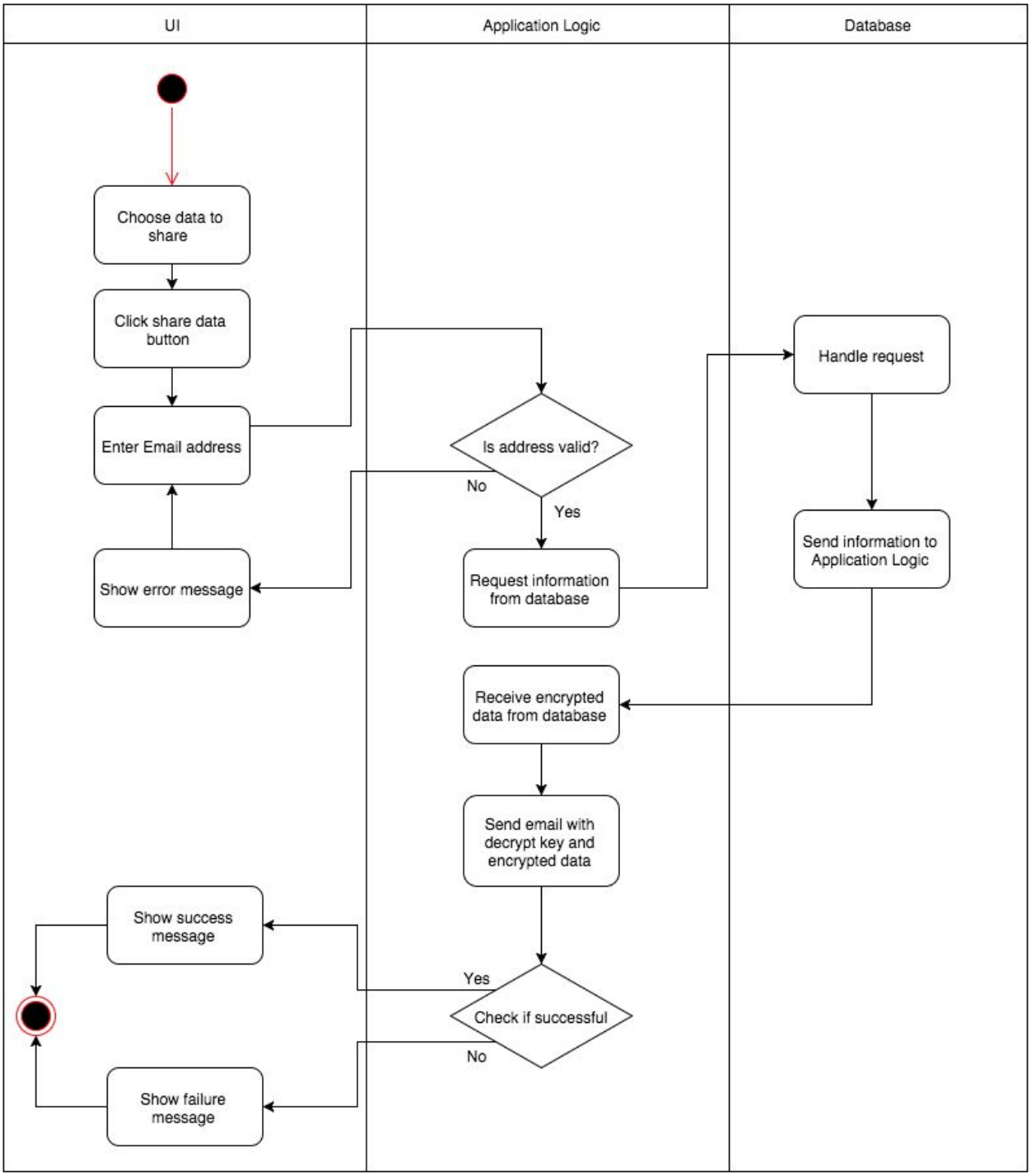


Figure 4.5 – Activity Diagram of non-local data entry sharing

d. User Interface Mockup

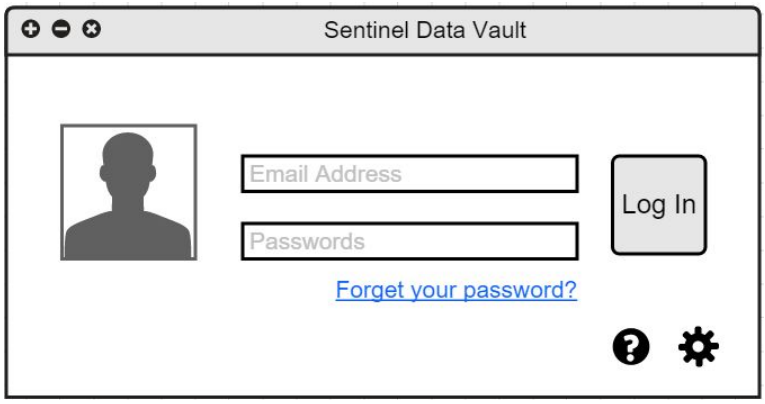


Figure 4.6 – Login view

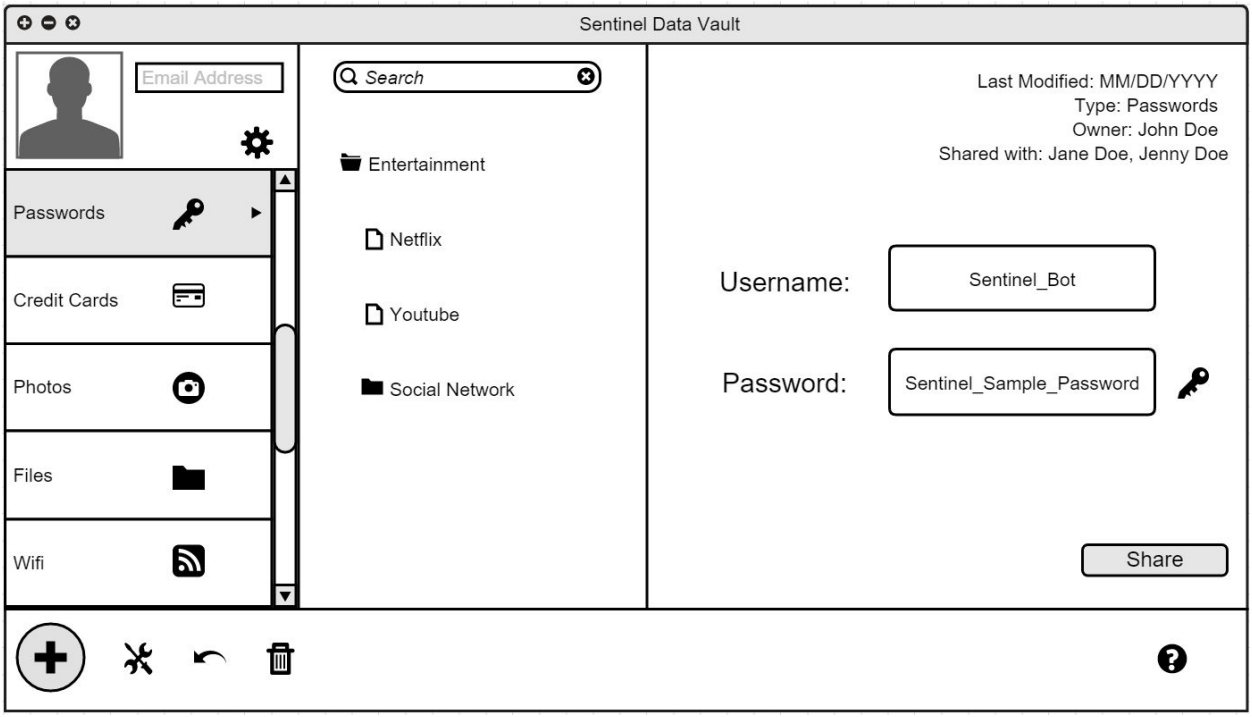


Figure 4.7 – Main view