ICSI 499 Capstone Project Report

Health Journaling App

Team Members:
Ethan Prescott, Jack Arevalo, Seoyeon Choi, Stephen Alvarez

College of Engineering and Applied Sciences
University at Albany, SUNY

Project Sponsor : Dr. Sherry Sahebi

05-03-2024

Acknowledgements

We are deeply grateful to Dr. Sherry Sahebi, who generously dedicates her time to conduct insightful meetings, share her expertise, and provide invaluable guidance throughout the development of our health journal app. Her thoughtful comments and constructive feedback were instrumental in achieving the success of our project. We would also like to thank Professor Pradeep Atrey for his excellent management of the Capstone course and skillfully organizing the course to facilitate our learning and growth. His consistent feedback on our milestone progress has been incredibly helpful in our academic journey.

Abstract

The health journal app offers two main features: The goal is to allow users to monitor health symptoms, including illnesses, previous X-rays, lab work, and doctor appointments, and facilitate data sharing for research purposes. Compatible with both iOS and Android devices, the app allows users to actively manage their health by maintaining comprehensive records. With user consent, health data is securely stored in cloud-based databases, allowing researchers to analyze and identify patterns within large data sets for diagnostic purposes. Alternatively, if users choose not to share their data, it is stored locally and securely in an individual device database. The app provides users with a convenient way to monitor and manage their health while contributing to valuable research efforts. Whether data is stored locally or in the cloud, users have control over their health information, ensuring privacy and security.

Contents

1 Problem Analysis

- Researchers and customers face two different challenges that can be solved simultaneously. Researchers need a platform to collect voluntary information from customers for data evaluation, but there are few centralized locations where customers can manage their self-reported health records. This presents an important opportunity as it provides an application form that allows customers to volunteer their medical background while protecting their privacy. The data collected can be used to train computer models that identify patterns within volunteer information.
- The key challenge is to build a secure and efficient platform to collect voluntary customer data while ensuring quality. Our solutions must ensure the security of the data collected and protect the privacy of our customers, while ensuring that the quality of the data meets the standards required for effective evaluation. Achieving this balance is critical to successfully deploying these platforms with regard to addressing both privacy and data reliability concerns.
- Furthermore, in the United States, high healthcare costs and long wait times for doctor visits create serious problems. Many diseases can be treated effectively if diagnosed early, but the cost and inconvenience of hospital visits often prevent individuals from proactively tracking their health. While there are numerous apps available for tracking various health metrics, they typically focus on specific aspects of health. What's lacking is a comprehensive, user-friendly app that integrates doctor appointments, medication reminders, and symptom tracking into one platform. While multiple apps can address these individual needs, there is a clear gap in a single solution that addresses all aspects of personal health care.

2 Proposed System/Application/Study

2.1 Overview

The health journaling app is designed for iOS and Android platforms and offers comprehensive features such as symptom tracking, appointment management, medical history tracking, and journaling features. Our primary focus is to provide a user-centric experience and prioritize the needs and preferences of our customers over those of their healthcare providers. In this section, we take a closer look at the implementation details of each functional requirement and briefly describe how we leveraged database management systems and applied software engineering principles acquired during our studies at the University at Albany.

2.2 Project Requirements

2.2.1 User Classes

- **Individuals**

  The primary user class for this app is individuals who utilize it to record symptoms, illnesses, lab work, and previous medical tests such as x-rays. Although the app could potentially be expanded to include additional user classes, such as administrators and doctors, the current scope of the project, as defined by the sponsoring professor, prioritizes the ease with which users can journal their health and track medications and doctor appointments for each respective date.

2.2.2 Function Requirements

2.2.2.1: Symptom Tracker

- Users will be able to log any symptoms with or without an associated illness in addition to the date of the symptoms. The user should have the option to store the data both locally

or using cloud storage.

Solution: Integrate the symptom tracker into the journaling feature

- Users access the journaling option to create an entry, facilitating symptom logging and easy access to stored data near the current date. By consolidating features within the journaling function, users are presented with a streamlined experience, eliminating overwhelming options. Additionally, the implementation includes a query feature that suggests previously logged symptoms as users type, enhancing ease of use and efficiency in symptom journaling.

2.2.2.2: Appointment Management

- Users should be able to keep track of the date, starting time, end time, and location of upcoming doctors appointments. Users are able to click a specific date to see the track of their date. The app should also notify the user when the appointment is about to start. Additionally, the users should be able to store their data both locally or using cloud storage.

Solution : Users can input data regarding their medical appointments, enabling them to track their appointment results by clicking on each date displayed on the calendar screen. Users have the option to review past appointments by simply clicking on previous dates on the calendar, streamlining the process of managing their health schedule. This consolidated approach allows users to multitask and efficiently track their overall health without the need for multiple apps.

The ultimate goal of this functional requirement is to prioritize user-friendliness while highlighting the benefits of using this app, ultimately encouraging users to actively engage with its features. Moreover, it specifically emphasizes the journaling aspect of our app, aiming to provide users with a personalized journal for tracking their health rather than merely serving as a repository for their medical information.

2.2.2.3: Medical Tracking

- Users can input the time, date, frequency, and dosage (in mg) of prescribed medication. The app offers useful options to add medication details such as lists of drug names, dosages, dosing schedules (e.g., morning, evening, during the day), and frequencies (with

the ability to select multiple days). All data is stored in the database as string types, with storage location (local or cloud) determined by user consent.

Solution 1 (Local or Cloud): Utilizing SQLite and Firebase, in order to prioritize user privacy by default, storing data locally unless the user opts to share it for research purposes. The user interface presents options for storing data locally via SQLite and Firebase. After inputting information, the UI prompt in the storage settings page has a toggle button that allows users to freely choose cloud storage (via Firebase) or local storage (via SQLite), granting complete control over data privacy. Currently our app focuses on working with local databases. This approach ensures flexibility, accessibility, remote data access, and synchronization, maintaining data consistency across platforms.

2.2.2.4: Overall Journaling Functionality

- Users will be given guided Journal Prompts to easily organize, maintain and document entries into their health journals, along with this the user will be able to add basic information such as list of symptoms , list of illness, and list of tests and labworks.

Solution (Local):
- The app uses SQLite and React Native to store the information obtained from these prompts in a local database, ensuring accessibility even without an internet connection. This local storage not only facilitates faster data access but also enhances privacy. However, leveraging a cloud database provides significant benefits such as data synchronization, backup, and increased flexibility. To optimize app functionality, we designed a solution that combines local and cloud databases and prioritizes user privacy. By leveraging the strengths and mitigating the drawbacks of both approaches, you can make data storage more reliable while maintaining a secure network. By default, the application stores data locally in a highly secure database, with users also given the option to choose cloud storage if desired. When cloud storage is enabled, the local database is exported to Firebase as a JSON file and priority is given to cloud based Firebase functions rather than SQL functions.

2.2.3 Non-Functional Requirements

2.2.3.1 Performance

1. Accuracy: Assure user entered health data is accurately recorded and displayed to

users such as calculation, data analysis.

2. Efficiency: Application should interactively report data, show results based on users
   interaction with application

2.2.3.2 Security:

1. Data Encryption : User's data should be protected from unauthorized access. All data is encrypted.
2. User Authentication: Provide password protection or two-factor authentication to prevent unauthorized users from accessing data. (End-to-End Encryption)
3. Secure Data Storage : If a user wants to share data, securely store data on server or cloud platform.
4. Data storage location control: Allow each user to choose where they want to store their health data, either locally on their mobile device or in the cloud. Provide a clear explanation of this security functionality so that each user can make their own decision.

2.2.3.3 Reliability:

1. Error Handling: Provide robust error-handling mechanism to prevent application crashes.
2. Backup and Recovery: prepare backup data and recovery plan to prepare for system failure or unexpected data loss cases.
3. Monitoring and Maintenance: regularly monitor applications to see whether there is but or security vulnerability.

2.2.3.4 Availability:

1. Minimize Downtime: In order to ensure smooth performance while there is high traffic, provide users request evenly toward the server to avoid overloading.
2. Notification System: tell the user in advance if the app needs to go down for

maintenance and update.

2.2.3.5 Scalability:

1. Scalability for Database : Choose appropriate database solution so that application is able to handle growing data volumes and users
2. Auto-Scaling : Automatic resource allocation based on real-time demand.

2.2.4 Operating Requirements

2.2.4.1 Operating Requirements: IOS and Android OS

The app is compatible with both iOS and Android operating systems. Users can test the app by installing the Expo Go app from the App Store or Google Play Store. Additionally, the program can be executed using Terminal or Visual Studio Code by installing the necessary node_modules with "npm install" and running the program with "npx expo start."

2.2.4.2 Technology Stack

The app was built using React Native, SQLite, and Firebase.

2.2.4.3 Other Software

Development was primarily carried out by a team of four members using Visual Studio Code and GitHub. The team followed a collaborative workflow where each member made pull requests from the master branch to work on their respective tasks. After completing their tasks, members requested pull requests, resolved conflicts, and tested the app on both Android and iOS platforms. Upon visual verification and error checking, successful changes were merged into the master branch, allowing all team members to work simultaneously on the project.

2.2.5 Design and Implementation Constraints

2.2.5.1 privacy and data security

The app must be strictly followed to keep individual users privacy issues and not sharing users health information if they did not consent to share it.

2.2.5.2 Accessibility

The app's design and visualization should be adjustable to both iOS and Android platforms, ensuring that CSS styling is adaptable to provide a seamless user experience across both operating systems.

2.2.5.3 Compliance

The app must follow the regulation or health sector industry standards such as following federal law HIPAA (Health Insurance Portability and Accountability Act) to protect individuals credential health information being used without patient’s agreements or knowledge.

2.2.5.4 User Experience Design

The app's design should prioritize usability, making it easy for users to navigate between screens and interact with various features and functionalities effortlessly.

2.3 Technical Design

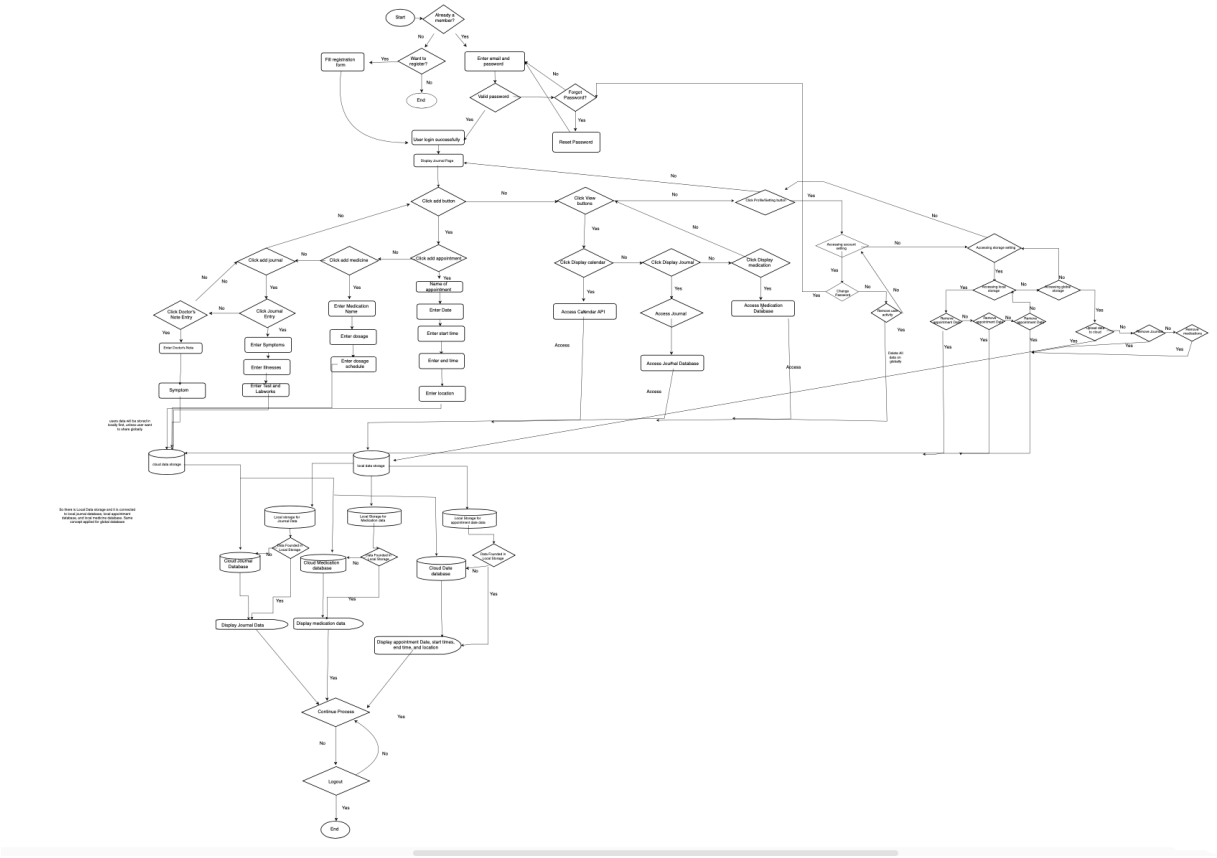

Figure : overall system flow diagram.

Based on an extensive system flow diagram, we describe the smooth progression of the system from the initial login page through the health journaling process, which includes storing health information and subsequent retrieval for user viewing.

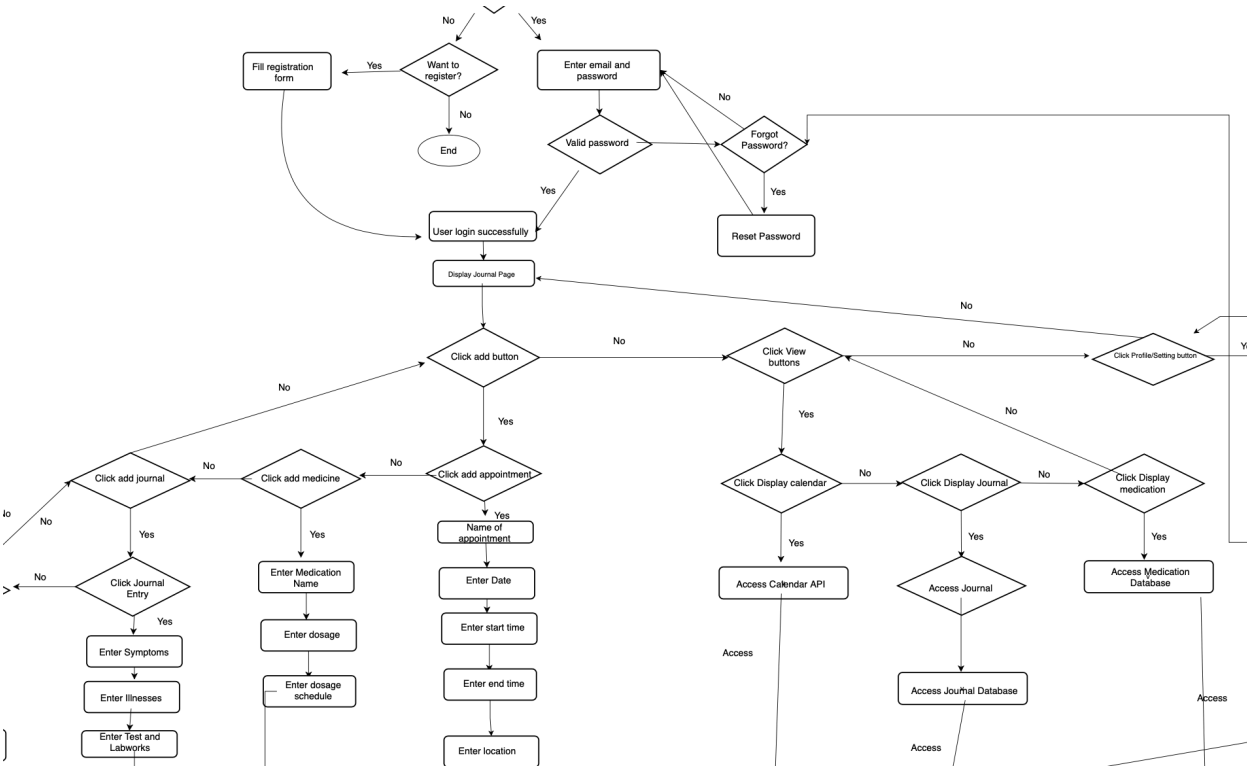

Figure 2: Login and Create account process.

This system flow chart focuses on the login process. When a user uses this app, they will first see a login page. If a user is registered with this app, the system will prompt the user to enter the user's email and password. If a user forgets their password, they can re-enter it or reset their old password to create a new password. If the user is not currently registered, we will ask the user to create an account and then log in to the application. In our case, logging in is important because users store their personal health information. Therefore, once the login is confirmed, the system allows the user to interact or view his or her health diary data and information.

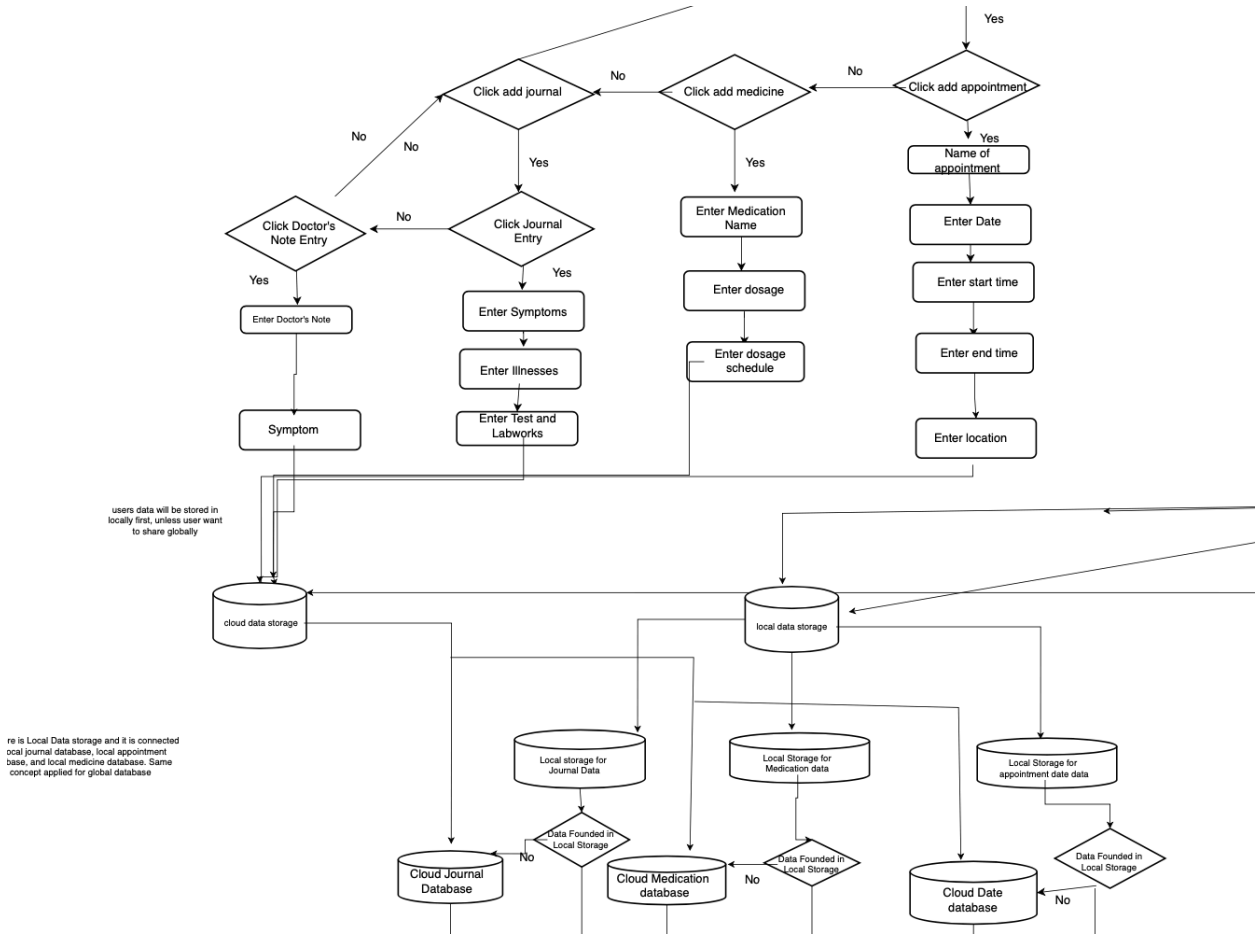

Figure 3 : Entering Data

This diagram shows how the user will interact with the system when the user tries to enter data and asks whether the user wants to store the data locally or globally. There are two main databases, a local database and a global database. Each of these databases branch out into smaller databases which store specific data. By default all user's data will be stored locally on their device and when users are allowed to share their data on cloud database, data will be uploaded on cloud database. The reason we created local databases and cloud databases is to not evade each user's privacy at the same time researchers can collect data from cloud databases and use those.

Second, we don't want one large global or local database. The reason we branch out is because having three separate databases for each data type is better for data organization and management. Additionally, three separate local and global databases allow you to allocate

resources based on the user's specific needs and provide more flexibility in scaling as your data grows. Separated databases also provide better security controls for each piece of data.

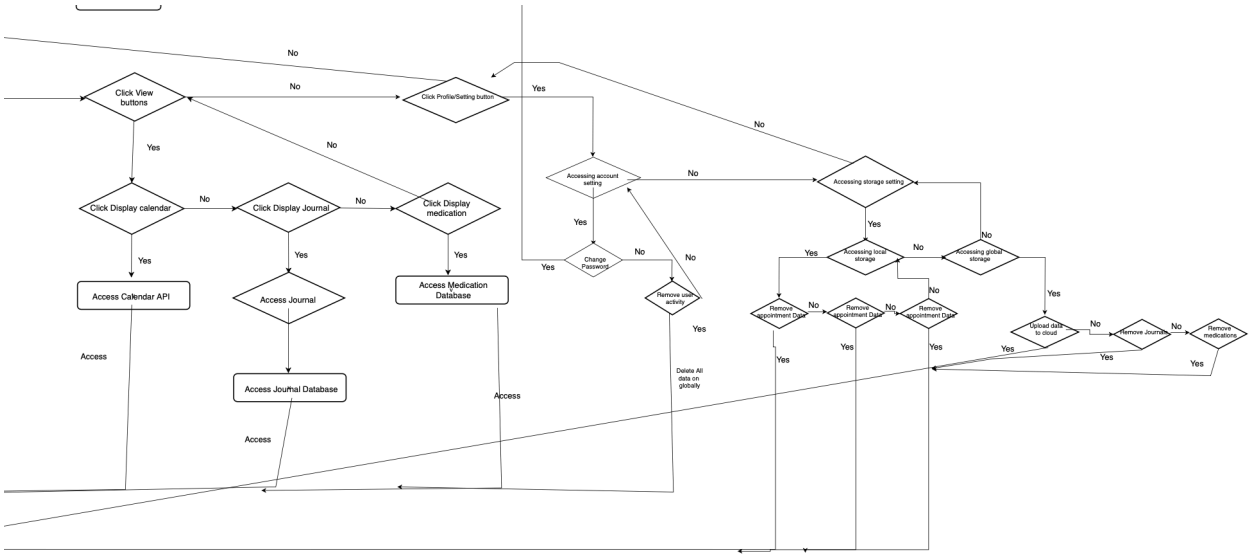

Figure 4: accessing data and setting implementation

This diagram shows how users interact with the system when they click buttons to display their stored data. When a user wants to view their stored data (journal, medication note, and appointment schedule), the system will access the cloud database first and if there is no match data, it will check local database storage. When the user clicks the setting button, users can have choices to remove their entry, change their storage setting, and account setting.

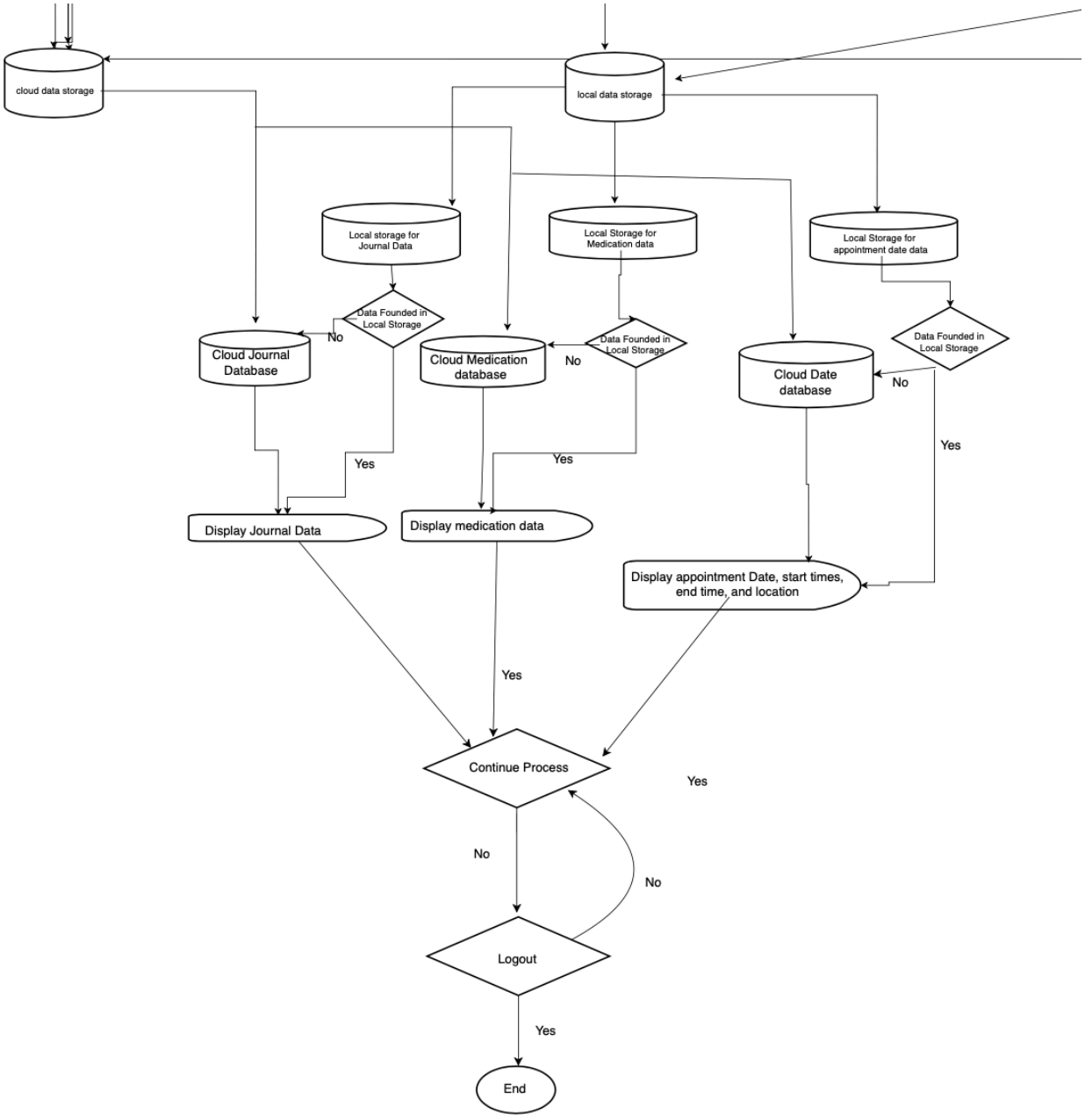

Figure 5: Local and Global database system flow.

This system flow shows when a user wants to view stored data. Entering each of the screens queries the database tables for associated information depending on if local or cloud storage is enabled. It then displays the requested data to the user.

2.4 System Implementation

Our team utilized GitHub as our primary version control software, leveraging features such as pull requests, resolving merge conflicts, and merging changes into the master branch to enhance overall functionality. This approach facilitated simultaneous work, minimized redundancy in implementing overlapping functionalities, and enabled effective milestone tracking, task planning, and progress monitoring for individual team members.

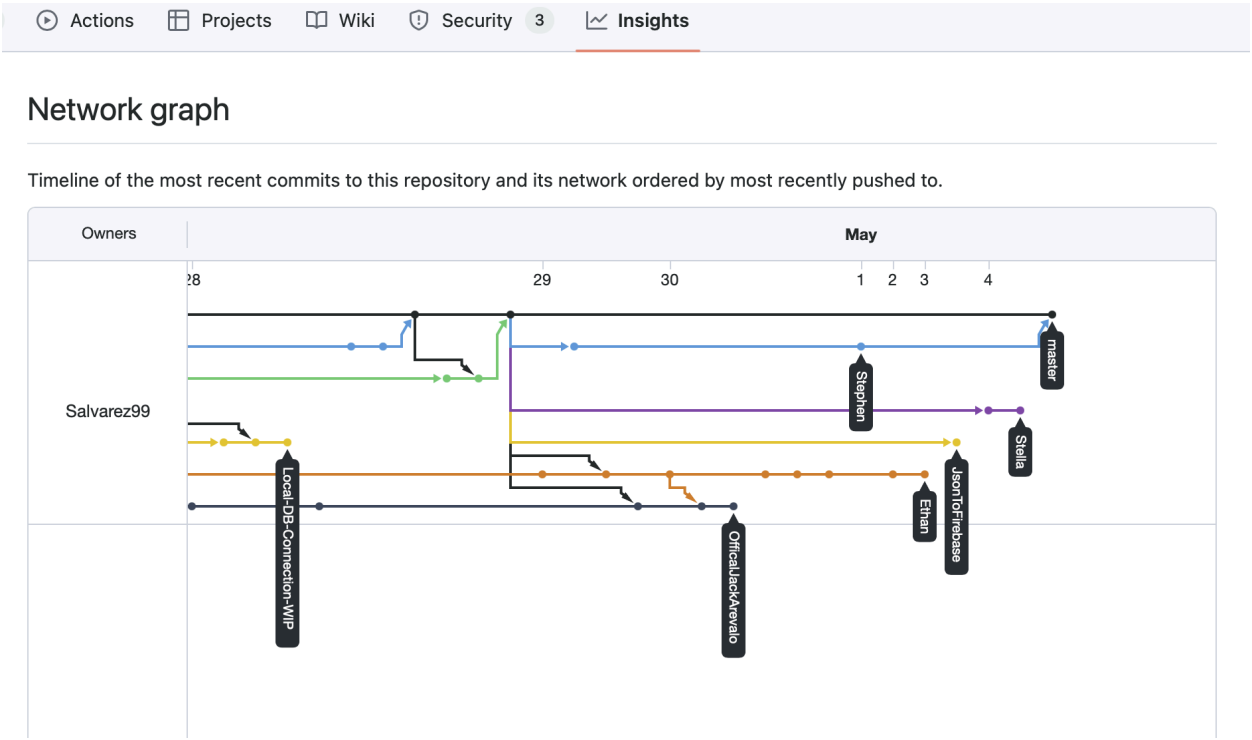

Figure 6 : Graph of Branches on GitHub Repository, 05/04/2024

For frontend programming, our primary tool was Visual Studio Code, seamlessly linked to GitHub. This integration allowed us to pull from the master branch and implement code efficiently. Utilizing Visual Studio Code simplified error diagnosis and debugging through the integrated terminal.

To test styling and visualization across both iOS and Android platforms, we used Android Studio

emulator(SDK) and physical Android devices. For testing the iOS portion, we utilized physical iOS devices to ensure accurate visualization and functionality testing.

For the backend side …

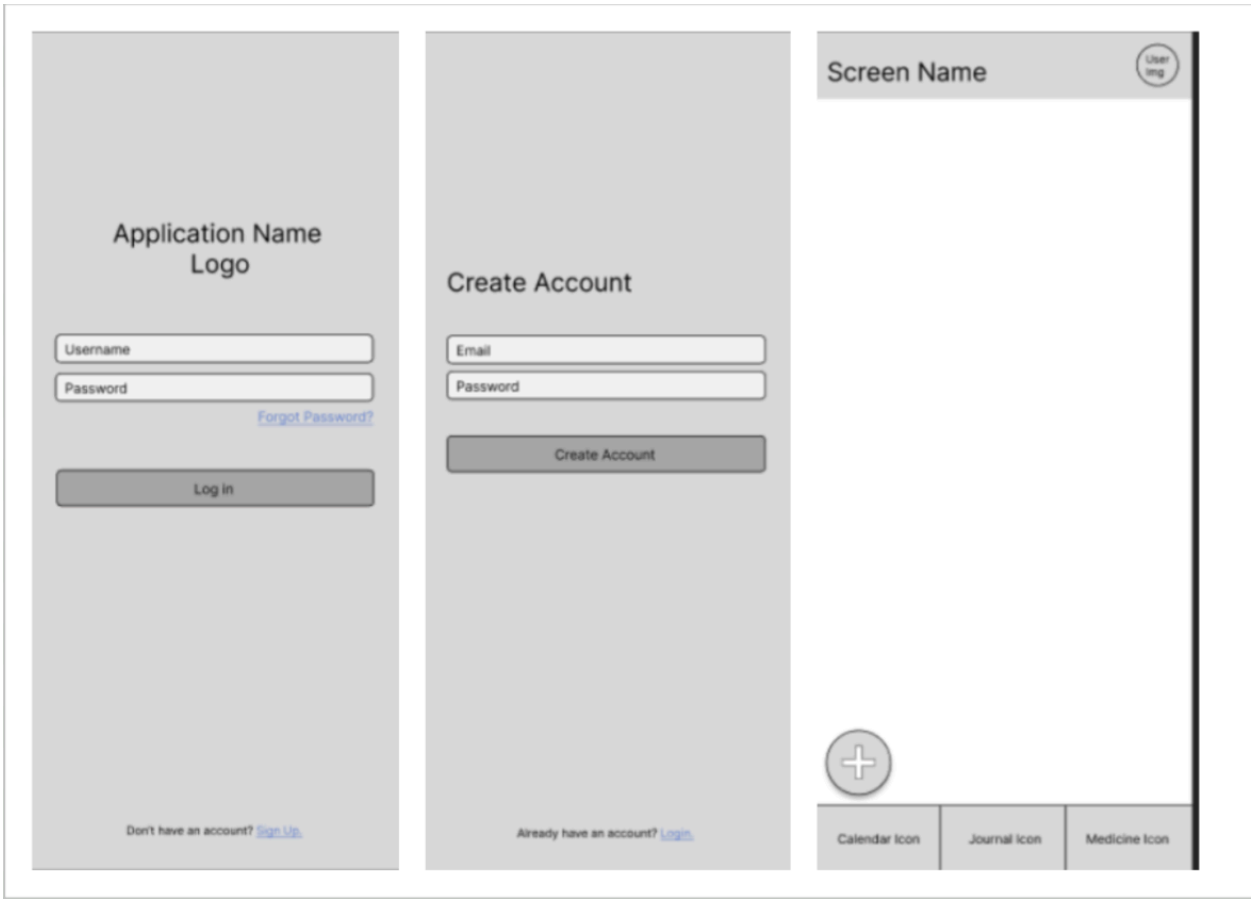

Figure 7 : GUI login, Sign In, and main page after login.

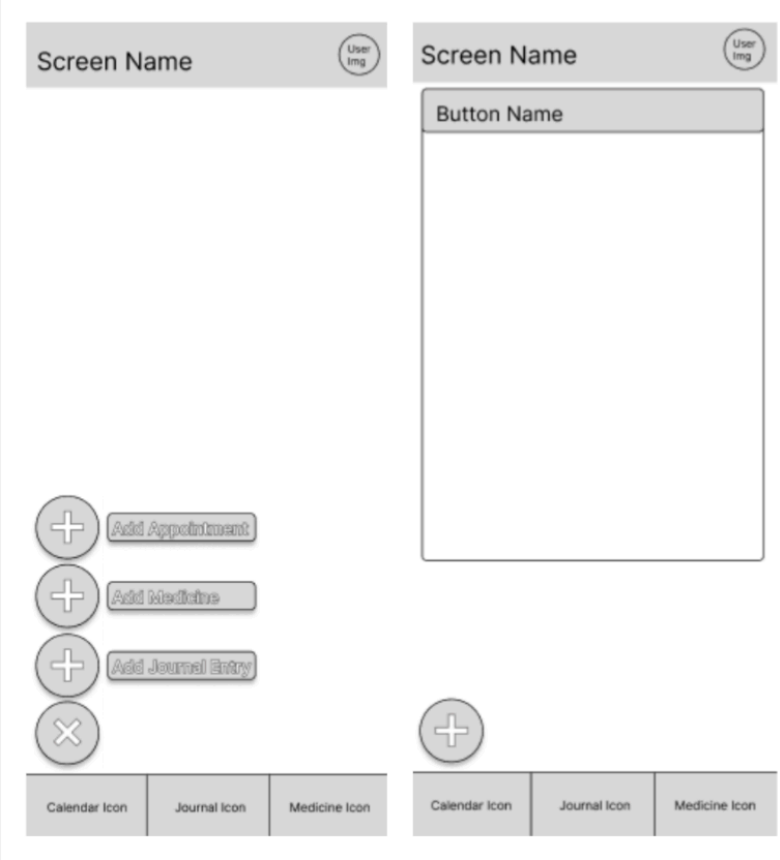

Figure 8 : Add entry forms buttons.

To enhance system implementation effectiveness and visualization, we developed a graphical user interface (GUI) to demonstrate visualization and monitor the implementation of all necessary functionalities.

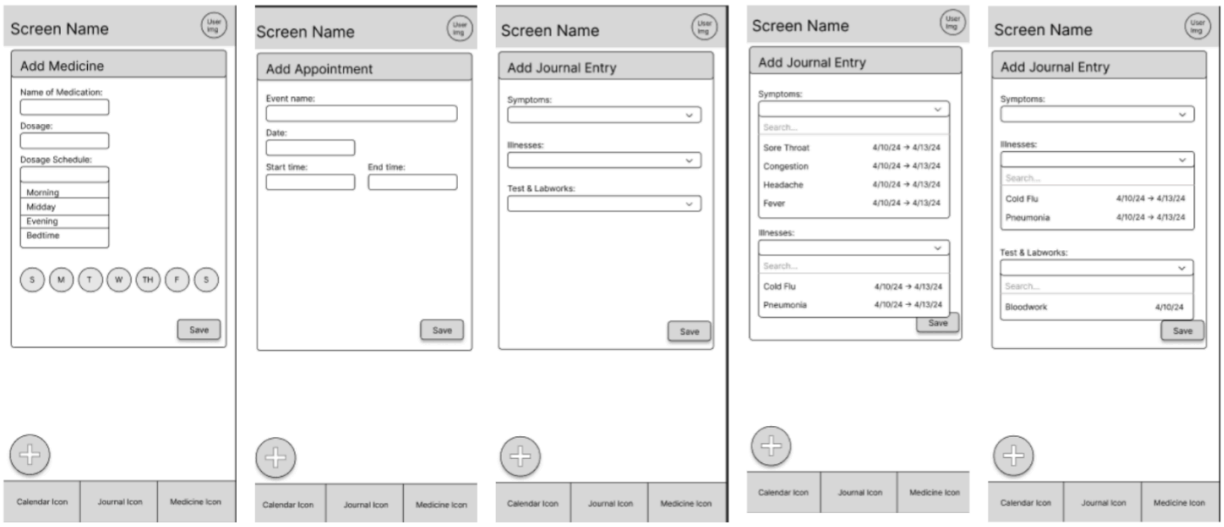

Figure 9 : Input forms.

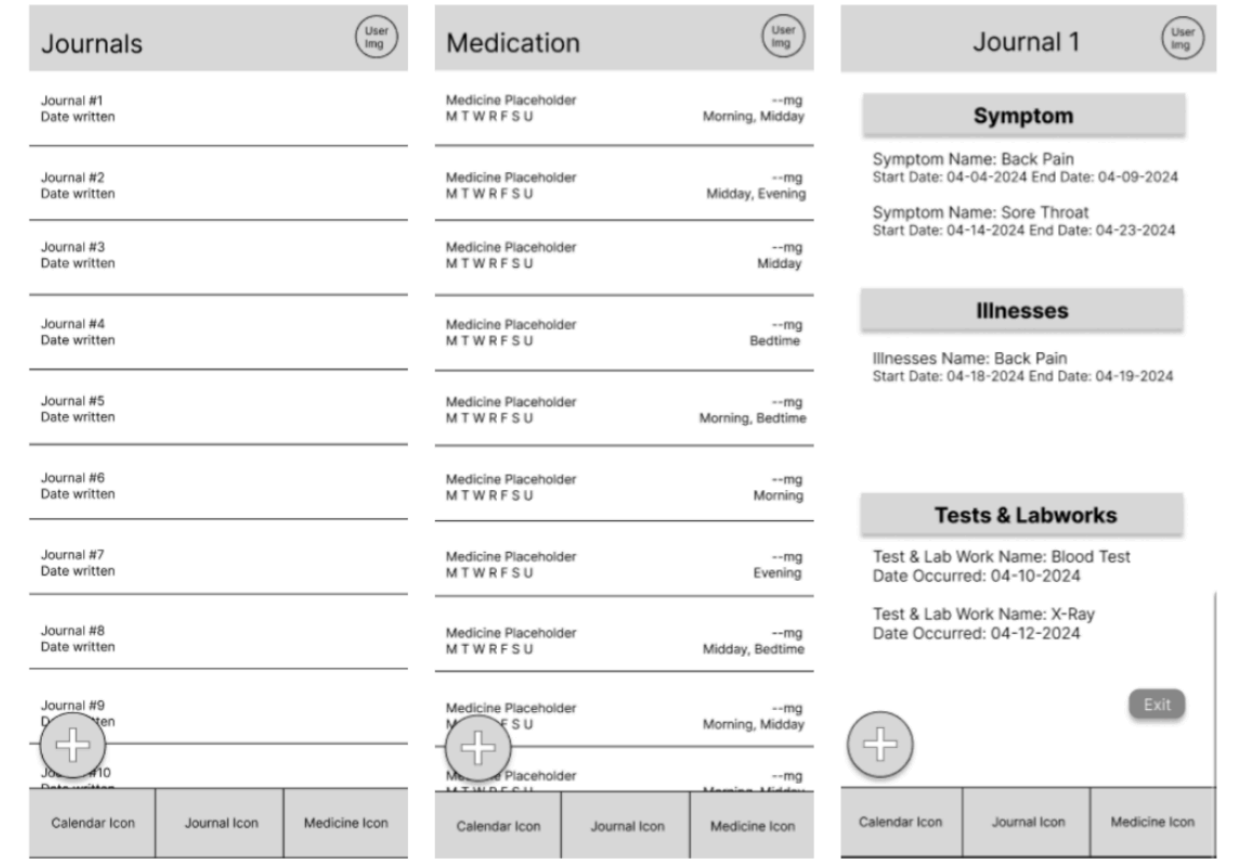

Figure 10 : displaying stored journal entries and appointment data.

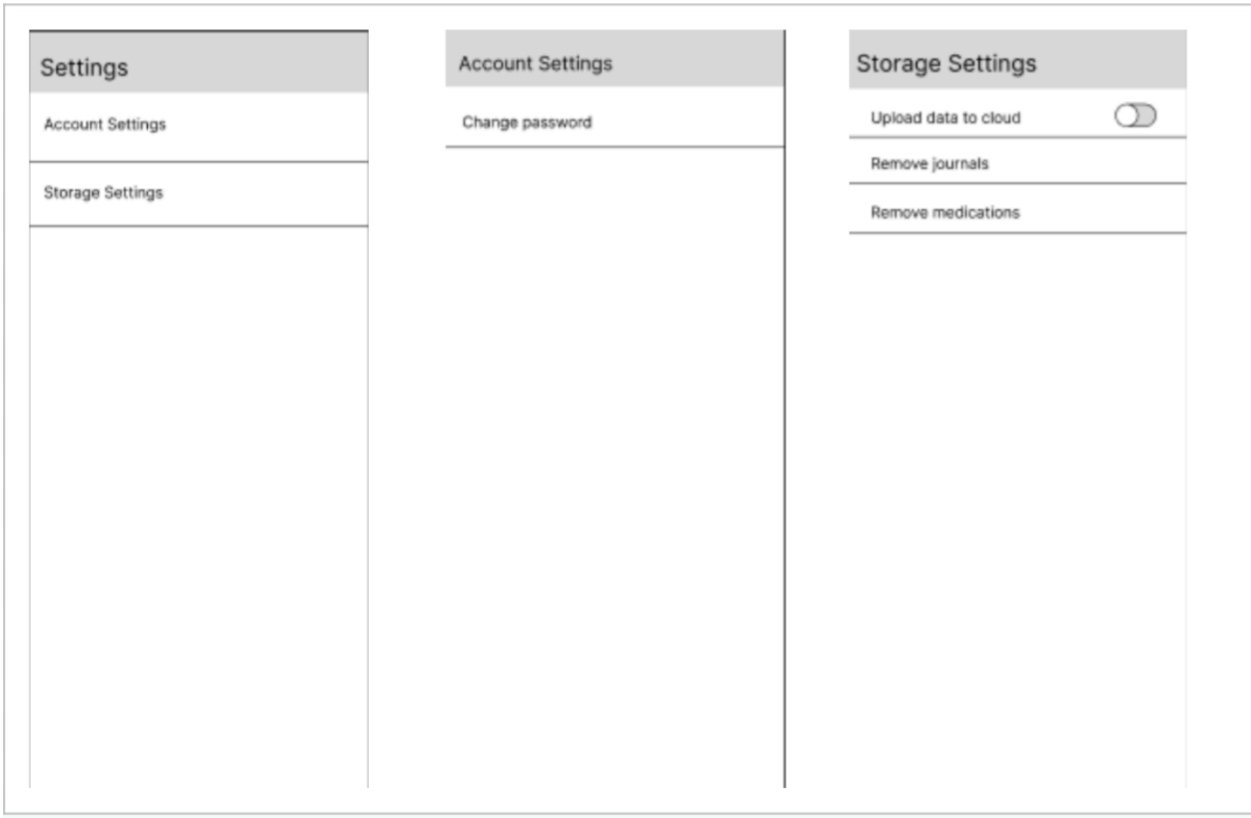

Figure 11 : setting page.

2.5 Use of Computer Science Theory and Software Development Fundamentals

In this section, we will delve into the theoretical concepts and knowledge from computer science that we applied during the implementation and development of the health journaling app.

2.5.1 Use of Computer Science Theories

2.5.1.1 Database Management System :

Database uses transaction management to group multiple SQL statements into a single

unit of work, in which either all succeed or all fail. This ensures data integrity even in the presence of concurrent operations or system failures. For example, each table creation and data insertion operation in the provided code is encapsulated within a transaction, ensuring that either all operations succeed or none are applied. This allows us to implement the opening of the database when the user first logs into the application, and insertion of data into the database can be handled within another file in our project. All this is possible to how we separated the transactions within separate functions.

2.5.1.2 Distributed System

We store data locally and upload data to the cloud based on users consent and this is considered as one form of distributed system where the app operates as a distributed system where data storage is distributed across as individual users local devices and remote cloud servers. This distributed system architecture offers several advantages, including enhanced accessibility, suitability for various usage scenarios, and fault tolerance.

For details, storing data locally enables fast access, enhancing performance by quickly displaying stored data on the screen. Additionally, the cloud server is not burdened with every single user request, such as storing and fetching data, which can reduce complexity. Furthermore, our distributed system is scalable, capable of accommodating increased demand as the user base expands. With more users joining the platform, the system can seamlessly scale its resources to meet growing requirements, ensuring optimal performance and user experience for all users. In summary, our adoption of a distributed system architecture empowers our health journaling app with robustness, efficiency, and scalability, ultimately enhancing user satisfaction and data management capabilities.

2.5.1.3 Foreign Key Constraints

Foreign key constraints are used to maintain referential integrity between related tables. For example, the tables userSymptom, userIllness, and userTest all share data regarding journals but a single journal could share information from each of these tables. In order to link them all as a single entry the table journalEntry was created and it’s primary key id

was used as a foreign key in the 3 user tables. With that foreign key we are able to recognize any table that shares the same foreign key JID as a part of a single journal.

2.5.2 Use of Software Development Fundamentals

2.5.2.1 Modular Design

Our app breaks down necessary functionality into two main categories: displaying data and capturing data. Within these categories, we further break down the design by creating reusable components such as setting buttons, nav bars, and toggle buttons. Additionally, we divide functionalities into smaller modules to handle user authentication, medication management, appointment tracking, symptom and illness tracking, etc. This modular design provides significant benefits for testing, debugging errors efficiently without affecting the entire system, as well as for maintenance.

2.5.2.2 System Data Flow Diagram

During the planning phase of our project, we made use of Data Flow Diagrams to visualize and understand the flow of data through our system. These diagrams allowed us to see what data is being gathered, and how we planned to process this data. Afterwards this allowed us to ensure that the data would be stored in the proper locations. By using DFDs, we could clearly see the interactions between different components of the system, which helped in identifying any discrepancies in data handling.

2.5.2.3 Entity Relation Diagram

An Entity-Relationship Diagram was utilized in our project to streamline data storage by visualizing and organizing relationships between data entities. This allowed for the management of user-specific information in unique tables, while maintaining overall

tables for symptoms, illnesses, tests, and medications.

3 Experimental Design and Testing

3.1 Experimental Setup

In this section, we'll cover how the app can be effectively tested on physical devices or emulators to validate its functionality. For frontend testing, we predominantly utilized Visual Studio Code and leveraged the functionalities provided by the React Native Expo library to implement components and program the frontend.

- Prerequisites
    - For developers using Windows
        - https://reactnative.dev/docs/environment-setup?guide=native&os=windows
    - For iOS developers here is some documentation for initial setup for react native
        - https://reactnative.dev/docs/environment-setup?guide=native&os=macos
- Steps to run the application
    - Create a Folder and clone the Repository into said folder, then navigate to the root directory, in this case “HealthJournalApp” in the terminal. Once in the root directory run the command “npm install” this will install all dependencies needed for the application to run.
    - Afterwards run the command “npm start” this will start the build process and prompt you with a menu in the terminal to open the app in a few different ways.
- For testing on iOS devices,
    - Download the Expo Go app from the Apple App Store.
    - Once the Metro bundler is running in the terminal, the Expo app on your physical iOS device will automatically connect to it.
        - If you need to reconnect, simply type "R" in the terminal or shake your

physical iOS device and click the reload button.

- To test on Android devices
    - Run “npm start” and scan the QR code with the companion ExpoGo App on your android device.
    - Alternatively, you can press “a” or run “npm run android” if you have an android emulator installed.
- Troubleshooting
    - If the app encounters dependency issues despite having the node_modules installed within the project directory, resolving the problem can often be achieved by deleting the node_modules folder and reinstalling it, or clearing the cache.
- For backend side testing
    - The backend was tested mostly by utilizing front end elements to log the outcome of different queries to the console. Testing and debugging for the local database ended up being a challenge as the .db file generated by our database is inaccessible in the android emulator so we were not able to use any database management software to view the data, in addition to the limited nature of Javascript debugging.
- Firebase connectivity is set up by creating a firebase project on Firebase’s website and generating a key for your app. After you install the dependencies for firebase with react-native all that is left is to set up the firebaseConfig.js file and include the API keys in said config file along with setup for any other feature you’d like such as Firebase-Authentication.

  While our health journaling app can be tested using Metro on the terminal and Expo app, it's essential to note that user login accounts (emails and passwords) and logged health information, including symptoms, illnesses, x-rays, lab work, doctor appointments, and medications, will be securely stored in a backend database. Our app accommodates users' ability to input multiple symptoms, illnesses, x-rays, and lab work on the same date, without limitations on storing multiple data entries. Additionally, the app can fetch data from the backend and display it on the screen accordingly. This data management and visualization are facilitated through Firebase and SQLite.

3.2 Dataset

Since our app relies heavily on user interaction, our dataset may vary depending on how users engage with our application. For initial testing purposes, the data is pre-populated to evaluate how the app helps users enter health information and arrange and display that data on the screen.

Users can also test the app by entering their health symptoms. For example, User A might enter symptoms such as fever starting at 9 AM and ending at 11 AM and cough starting at 2 PM and ending at 2:30 PM on May 4, 2024. afternoon. This data is then stored in the local database as a list of strings, which can be searched and sorted for display purposes.

An important feature of the app is its ability to retain health data entered by the user even after the app is closed and reopened. This ensures that the data you enter is saved and remains accessible for future reference.

3.3 Results and Analysis

Our application is designed to fulfill its primary purpose of collecting users' health data while adhering to federal law HIPAA by enabling individuals to voluntarily share their health data as they see fit. Concurrently, we emphasize delivering functionalities that provide significant benefits to users, allowing them to efficiently track their health and manage medication and doctor's appointments within a single app.

For the frontend implementation and analysis:

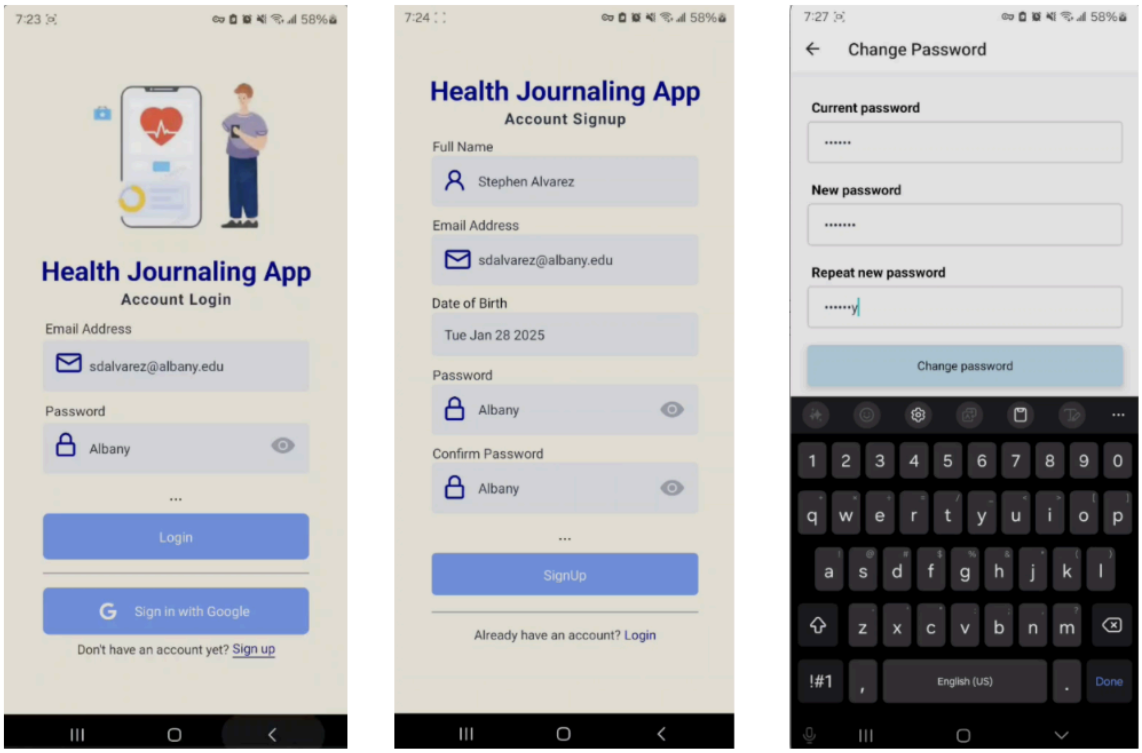

Figure : Login, sign, account setting page in app.

Login, Sign-up, and Account Setting Pages: We've developed a login and sign-up page enabling users to create accounts while ensuring validation through email and password verification. Additionally, in the settings page, users can conveniently change their passwords, updating them in the database accordingly.

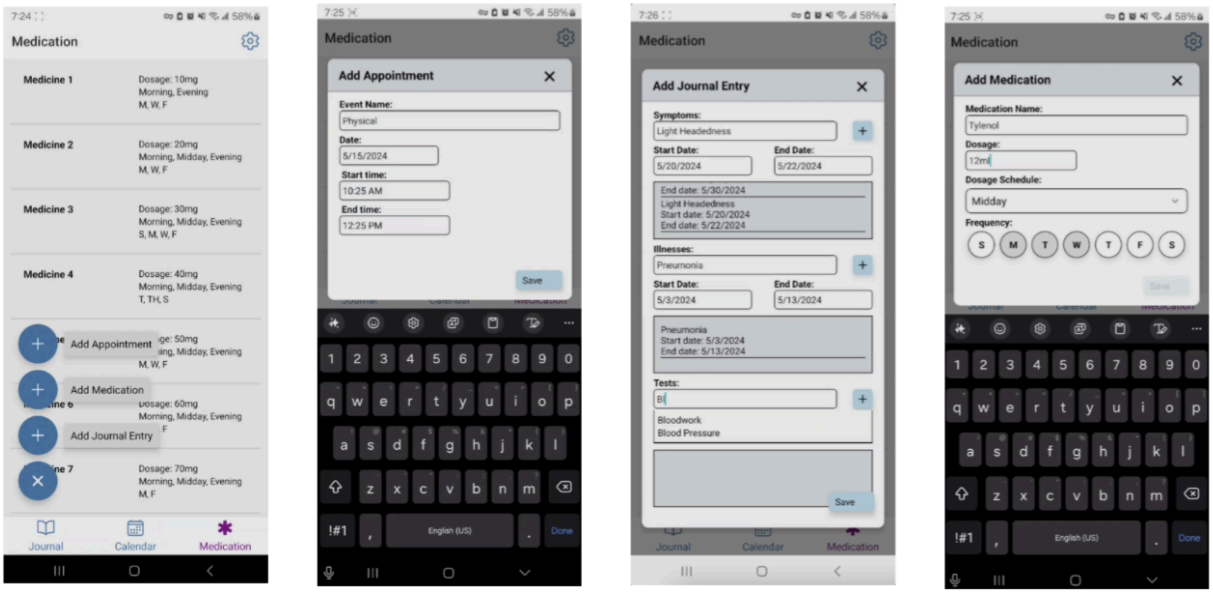

Figure : Input form screens.

Input Forms for Medication, Health Journaling, and Doctor's Appointments: We've created three distinct input forms facilitating the entry of medication details, health journaling information, and doctor's appointment data. To enhance the journaling experience, we've incorporated a query feature that suggests previously entered symptoms or illnesses, streamlining the data entry process.

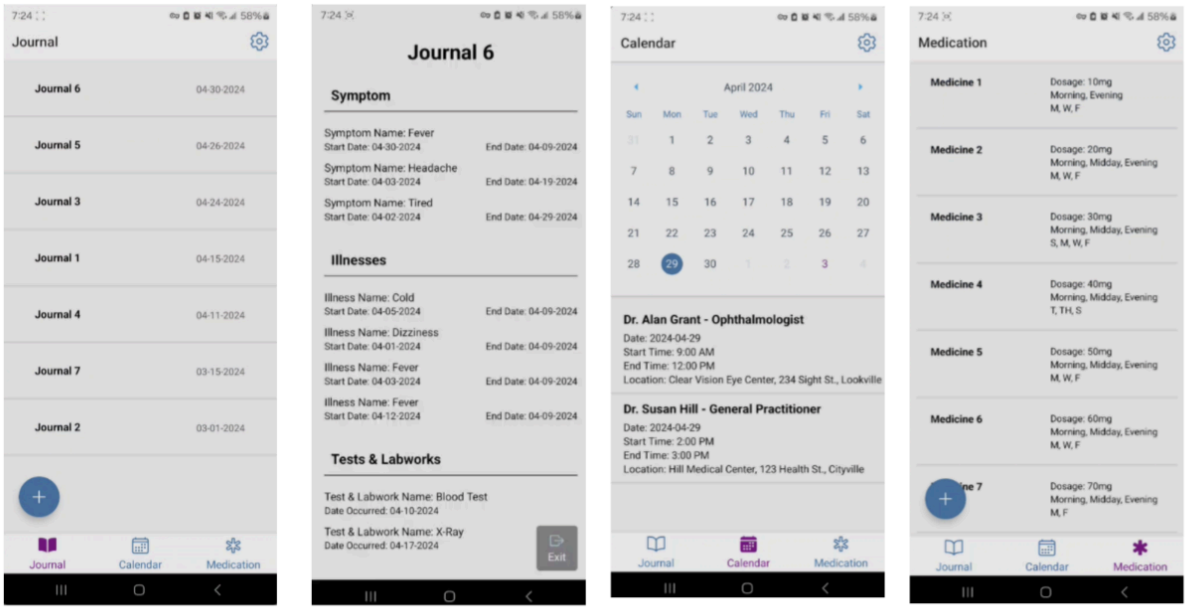

Figure displaying screens.

Display Screens for Reviewing Previous Records: We've created three dedicated display screens (viewing journal entries, medication list, and doctors appointment on calendar) enabling users to review their past records, supporting the app's core objective of helping users track their symptoms for use during hospital visits.

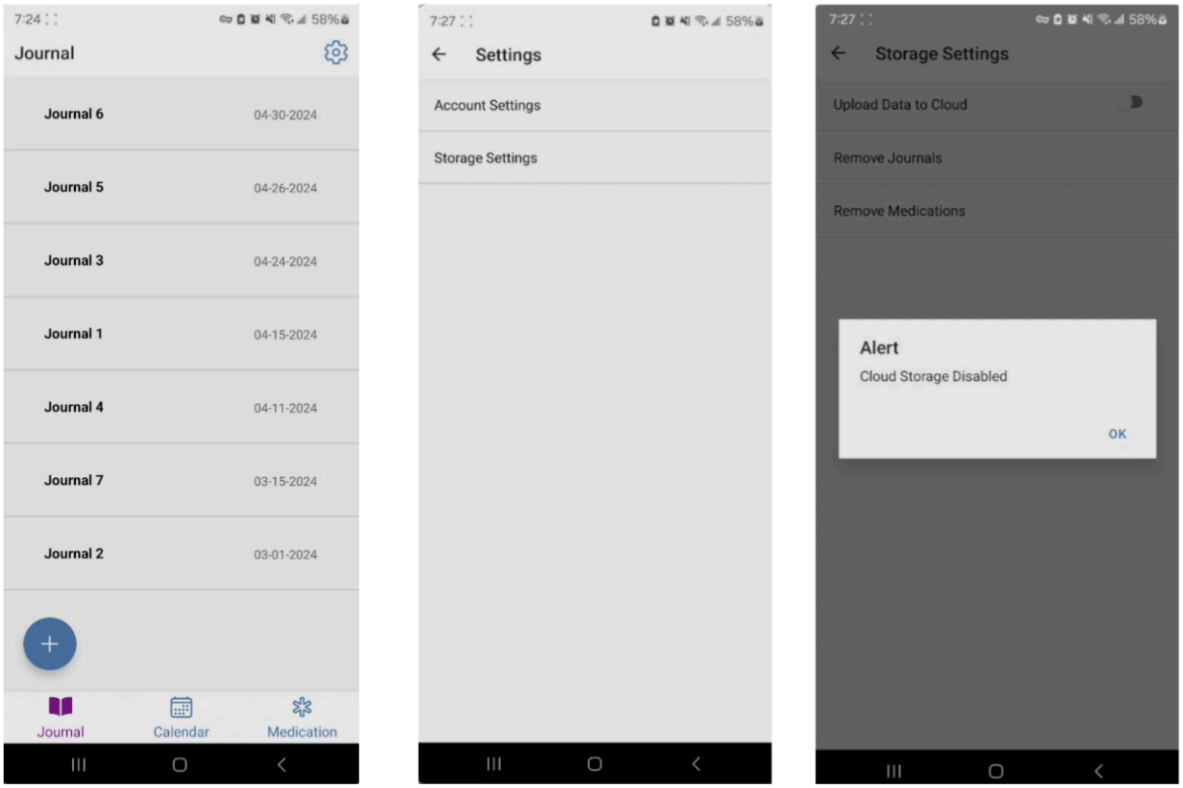

Figure : Setting Page

Setting Toggle Button: A setting toggle button is provided, allowing users to conveniently modify their passwords, choose to share their health data with the app for storage in the cloud, and remove journal or medication entry data.

Our app effectively delivers the functionalities needed with robust and efficient visualizations optimized for both iOS and Android platforms. Users can log multiple symptoms, illnesses, X-rays, doctor's appointments, and medication details. Our application processes these inputs, sorts, organizes, and stores them in the appropriate local database. Subsequently, it retrieves the relevant data, formats it as necessary, and displays it to the user.

Data security is a top priority; therefore, information is securely stored in our local databases, which also helps in the proper verification of registered users. Moreover, in compliance with HIPAA regulations, users maintain full control over their health data. They can update or delete entries, ensuring they have the utmost authority over their personal information. This functionality not only enhances user autonomy but also ensures our application meets stringent legal standards.

4 Legal and Ethical Practices

4.1 Legal Considerations

To comply with legal standards, health data applications must adhere to privacy laws like HIPAA or GDPR, ensuring robust security to safeguard personal health information (PHI) and obtain user consent transparently. Clear communication about data usage, risks, and user control is vital, alongside adherence to medical regulations and industry standards.

4.2 Ethical Considerations

The application securely stores user information and only allows access to individual accounts. Transparent data handling practices inform users about how their data is used. Access controls help users manage their data sharing preferences, building trust in the platform's commitment to privacy and security.

5. Effort Sharing

To ensure smooth collaboration and balanced workload distribution, our team conducted bi-weekly online meetings via Discord. We divided the team into two pairs: two members focused on frontend development and the other two on backend tasks, allowing simultaneous progress.

For frontend development, we fostered idea-sharing and frequent discussions to mitigate overlapping implementation issues. We utilized Visual Studio Code and GitHub to streamline our workflow, enabling effective feedback sharing and iterative improvements to enhance app functionalities.

Work was split up in the back end by having one of us focus on Firebase and one of us focus on local storage. As the project got further into development and we began to focus more on the local storage over the cloud storage, the work was split up by one working on Database management and the other working on the connection between the frontend and backend. Similarly to the front end we utilized github to streamline our workflow.

Individual tasks were allocated within each frontend and backend duo, while ensuring alignment with overall project objectives. Despite this division, our collective aim remained focused on delivering a functional app.

Table 1: Effort Sharing

Team Size	Joint efforts	Ethan Prescott	Jack Arevalo	Seoyeon Choi	Stephen Alvarze
4	J(≈ 20%) Preparing Showcase Preparing each milestone (including reports, video, ppt) Using Github. Fix merge conflict. Participating in a meeting.	I(≈ 20%) Set up and implemented Firebase Authentication Created and designed the LocalDatabase Manager Created pre-populated tables used for search elements Connected UserTables to JournalTitle to display properly Actively created new SQLite	I(≈ 20%) Set up the sqlite local database Researched how to query sql functions in react native Implemented the connection for inserting data into the medicine tables Implemented the connection for displaying the data from the tables	I(≈ 20%) Data flow diagram and system flow diagram Fetching and displaying screens : Journal entry screen to display stored journal Journal title screen to display details of stored illness, symptom, and x-ray and lab work data Login and Signup pages	I(≈ 35%) GUI mockup Mock ERD for backend database Input forms : journal entry, medication entry, appointment entry to take inputs from users and pass to the backend database. Overall application navigation Setting page with storage setting implementation.

		functions and tables to suit the requirements in the front end and make connectivity between backend and front end more convenient.		Setting page, Setting button, and account setting page. CSS styling	Storing Inputted Journal Entries to local DB : Made use of SQL queries to store gathered data into proper tables Helping team members by providing instruction and sample documentation to aid in implementation. Set up and managed work flow on github ensure minimal merge conflicts.

J : description of tasks jointly performed

I : description of tasks individually performed

6. Conclusion and Future Work

In conclusion, the health journaling app provides various health record app functions in one, while maintaining ease of use, eliminating unnecessary functions, providing users with a sense of freedom, and preventing unauthorized access to individual users' health information. It satisfies two main purposes: providing secure login for users. And those who want to share more of their health data with apps so that the data can be stored in the cloud to provide more researchable resources in the future without violating users' rights

to keep their personal and health information private. The app also moves to different screens and scales using platform size (ios or android) to provide great visualization.

For future scope, this app can integrate each user's data stored and provide PDF version of diagrams showing their health, including managers or doctors as different user classes, actuarial hospitality can be associated with this app, and patients can send their health records remotely. By reducing wait times and improving overall turnaround times, this app can help hospitals effectively check the health of individual patients remotely and provide healthcare services to more people. And to further enhance security and prevent leakage issues, we're implementing more features to quickly transition between storing data in local and cloud databases.

7. Bibliography

[1] Simon LundhSimon Lundh 2133 bronze badges, Nina ScholzNina Scholz 383k2626 gold badges356356 silver badges409409 bronze badges, and FalyFaly 13.4k22 gold badges2020 silver badges3838 bronze badges, “Sort by closest date to dates which have occurred and will occur,” Stack Overflow, https://stackoverflow.com/questions/47071623/sort-by-closest-date-to-dates-which-have-occured-and-will-occur (accessed May 4, 2024).

[2] “React native tutorial - 75 - passing data between screens,” YouTube, https://youtu.be/oBAOr1OswkQ?si=NQ_XdTnzKk3t8xGd (accessed May 4, 2024).

[3] D. Pavlutin, “How to use fetch with Async/await,” Dmitri Pavlutin Blog, https://dmitripavlutin.com/javascript-fetch-async-await/ (accessed May 4, 2024).

[4] “React native change password form template,” React native Change password form template, https://www.rnexamples.com/react-native-examples/bN/Change-password-form (accessed May 4, 2024).

[5] “Expo react native login system #1 | creating the pages (step by step),” YouTube,

https://www.youtube.com/watch?v=BQ-kHwLlhrg (accessed May 4, 2024).

[6] M. Priyanjalee, “Understanding the map() function in react.js,” Medium, https://medium.com/analytics-vidhya/understanding-the-map-function-in-react-js-1d211916fea7 (accessed May 4, 2024).

[7] “Lists and keys,” React, https://legacy.reactjs.org/docs/lists-and-keys.html (accessed May 4, 2024).

[8] Squashlabs, “Accessing array length in this.state in reactjs,” Squash, https://www.squash.io/accessing-array-length-in-thisstate-in-reactjs/ (accessed May 4, 2024).

[9] jasonee96, “React useeffect cleanup function within if statement,” The freeCodeCamp Forum, https://forum.freecodecamp.org/t/react-useeffect-cleanup-function-within-if-statement/556965 (accessed May 4, 2024).

[10] “React-native-calendars,” npm, https://www.npmjs.com/package/react-native-calendars (accessed May 4, 2024).

[11] “Low-code calendar component,” The Draftbit Community, https://community.draftbit.com/c/code-snippets/low-code-calendar-component (accessed May 4, 2024).

[12] “Low-code calendar component,” The Draftbit Community, https://community.draftbit.com/c/code-snippets/low-code-calendar-component (accessed May 4, 2024).

[13] “Health Insurance Portability and accountability act of 1996 (HIPAA),” Centers for Disease Control and Prevention, https://www.cdc.gov/phlp/publications/topic/hipaa.html#:~:text=The%20Health%20Insurance%20Portability%20and,the%20patient’s%20consent%20or%20knowledge. (accessed May 4, 2024).

[14] Chatgpt, https://openai.com/chatgpt (accessed May 6, 2024).

[15] React-Native-Datetimepicker, “React-native-datetimepicker/Datetimepicker: React native date & time picker component for IOS, Android and windows,” GitHub, https://github.com/react-native-datetimepicker/datetimepicker (accessed May 6, 2024).

[16] hoaphantn7604, “Hoaphantn7604/react-native-element-dropdown: A react-native dropdown component easy to customize for both IOS and Android.,” GitHub, https://github.com/hoaphantn7604/react-native-element-dropdown (accessed May 6, 2024).