Serenova

CS307: Team 24 Design Document

Team Members
Cristina Corley, Kara Orander, Ishwarya Samavedhi
Ava Schrandt, Mary Voorhees, Caitlin Wilson

1. Purpose **3**

1.1 Functional Requirements 3

1.2 Non-Functional Requirements 7

2. Design Outline **9**

2.1 High Level Overview 9

2.2 Overview of the Client/Server/Database Architecture 11

2.3 Sequence of Events Overview 11

3. Design Issues **13**

3.1 Functional Issues 13

3.2 Nonfunctional Issues 16

4. Design Details **21**

4.1 Class Diagram 21

4.2 Classes and Models Descriptions 21

4.3 Sequence and Activity Diagrams 27

4.4 Navigation Flow Map 35

4.5 UI Mockup 36

4.6 Database Design 46

1. Purpose

In a world where people are consistently striving to improve their overall sleep experience, most sleep tracking apps fail to actively help their users find a path to a better sleep schedule. Existing iOS sleep tracking apps, such as Apple's Sleep, AutoSleep, and Shuteye, do track the user's sleep patterns and offer general advice, but fall short of providing a personalized plan for sleep improvements with a community forum feature for sharing experiences. Such apps also hide these features behind a paywall and have limited integration of health apps and devices that users may have.

Serenova aims to raise the standard for iOS sleep tracking apps. Our app will have more features than an average sleep tracker, including a dream journal and improvement challenges to help users fix their sleep schedule to get more rest. There will be elements involved that let users interact with each other and share their own experiences and encourage each other. Serenova will cover the shortcomings of previous sleep tracking apps by including free features that support the integration of various health apps and devices and fostering a community between all users, all while providing an intuitive user interface.

1.1 Functional Requirements

1.1.a Login and Account Info

As a user,

- I would like to be able to create a unique login username and password.
- I would like to be able to reset my password if I am not currently logged in with a forgot password option.
- I would like to be able to view my account profile
- I would like my account to display and save any personal content including my app settings, past sleep data, login information, dream journal, and community posts.

- I would like to be able to manage my account while logged in and take actions such as changing my password, deleting my account, and changing my username.
- I would like to be able to personalize my profile with an account photo, name, and color scheme.

1.1.b Sleep Log

As a user,

- I would like to be able to manually log my sleep times.
- As a user who owns a smart watch, I would like to automatically sync my device with the app so that I can see my sleep duration, quality and patterns.
- I would like a page that shows me a bar graph detailing how much sleep I got each day over the past month.
- I would like to be able to set sleep duration goals.
- I would like to receive in-app rewards for meeting my sleep duration goals.
- I would like to be able to see my in-app rewards

1.1.c Alarm

As a user,

- I would like to have an alarm that can wake me up during my lightest sleep phase every morning.
- I would like to receive a reminder to wind down and begin my bedtime routine every night based on my desired and recommended sleep schedule each day.

1.1.d Sleep Analysis

As a user,

- I would like a page that shows a sleep analysis in the form of a graph.
- I would like to be able to observe my sleep environmental conditions such as room temperature, noise levels, and light exposure.

- I would like this page to show any external noise, snoring, and movement in a bar graph form.
- I would like the app to use values from this graph, my sleep stage durations, and my sleep duration to calculate a “sleep score” based on my duration of sleep and any times I was awake during the night.
- I would like to see what percentage of my sleep was deep, REM, or light sleep and the duration of my deep, REM, and light sleep.
- I want to be able to see a long-term comparison analysis of my sleep patterns by duration and sleep score.

1.1.e Chatbot

As a user,

- I would like to be able to interact with an online AI chatbot with a chat interface.
- I would like to be able to get accurate answers to my questions and queries from the chatbot.
- I would like the chatbot to offer information about healthy sleep patterns.
- I would like any chatbot to specialize in this app, and only provide information regarding my sleep and features this app has.
- I would like the chatbot to learn from my preferences and improve its recommendations over time.
- I want the chatbot to provide tips on improving sleep quality.
- I would like to be able to use the speech-to-text option to interact with the chatbot.

1.1.f Community Forum

As a user,

- I would like to be able to send friend requests to selected users to build my own community of friends.
- I would like to have an outlet to connect with others through a community forum.

- I would like to share and receive sleep experiences, tips and advice about sleep-related topics on this forum.
- I would like to be able to see forums of various topics, with each forum's content restricted to only covering its particular subject.
- I would like to have a direct messaging system to communicate with one user at a time.
- I would like to be able to directly message multiple users in a group chat. (If time allows)
- I would like to be able to block certain users so they cannot see my posts or add me as a friend (if time allows)

1.1.g Dream Journal

As a user,

- I would like to have a separate page where I can post my dream journal to a public page for other users to view.
- I would like to have a space where I can record and reflect on my dreams.
- I would like to be able to discuss my dreams with others in a comment-style.
- I would like to share my own comments, thoughts, and dream journals exclusively with my in-app friends.
- I would like to be able to change settings on my dreams journals, so I can choose to make them public, private, or only available to friends.
- I would like to add tags to the dream journals I post.
- I would like the app to provide a summary of recurring themes I have been seeing in my dream.

1.1.h Personalized Recommendations and Meditation

As a user,

- I would like to receive personalized tips and recommendations based on my sleep data and habits.

- I would like to see any suggestions or insights into my sleep cycle.
- I want to be able to view articles, videos, and tips on sleep hygiene, the science of sleep, and strategies for improving sleep quality.
- I would like to be able to have options to connect with sleep specialists and/or healthcare providers.
- I would like a page that brings up articles from scientific sources on how I can get better sleep.
- I would like to have access to relaxation techniques, such as guided meditations, breathing exercises, and ambient sounds.
- I would like relaxation techniques to be recommended to me based on my profile.
- I would like to be guided through relaxation techniques such as breathing exercises.
- I would like to get personalized sleep tips and recommendations using supplementary data from other IOS apps.

1.2 Non-Functional Requirements

System Requirements

- Our app should be compatible with iOS devices.
- Our app should integrate Google/Apple Maps to connect users with local doctors.
- Our app should integrate Apple HealthKit API to collect and process user data for sleep analysis.
- Our app should integrate with Apple Watch and/or Fitbit to produce accurate sleep analysis.
- Our app should integrate Firebase as the backend web server service.
- Our app should run 24/7 in the background to track the sleep environment and collect data throughout the night.

User Requirements

- Users will need some sleep tracking technology such as a Fitbit or smart watch.

Data Privacy and Security

- Our app should ensure high levels of security for user data, including encryption and privacy controls via Firebase
- Our app should implement a Firebase-based backend that will offer a secure authentication system for user verification which will effectively protect unauthorized users from accessing data that does not belong to them.

Scalability

- Our app should be able to handle requests from multiple users at the same time. Using Firebase's cloud functions will greatly increase our flexibility when it comes to our project's scalability as it will allow us to scale out when more resources are necessary.

Usability

- Our app should be user-friendly and easy to navigate.
- Our app should be able to support up to 1000 concurrent connections at any given time, and run 24 hours per day.

Response Time

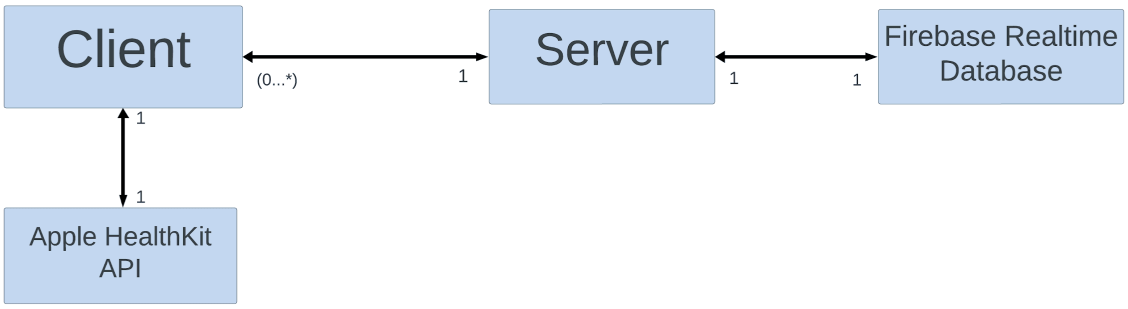
- Our app should have a fast request and response time.
- Our app should optimize bandwidth usage.
- Our app should have an average response time of 1 second, while handling up to 1000 requests per second.

2. Design Outline

Our project is an iOS sleep tracking app designed to help users understand their sleep patterns, share insights, and connect with a community focused on improving sleep quality. The app operates on a Client-Server model, where a single server handles multiple client connections simultaneously. Through a user-friendly iOS interface, users can access processed data, gain insights, and receive personalized sleep improvement recommendations. The server interacts with a firebase database to store and manage user sleep data, informative articles, and user IDs. To increase the app's core functionality, it integrates with Apple HealthKit API to incorporate additional sleep-related metrics into the analysis. This integration enables the app to provide a detailed and enriched sleep analysis, ensuring users receive the most accurate and helpful information for their sleep improvement journey.

2.1 High Level Overview

This project is designed to support numerous clients simultaneously interacting with the server, each capable of requesting sleep data and comprehensive analysis results. Upon receiving a request, the server will interact with the Firebase Realtime Database to fetch the required sleep data. The server will dispatch this processed data back to the clients. When a client requests a detailed analysis of a specific sleep cycle, the server will retrieve the corresponding data for that day from the Firebase Realtime Database, and transmit the comprehensive analysis to the requesting client. Client-side operations, like sorting or filtering the history of sleep sessions, are handled locally to reduce server load and enhance the responsiveness of the user interface. Data is also processed and augmented with additional metrics obtained from Apple HealthKit API, and organized accordingly. The Firebase Realtime Database plays a critical role in real-time data synchronization across client devices, ensuring that users have access to the most up-to-date sleep data.



1. Client
   - The client provides the user with a graphical interface to interact with the sleep tracking system, displaying sleep patterns, insights, and recommendations.
   - Can interact with Health Kit API directly without involving the server, and makes health data requests directly to the HealthKit API
   - The client sends processed sleep data or analysis requests to the server, receiving back comprehensive insights and processed information.

2. Server
   - The server acts as an intermediary between the iOS clients and the database.
   - The server also handles the logic for user authentication, data processing, and compilation of the sleep analysis in a structured format to be sent back to the client.
   - The server compiles the processed data into a structured format and sends it back to the client, ensuring efficient data transmission and integrity.

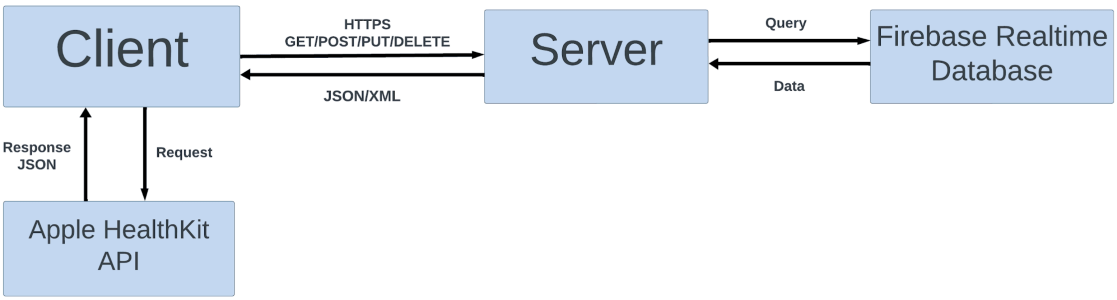
3. Database
   - The database is designed to store comprehensive sleep data, user profiles, preferences, and historical analysis securely and efficiently.
   - Receives processed data from the server
   - It handles complex queries from the server, including insertions, updates, and retrieval of data.

4. Apple HealthKit API

- Serves as a gateway between the client app and the device's health sensors.
- Provides access to sleep-related data like sleep stages, heart rate, and motion data.
- Processes and stores some basic sleep data on the device.

2.2 Overview of the Client/Server/Database Architecture

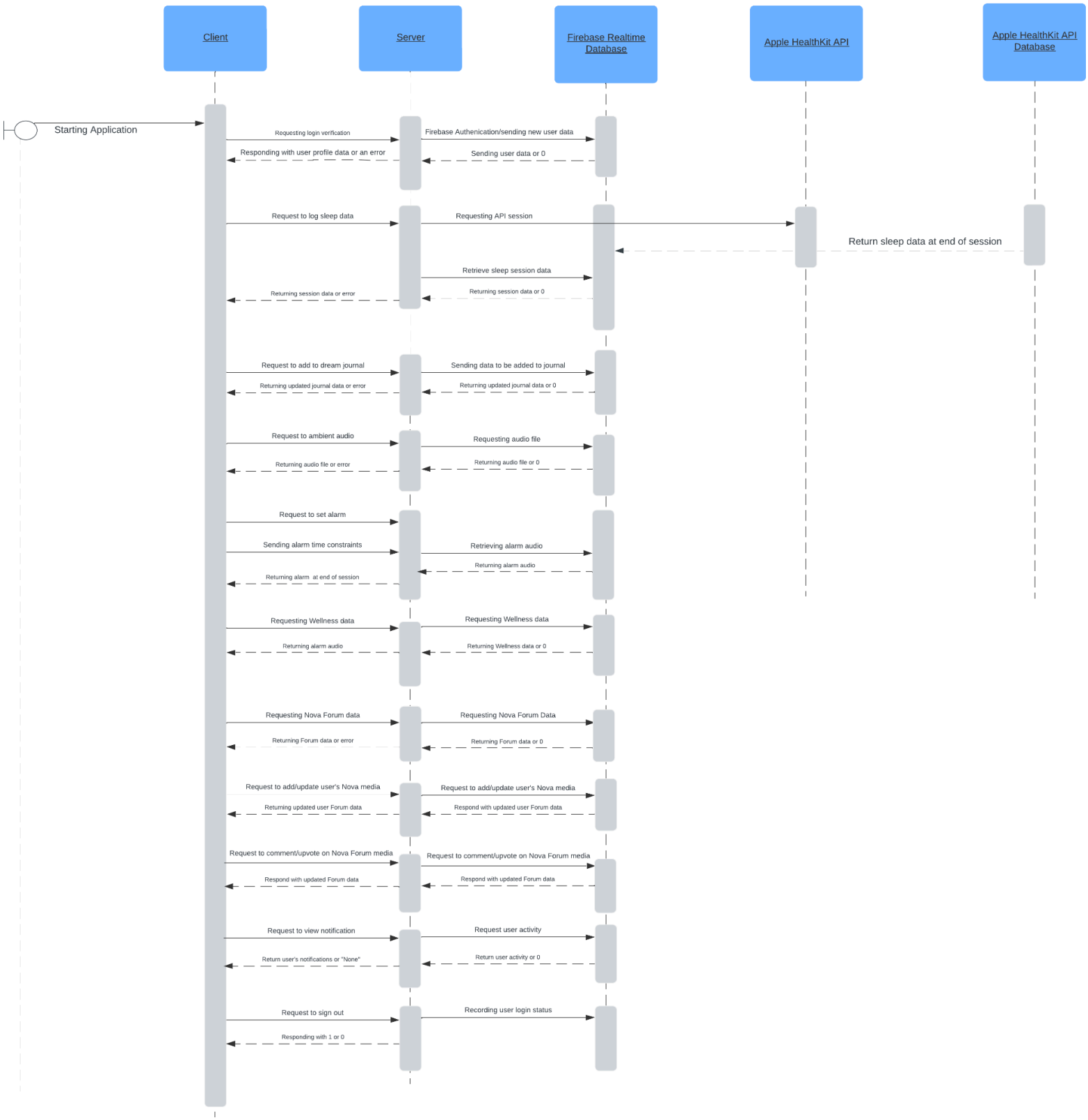
Our application will utilize a streamlined client-server architecture using Firebase, a comprehensive Backend-As-A-Service solution. The **client** will send and receive requests via HTTPS, which the server will receive through Firebase Hosting/REST API, and satisfy the request by communicating with Firebase real-time database. Examples of such requests sent via HTTPS GET, POST, PUT, DELETE commands would be creating new users/user login using Firebase Authentication, storing and sharing sleep data, recording sleep journals, sending messages between users, and activities on the social platform part of the application.



2.3 Sequence of Events Overview

Broad Overview: The user starts with the client and provides their account information. The REST API takes this information from the client and sends it to the database to update the data associated with the account. By having the API interact with the database, data can be uploaded and modified in a more efficient manner. Having the client and API talk to each other makes it easier to manage what information the user

sees. One constraint of this may be that it makes the process of updating and retrieving data more involved, even though the process will be more organized.



3. Design Issues

3.1 Functional Issues

Functional Issue #1: How many types of users should we involve in our application?

- Options:
    - **One type of user - all users are of the same type/have access to the same features and functionality of the app.**
    - Two types of users: “Free” users and paying subscribers, with the subscribers having access to extended/additional features.
    - Including an additional user type for certified doctors/sleep specialists.
- Choice: We opted to create only one type of user for this application.
- Reasoning: Because most of the features and APIs we are integrating are free or of minimal cost, we do not have a need to include a premium membership/have paying subscribers. Additionally, ensuring the status of users to be registered as “certified” users was difficult for us to verify. Therefore, to prevent any liability issues, we will have one universal type of user with no certifications.

Functional Issue #2: What kind of social features should our application involve?

- Options:
    - No communication between users
    - Simpler social features: private messaging between friends
    - **A direct messaging platform and a page where users are able to upload content for other users to see.**

- Choice: Integrating a more complex social aspect for our application that follows the structure of a social platform
- Reasoning: When designing our app, we wanted one of the features that will set it apart from existing sleep tracking apps to be community engagement. We want our users to be able to communicate, ask questions, and share stories via a messaging system and a page where they can upload and view sleep-related content/tips in the form of a social media page. With only a private message feature, there are less options for users to interact.

Functional Issue #3: Should include personalized recommendations for improving sleep habits?

- Options:
    - No
    - **Yes**
    - Yes, but only general and non-personalized.
- Choice: Yes - include personalized recommendations.
- Reasoning: We are looking to create an algorithm to produce the most relevant tips for a user based on logged sleep patterns. This adds a complexity to our app that extends from offering general sleep recommendations. However, when implementing this option, the personal recommendations will be more superficial and no medical advice will be given (disclaimers provided).

Functional Issue #4: What direct message features should we include?

- Options:
    - Direct Message between a single User and one of that user's friends

    - Direct Message between a single User and someone who is not one of that user's friends
    - **Allow a user to make a group chat and add only friends to a group chat and also allow for private messages between 2 friends**
    - Allow a user to make and add friends and any other users to a group chat
- Choice: Allow a user to make a group chat and add only friends to a group chat and also allow for private messages between 2 friends.
- Reasoning: With all of the user interaction on our application, we want users to feel comfortable. If another user who does not know this user gets added to a chat or group chat without their permission or knowledge, it causes some privacy issues.

Functional Issue #5: How will we get sleep data from users?

- Options:
    - Only from the user manually inputting their data
    - Only syncing to a user's watch or other sleep tracker
    - **By letting the user choose to log manually or sync with a tracker**
- Choice: Letting a user choose to log manually or synch with a tracker
- Reasoning: We don't want our app to be limited to a certain kind of user. We want those without personal sleep tracking devices to still use and benefit from our app. We also want those who have sleep tracker devices to see all of their sleep data that their tracker provides for them as well.

Functional Issue #6: How will the alarm be set up?

- Options:

    - The user can manually set an alarm for a certain time
    - An alarm will go off during the user's lightest sleep phase inside a time window set by the user
    - **Both**

- Choice: Both a manually set alarm and alarm during the lightest sleep phase
- Reasoning: Most sleep tracking apps only let a user set an alarm manually. We are looking to add a new aspect of an alarm. While it is still important that the user is able to set their alarms manually, our alarm will go off during the user's lightest sleep phase during a time window set by the user in order to ensure that the user is well rested and refreshed.

Functional Issue #7: What information do we need from users when creating an account?

- Options:
    - Full Name, Username, Password
    - Full Name, Email, Password
    - **Full Name, Username, Email, Password**
- Choice: Full Name, Username, Email, Password
- Reasoning: When setting up an account, the main things needed to verify a user is an email and a password, so it is important to have at least those elements. A user's username will help with extra verification and customization for the user when creating an account.

3.2 Nonfunctional Issues

Design Issue #1: Which iOS development language should we use?

- Options:

    - **Swift**
    - Objective C
    - C# with .NET framework

- Choice: Swift
- Reasoning: After discussion within our group, we opted to utilize Swift in creating our product. As a group, we have limited to no experience with developing iOS applications, a few members having experience with Objective C while a few other members having basic knowledge of Swift. After reviewing the compatibility of Swift with the various APIs we intend to use, it became clear that it would be the most efficient option.

Design Issue #2: How should we plan to develop our back-end web service?

- Options:
    - **Firebase**
    - MongoDB
    - Developing our own server and database
- Choice: Back-end Provider like Firebase
- Reasoning: As a group with little to no experience with iOS development, we decided to utilize the backend provider Firebase because it is the easiest to implement in Swift. Due to the complexity and time needed in other areas of our project, developing our own server would not be an efficient option. To develop our own web server to service the needs of our app in the areas of data processing, scalability, usability, etc., would probably not be feasible if trying to meet our goals for user response time and scalability options. In addition, because of our

group's inexperience in integrating a server with an iOS application, we thought it would be best to use a backend service for security purposes in handling user data. The databases, features and functionalities that come with many backend services make it the best option for our group.

Design Issue #3: Which Backend-As-A-Service should we use?

- Options:
    - **Firebase**
    - AWS Amplify
    - Parse
- Choice: Firebase
- Reasoning: There were many features of our app to be taken into consideration when choosing a backend web service. The goal of our app is to provide real-time data analysis for sleep patterns, as well as interactive features such as a chat/social platform and an AI chat bot. As a group who is new to using BaaS services, we were looking for an option that supports real-time data, is compatible and easy to integrate with our APIs, and overall has features that are comprehensive to our project. After deciding that we wanted to use a NoSQL database, we had previously been considering MongoDB before opting to go with a BaaS service. When researching services that had similar support, Firebase seemed to have the best reviews for creating new iOS applications. While other AWS and Parse both offered similar services, Firebase had the best comprehensive options (including a hosting service and automation with cloud functions) making it the best option overall.

Design Issue #4: How should we ensure secure user login and data storage?

- Options (authentication):
    - Auth0
    - MongoDB
    - Okta
    - **Firebase Authentication**
- Choice: Firebase Authentication
- Reasoning: When considering the best approach to user authentication and security, our options were to use a more specialized service, such as Auth0 and Okta, that come with many configuration options or to stick with using only Firebase Authentication. While Firebase Authentication provides a very basic security service, if implemented correctly, we believe it to be sufficient for our project. While its features are more limited than if we chose to integrate an additional service, the authentication services it provides are along the lines of similar measures taken by similar applications on the market. Because we are already using Firebase as our database, it also greatly simplifies the process of setting up security measures using Firebase Authentication. In addition, the cost of using one of the other alternatives is significantly more than Firebase.

Design Issue #5: How should we gather data for sleep-tracking analysis?

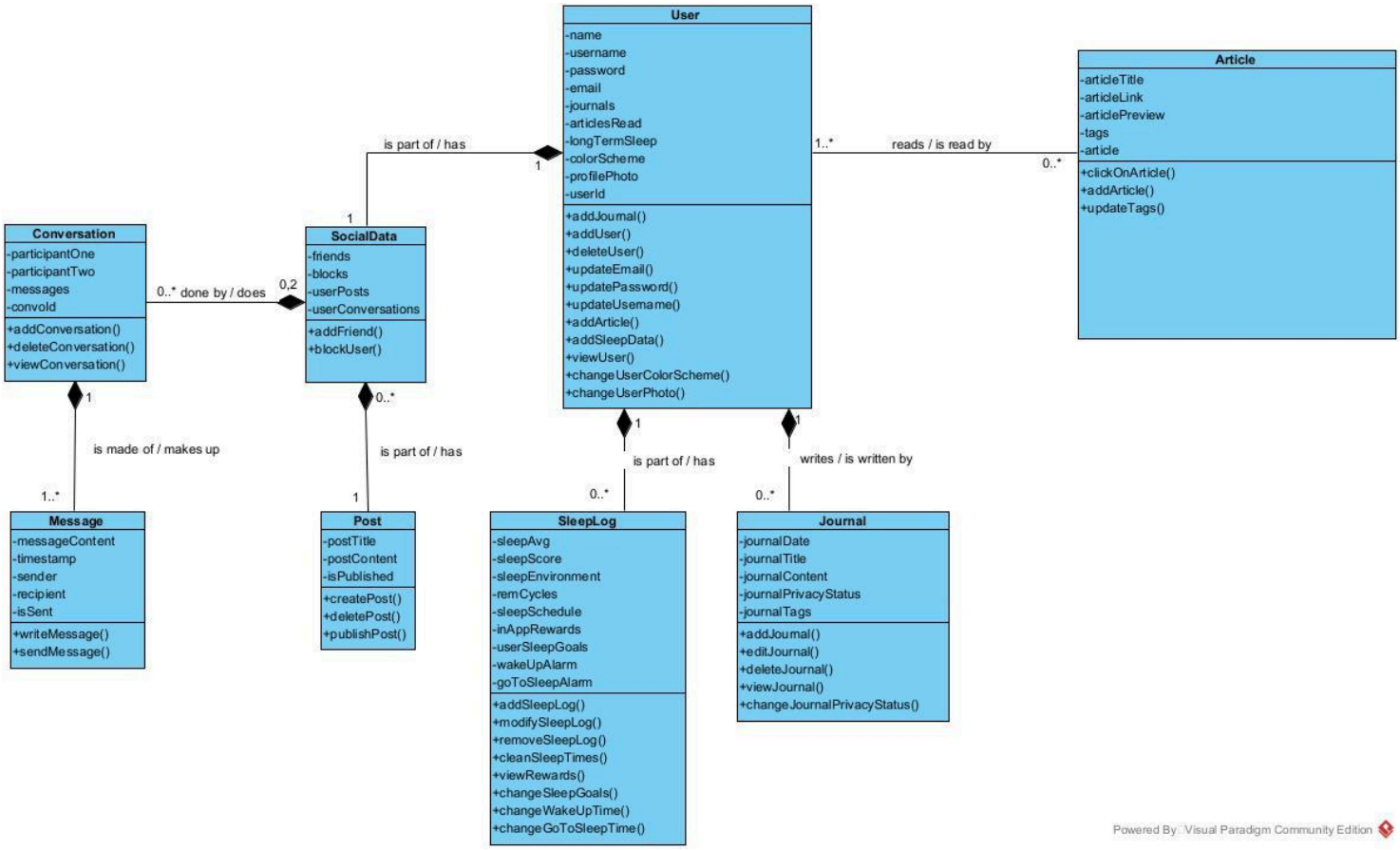
- Options:
    - **Apple HealthKit API**
    - Sleep API
    - Garmin Health API

    - ROOK API

- Choice: Apple HealthKit API
- Reasoning: When looking at different API's to integrate in order to gather data for sleep-tracking analysis, we decided that we would create an iOS app compatible with devices that used Apple's HealthKit API (Apple/iOS devices) and/or devices compatible with Google's Sleep API (android smartwatches, fitbits). These API's are compatible with the rest of our architecture and are free of cost. We understand that this limits our users to those who have either Apple/Google devices, but to integrate a third party API such as ROOK or to include even Garmin Health API would be significantly more complicated and costly.

4. Design Details

4.1 Class Diagram



4.2 Classes and Models Descriptions

The class diagram was constructed to reflect each object in the app: articles, journals, messages, conversations, posts, sleep logs, and users. User content was split into sections covering their journal information, social data, sleep data, and account data due to an excess of attributes and behaviors associated with the user.

User:

This class contains basic profile data that all users have. It has login information, account settings, and additional data unique to the user.

- Attributes:
  - *name:* The name of the user
  - *username:* The username the user uses to login
  - *password:* The password the user uses to login
  - *email:* The email associated with a user
  - *journals:* An array of all journal objects associated with a user
  - *articlesRead:* An array of articles a user read
  - *longTermSleep:* An array of SleepLog objects associated with a user
  - *colorScheme:* An enum for the user's choice color scheme for the app
  - *profilePhoto:* An image provided by the user for their account
  - *userId*: A unique ID assigned to each user upon account creation. This will help keep track of users in the database
- Behaviors:
  - *addUser():* Adds a user
  - *deleteUser():* Deletes a user and all information associated with that user
  - *addJournal():* Adds a Journal to the userJournals array
  - *updateEmail():* Changes a user's email
  - *updatePassword():* Changes a user's password
  - *updateUsername():* Changes a user's username
  - *addArticle():* Adds an Article to the userArticlesRead array
  - *addSleepData():* Adds a SleepLog object to the userLongTermSleep array
  - *viewUser():* Allows the user information to be viewed
  - *changeUserColorScheme():* Changes the user's color scheme
  - *changeUserPhoto():* Changes the user's photo

SleepLog (composition of User)

This class will hold sleep data for each user. It will contain information about the user's overall sleep patterns, goals, and progress, which will be displayed on the *Sleep Analysis* view of the app.

- Attributes:
    - *sleepAvg:* The average hours of sleep a user gets each night
    - *sleepScore:* A user's sleep score
    - *sleepEnvironment:* Array of environmental factors influencing a users sleep
    - *remCycles:* Array of different REM cycles associated with a user
    - *sleepSchedule:* Typical sleep schedule for a user
    - *inAppRewards:* An integer that holds the amount of "points" a user has won
    - *userSleepGoals:* An array of structs that holds a user's goals and their progress towards those goals
    - *wakeUpAlarm:* An integer that represents the time to wake up
    - *goToSleepAlarm:* An integer that represents the time to go to sleep
- Behaviors
    - *addSleepLog():* Adds a Sleep Log for a User
    - *modifySleepLog():* Modifies a Sleep Log
    - *removeSleepLog():* Removes a Sleep Log
    - *cleanSleepTimes():* Checks the Sleep Log created by the automatic API and the manual entries and combines them into a cohesive Sleep Log to prevent duplication
    - *viewRewards():* Allows the user to see the rewards they have earned for meeting goals
    - *changeSleepGoals():* Updates the user's sleep goals based on how their progress has been
    - *changeWakeUpTime():* Changes the time the alarm rings to wake the user up
    - *changeGoToSleepTime():* Changes the time the alarm rings to tell the user to sleep

SocialData (composition of User)

This class will include overall information about each user's friends, who they have blocked, what posts they have made, and who they are currently having conversations with. The information in this class will aid with displaying the *Nova Forum*, friends list, and direct messages for each user.

- Attributes:
    - *friends:* An array of Users to indicate a user's friends
    - *blocks:* An array of Users to indicate who a user has blocked
    - *userPosts:* An array of Post objects to indicate posts a user has made
    - *userConversations:* An array of Conversation objects to indicate conversations a user is a part of
- Behaviors:
    - *addFriend():* Allows a user to add another user as their friend
    - *blockUser():* Allows a user to block another user

Post (composition of SocialData)

This class will hold information about posts that the user is making on the *Nova Forum*, including the name of the post and its content. The information from this class will be used when displaying the *Nova Forum* page of the app.

- Attributes:
    - *postTitle:* A string that is the title of the post
    - *postContent:* A string that is the content of the post
    - *isPublished:* A boolean that indicates if the post is published or not
- Behaviors:
    - *createPost():* Creates a post
    - *deletePost():* Deletes a post
    - *publishPost():* Publishes a post

Conversation (composition of SocialData)

This class will hold each conversation that is transpiring between two Serenova users. It will hold information about the participants and the messages, which will be displayed on the direct messaging page for each user.

- Attributes:
    - *participantOne:* A string that is the username of some User object
    - *participantTwo:* A string that is the username of some other User object
    - *messages:* an array of message objects

    - *covoId*: a unique ID to help keep track of conversations in the database
- Behaviors:
    - *addConversation():* Creates a conversation between two Users
    - *deleteConversation():* Allows a user to delete a conversation between them and another user
    - *viewConversation():* Allows a user to view a conversation between them and another user

Message (composition of Conversation)

This class will hold information about each individual message that users are sending each other, including the content and other related metadata. The details of these messages will be displayed in the direct messaging page of the app.

- Attributes:
    - *messageContent:* A string of the actual message sent
    - *Timestamp:* Uses date struct to save the time and date a message has been sent
    - *Sender:* A User object who sent the message
    - *Recipient:* A User object who received the message
    - *isSent:* A boolean indicating if a message has been sent or not
- Behaviors:
    - *writeMessage():* Allows a user to write a new message to someone
    - *sendMessage():* Allows a user to send a written message to someone

Journal (composition of User)

This class will hold information regarding an individual journal written by a user. This information will be displayed on the Dream Journal page for the user that wrote the journal, and, depending on the settings, other users as well.

- Attributes:
    - *journalDate:* A date struct that is the time the journal was created
    - *journalTitle:* A string that is the title of the journal
    - *journalContent:* A string that is the journal's contents

    - *journalPrivacyStatus:* A boolean that indicates if the journal can be viewed by no other users, friends only, or any user
    - *journalTags:* An array of strings
- Behaviors:
    - *addJournal():* Allows a user to create a new Journal object
    - *editJournal():* Allows a user to edit a Journal object
    - *deleteJournal():* Allows a user to delete a Journal object
    - *viewJournal():* Allows a user to view their Journal
    - *changeJournalPrivacyStatus():* Allows users to see who else can see their Journal object

Article

This class will hold information for each article included in the app. It will be displayed on the Wellness Library page and may be sorted according to its content and/or the *tags* it contains.

- Attributes:
    - *articleTitle:* A string that is the article title
    - *articleLink:* A string that is the article link
    - *articlePreview:* A string that is part of the beginning of the article (limited to 100 characters)
    - *tags:* An array of strings
    - *article*: A unique string to help keep track of articles in the database
- Behaviors:
    - *clickOnArticle():* Opens up the link attached to an article
    - *addArticle():* Adds an article to a user's feed
    - *updateTags():* Changes the tags associated with an Article object

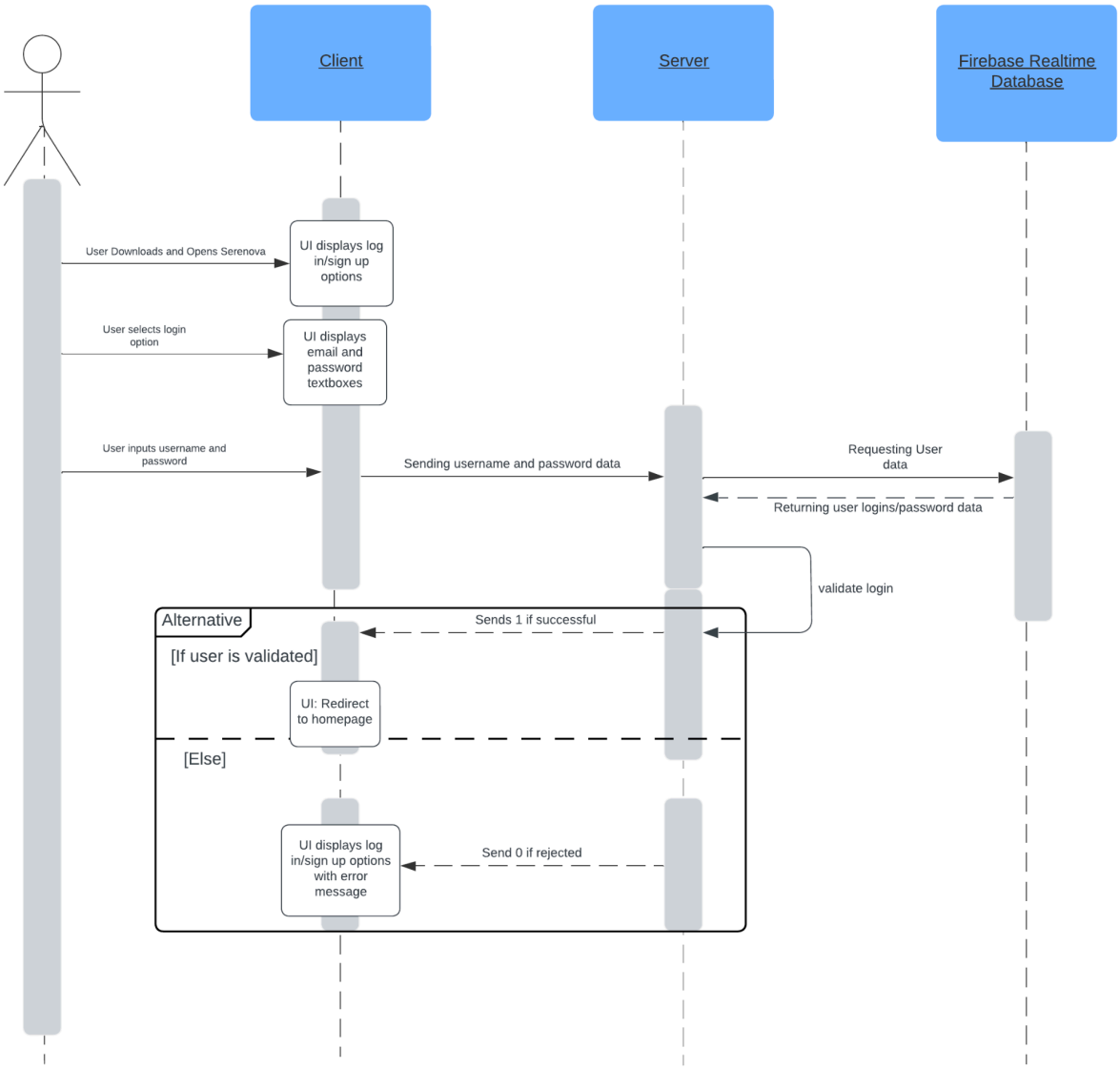
4.3 Sequence and Activity Diagrams

Sequence Diagrams:

The following diagrams show the sequence of events that occur between the User, Client, Sever, and Database architecture. They show the flow of requests and responses, showing the external APIs where applicable. These diagrams cover the main functionalities of user login, logging the sleep data, and interactions between multiple users via the Community Forum.

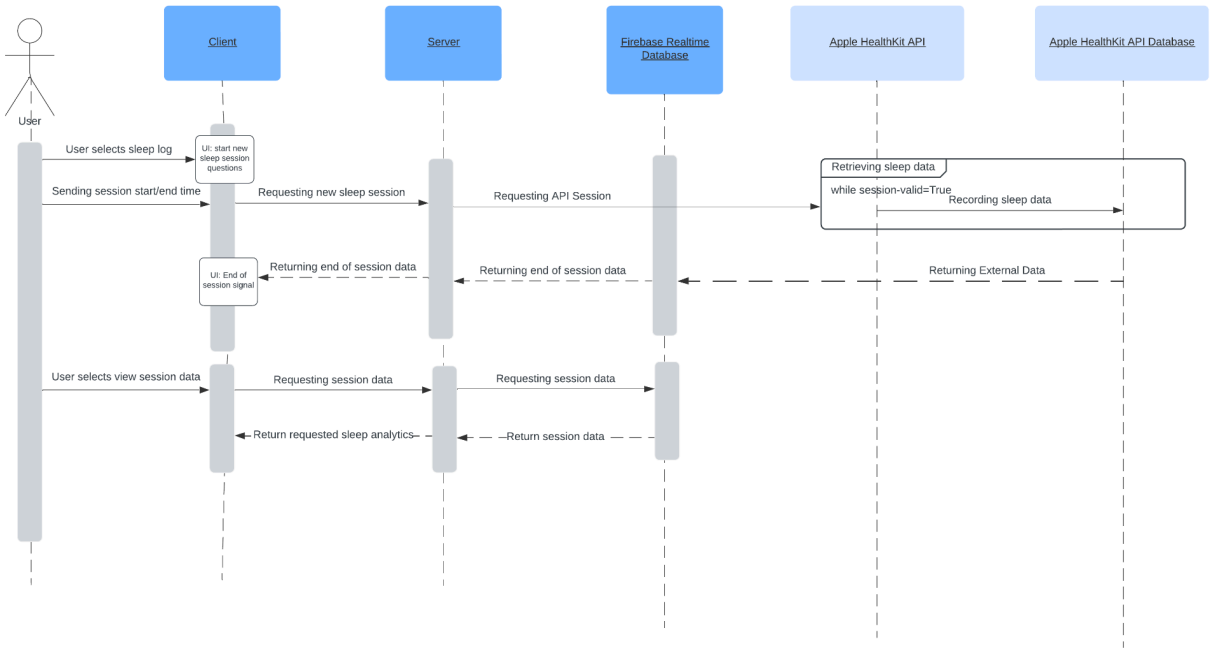
User Login:

When a user opens the app and is not signed in, they will be prompted to either log in or create an account. Upon choosing the login option, they will be prompted to enter their account username and password. This data will be sent to the server. The server will send a request to the database requesting for user login/password data. Once this data is retrieved from the database, the verification process will begin server-side via Firebase Authentication to verify the user. This process will continue until a login/signup process is validated.


Client
Server
Firebase Realtime Database
User Downloads and Opens Serenova
UI displays log in/sign up options
User selects login option
UI displays email and password textboxes
User inputs username and password
Sending username and password data
Requesting User data
Returning user logins/password data
validate login
Alternative
Sends 1 if successful
[If user is validated]
UI: Redirect to homepage
[Else]
UI displays log in/sign up options with error message
Send 0 if rejected

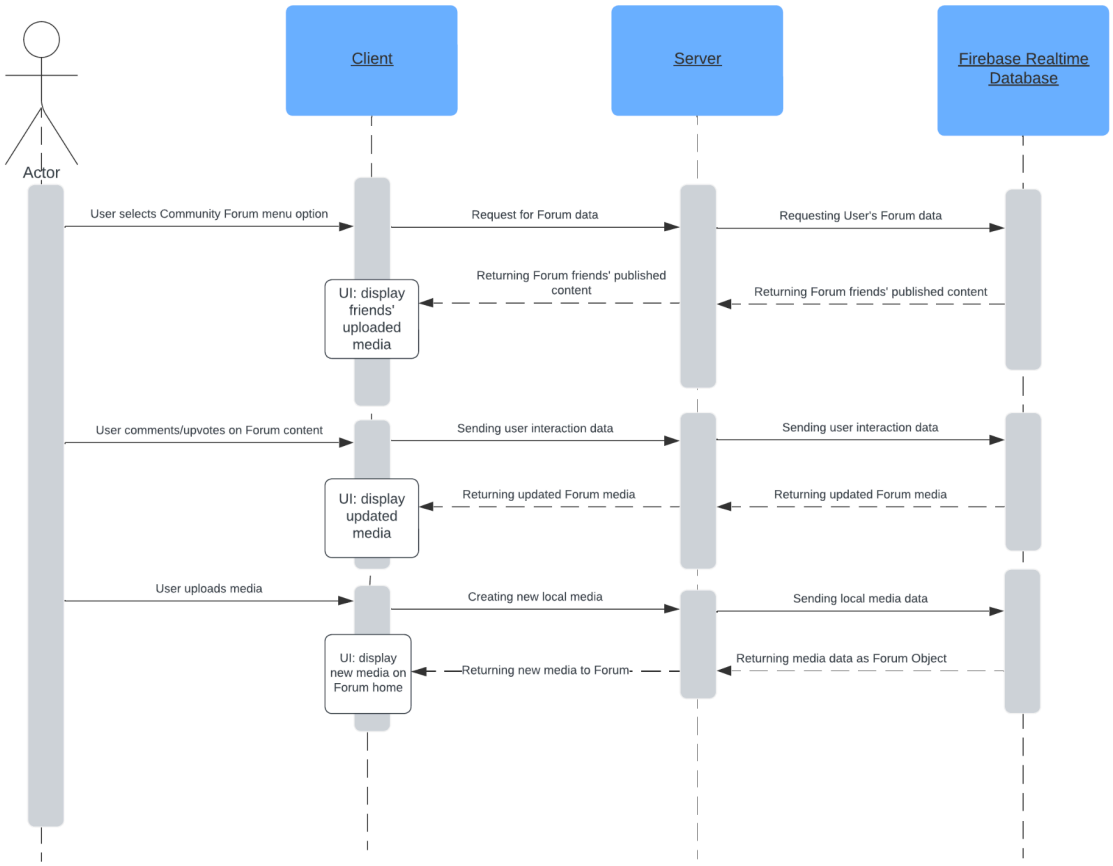
Logging Sleep Data/Viewing Logged Sessions

When a user wishes to log their sleep data, they will choose the alarm option in the menu and fill out the requested data and select “Start Sleep Session.” Once the session is started, the Apple HealthKit API will be integrated. The data gathered using the API will be recorded in the Apple Healthkit API External Database. Once the session is ended, the data will be returned and stored in the Firebase Realtime database and sent to the client side for the user to view.



Basic Nova Forum Activity

When a user selects the Community Forum menu option, a request will be sent to the server to retrieve the Forum data. Once the data has been retrieved, it will be returned to the server which will send the data back over to the client. Once the data is in the client side, the user will be able to view an organized display of the Forum and upload their own media to the forum for other users to see.

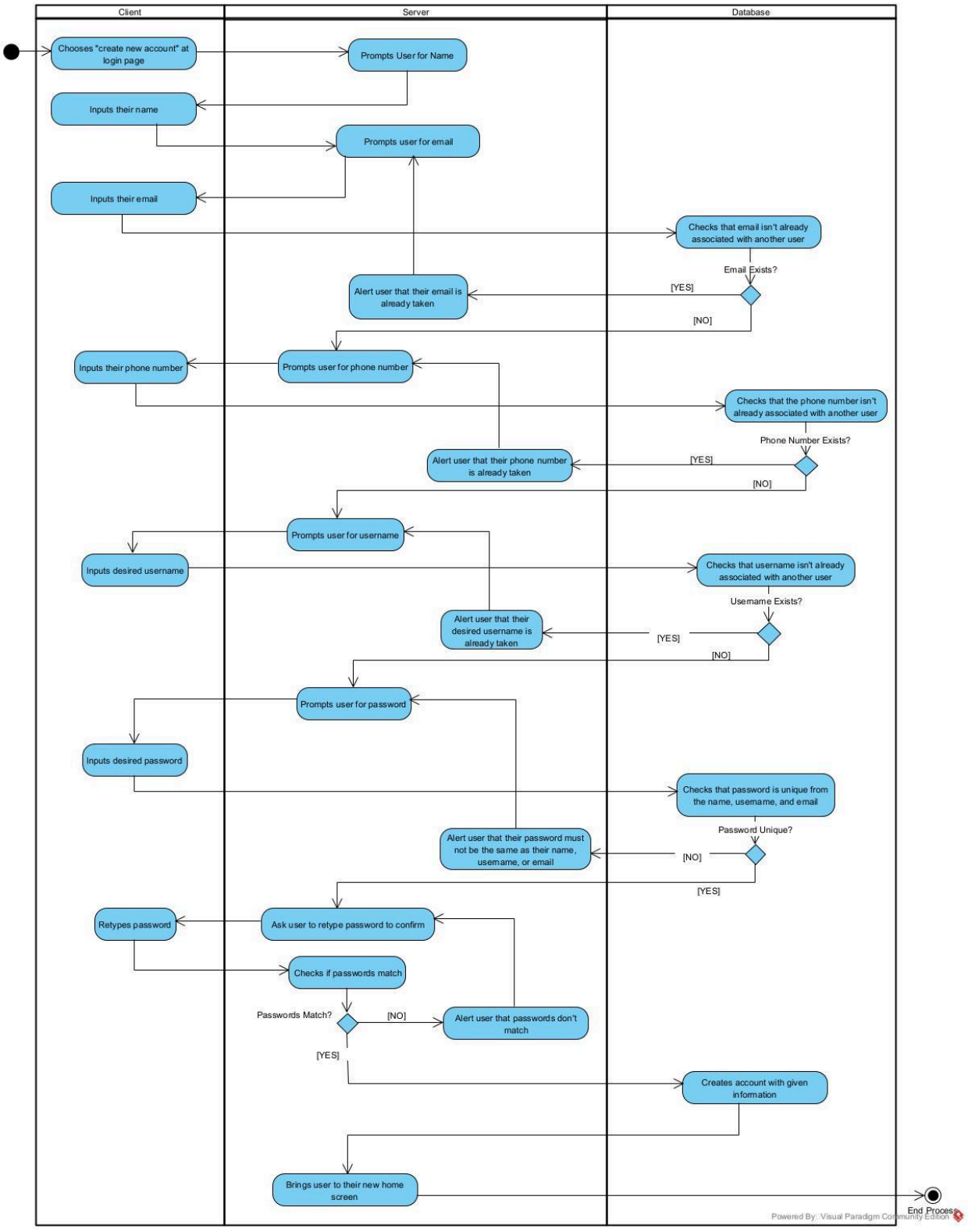


State/Activity Diagrams

In this section three of the main interactions between a user and a system will be exemplified. This will demonstrate some of the main interactions between a user and the app in different scenarios, such as creating an account, adding a friend, and logging sleep times.

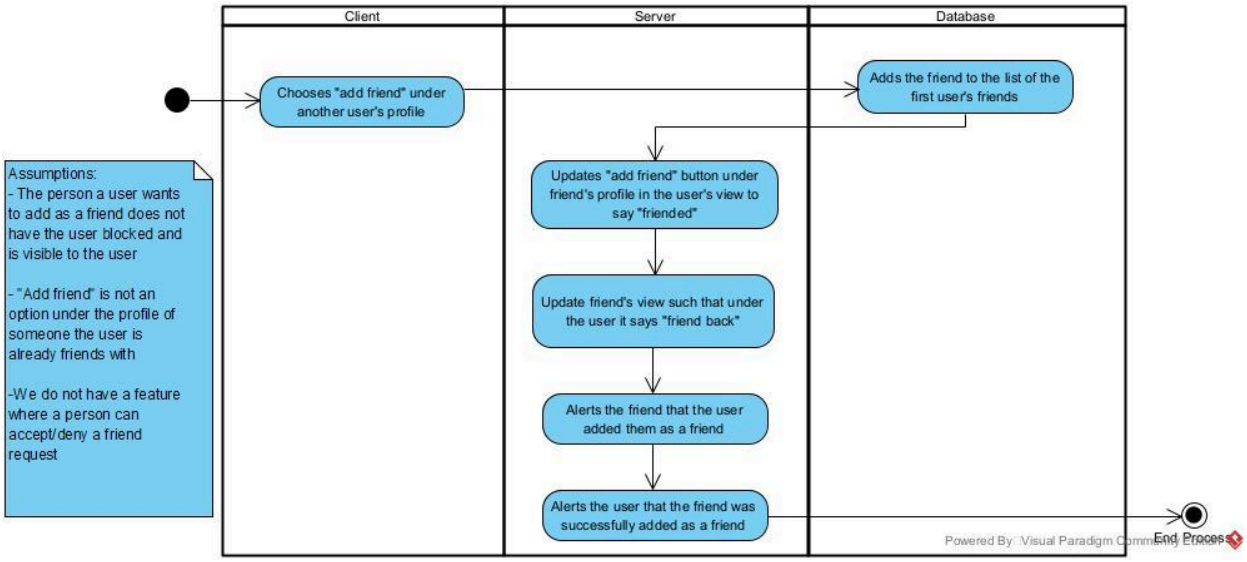
User Story #1: As a user, I would like to be able to create a unique login username and password.

This diagram will model the process of a user inputting their name, email, phone number, desired username, and desired password. It demonstrates how the system will prompt the user and check the database to ensure there are no duplicate emails, phone numbers, or usernames.


Client
Server
Database
Chooses "create new account" at login page
Prompts User for Name
Inputs their name
Prompts user for email
Inputs their email
Checks that email isn't already associated with another user
Email Exists?
[YES]
Alert user that their email is already taken
[NO]
Inputs their phone number
Prompts user for phone number
Checks that the phone number isn't already associated with another user
Phone Number Exists?
[YES]
Alert user that their phone number is already taken
[NO]
Prompts user for username
Inputs desired username
Checks that username isn't already associated with another user
Username Exists?
[YES]
Alert user that their desired username is already taken
[NO]
Prompts user for password
Inputs desired password
Checks that password is unique from the name, username, and email
Password Unique?
[NO]
Alert user that their password must not be the same as their name, username, or email
[YES]
Retypes password
Ask user to retype password to confirm
Checks if passwords match
Passwords Match?
[NO]
Alert user that passwords don't match
[YES]
Creates account with given information
Brings user to their new home screen
End Process
Powered By: Visual Paradigm Community Edition

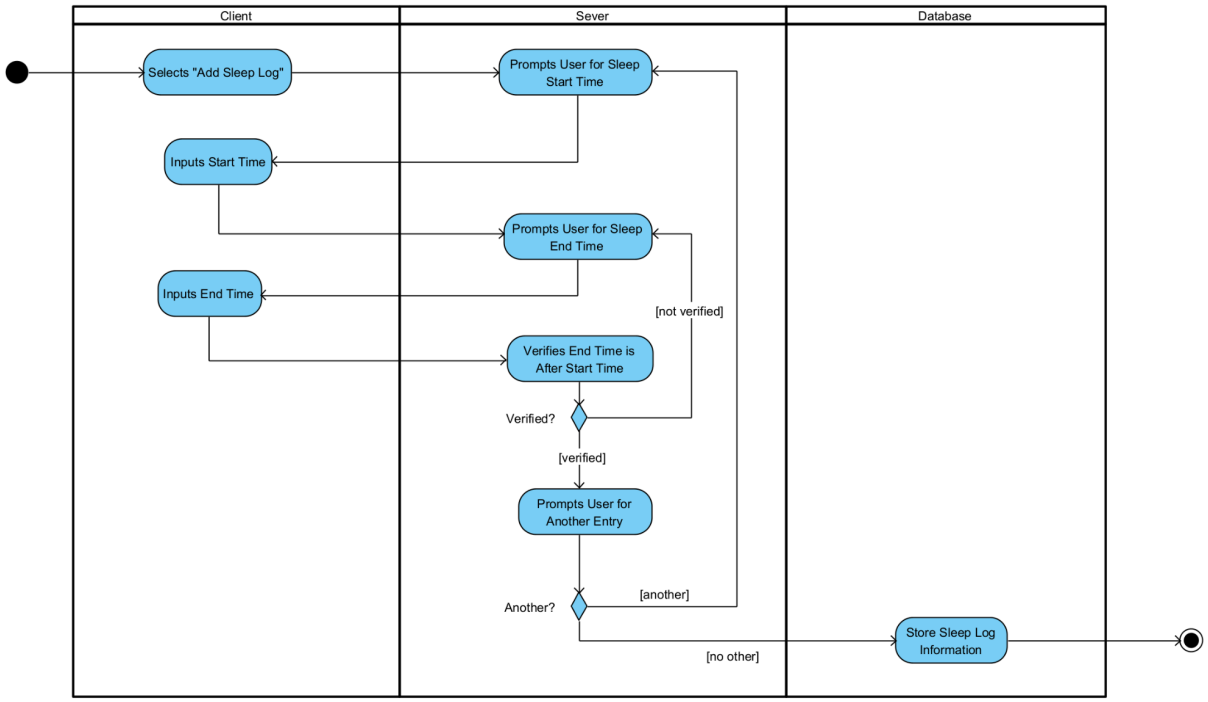
User Story #2: I would like to be able to send friend requests to selected users to build my own community of friends.

This diagram models how the system responds when a user adds another user as their friend. This includes modifying the SocialData to add another user as a friend.



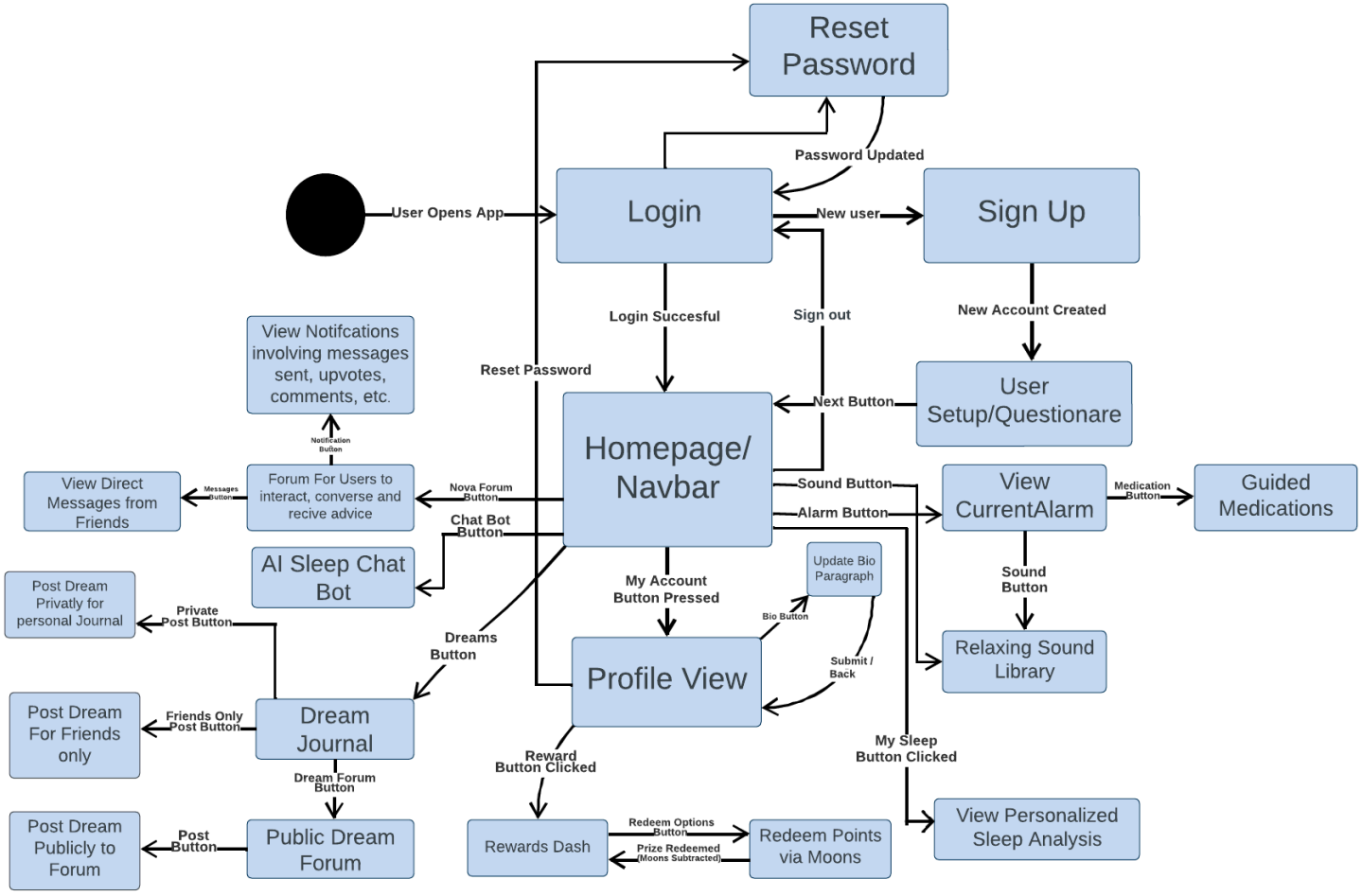
User Story #3: I would like to be able to manually log my sleep times.

This activity diagram models the process of a user manually adding a sleep log, inputting all the required information, and verifying that it is valid. Then, this information is stored in the database.



4.4 Navigation Flow Map

Our navigation design prioritizes straightforwardness and accessibility. Users are prompted to sign in on the main page using their account details. If they don't have an account, they're required to submit their user data. The main page showcases event details pertinent to the user. The navigation bar, present on each page of the iOS app, grants access to various sections including 'My Account', 'Dreams', 'Sounds', 'Alarm', 'My Sleep', 'Wellness', 'Chat Bot', 'Nova Forum', 'Notifications', and 'Sign Out'. Additionally, a back button on every page facilitates quick and efficient movement between sections.



This diagram shows the ease of access. All submodules and modules after login are easily accessible from the navigation bar from the side.

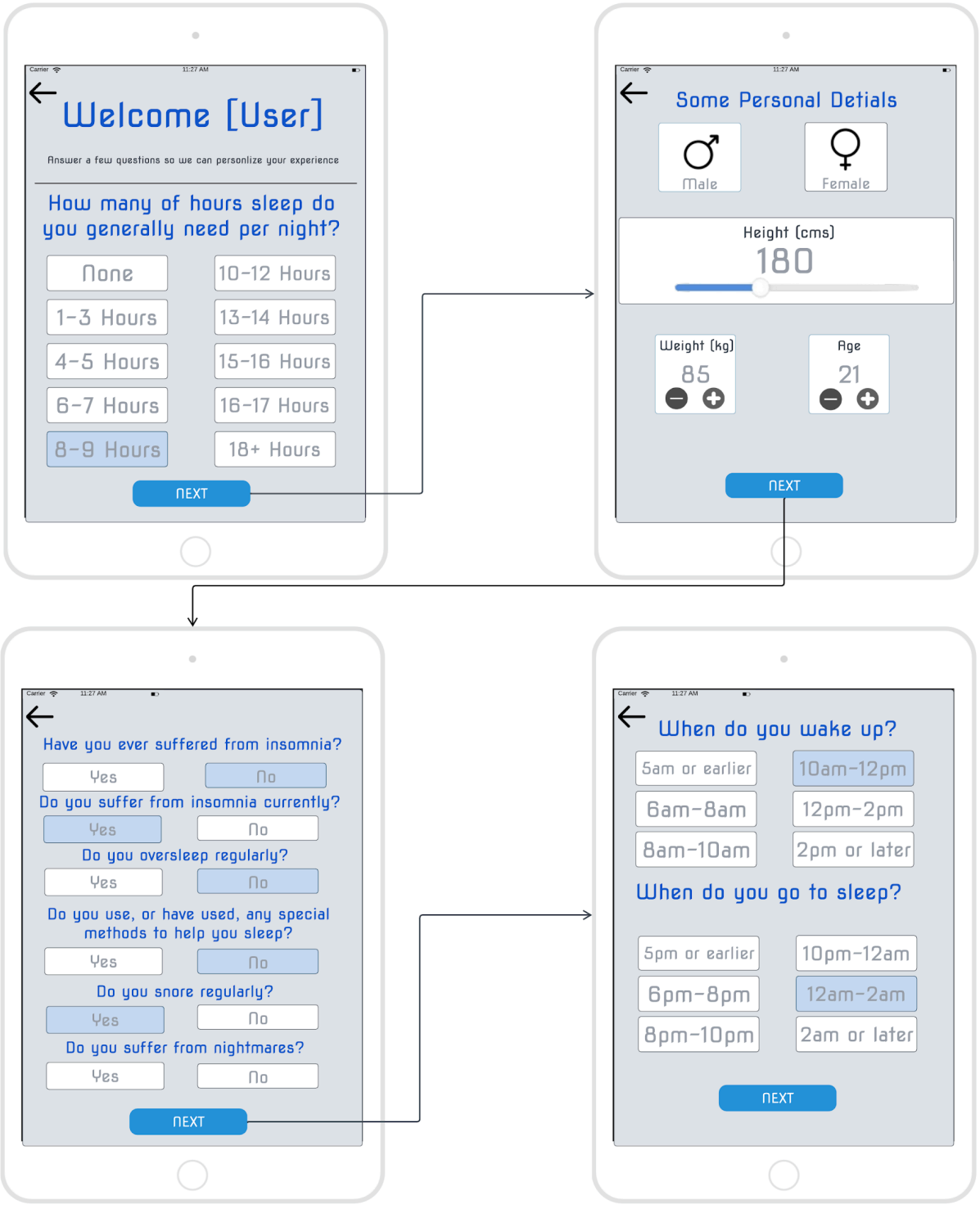
4.5 UI Mockup

First View & Login Process

Serenova will feature a loading screen for when users first open the app. To access their data, users will continue to the login page or the signup page to sign into their account.

User Onboarding

When users first open Serenova, they will be prompted to answer questions from a series of screens so that the app can familiarize itself with the user and provide a personalized experience.

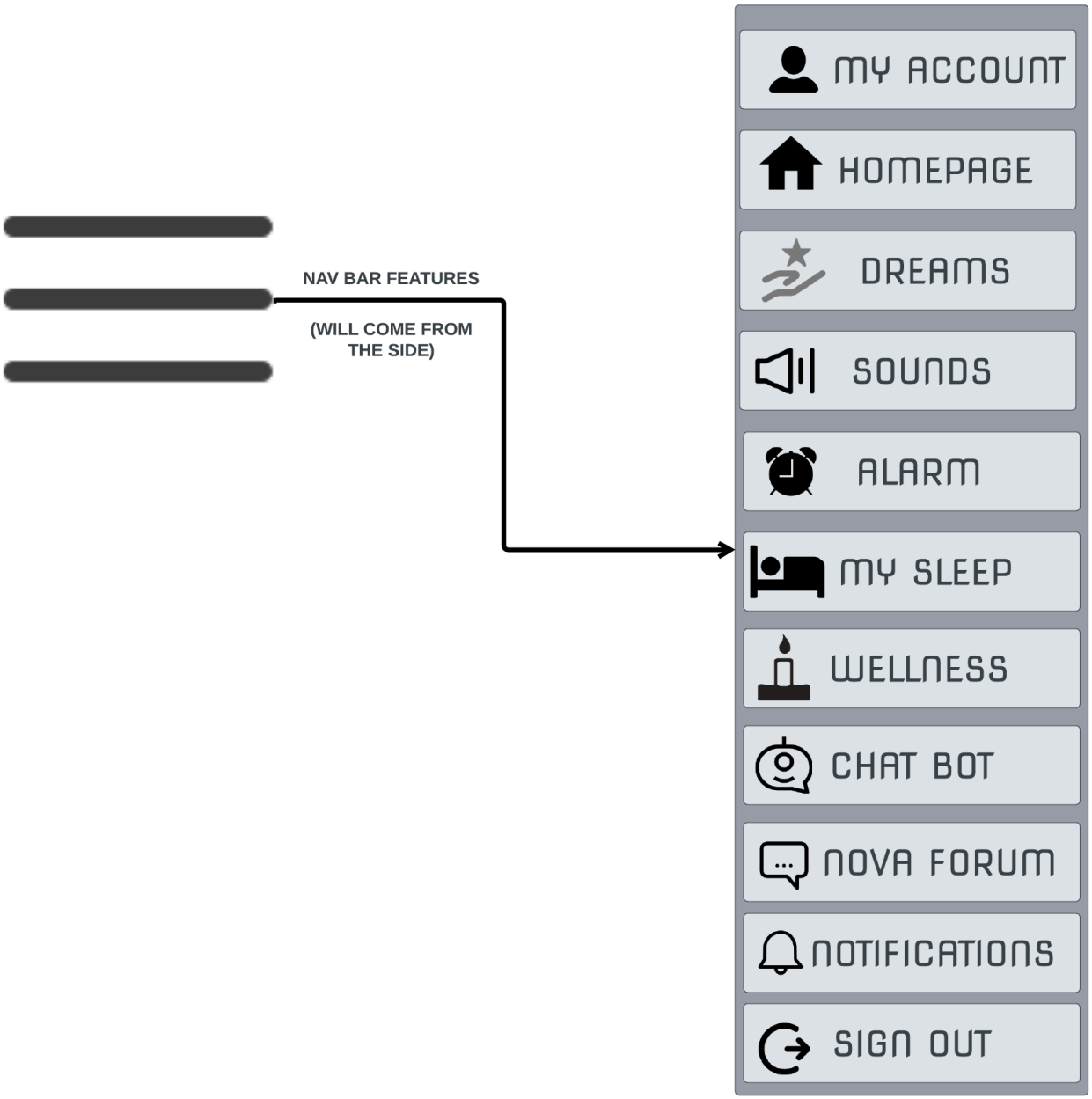


Dream Journal & Dream Sharing

Serenova will offer a separate space within the app for writing and sharing dream journals. Users will have a page for recording their dreams, a page for sharing their dreams within the *Dream Forum*, and a search option for finding specific types of dream journals.

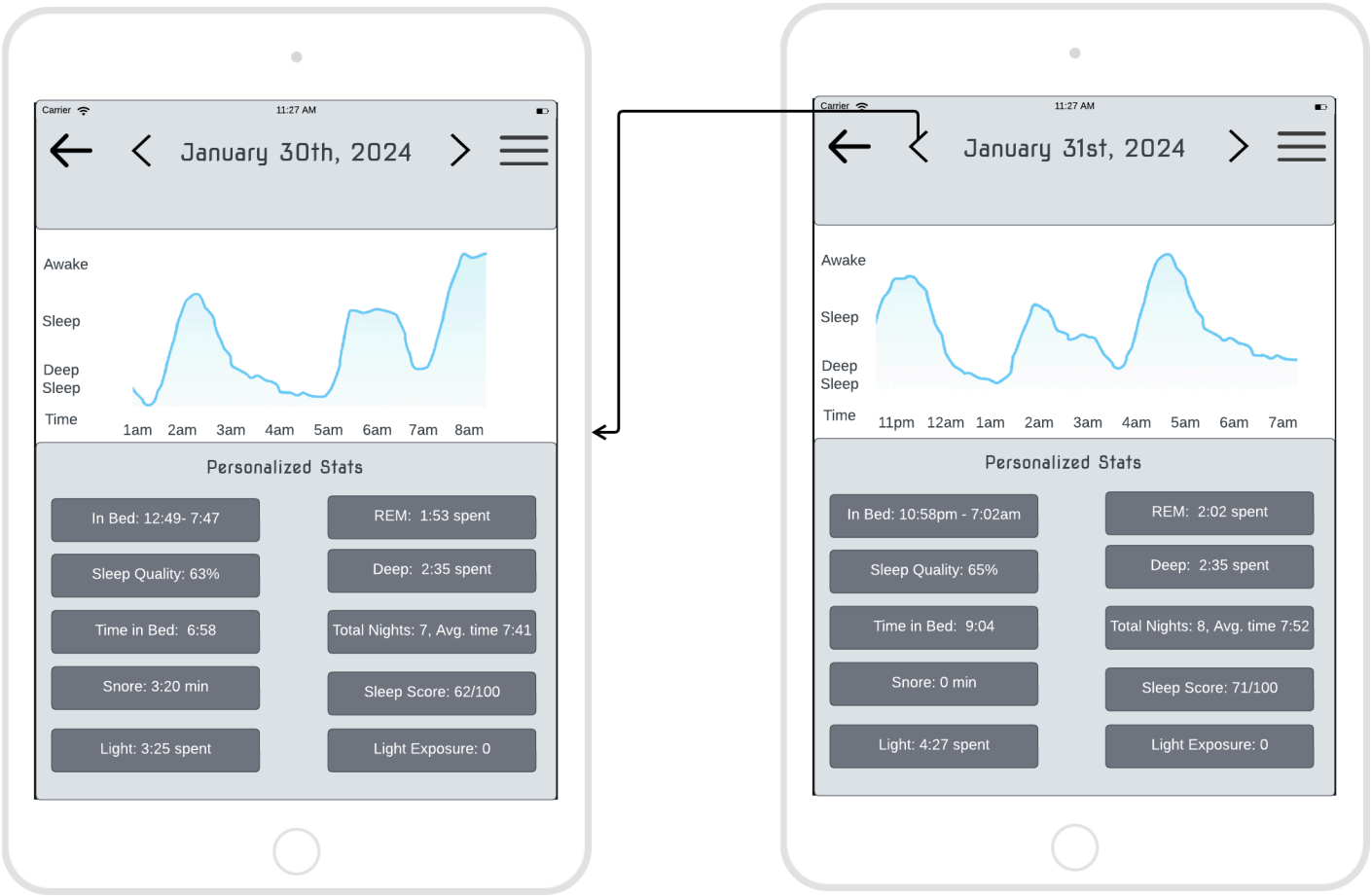
Navigation Bar

Serenova will have an intuitive, user-friendly navigation bar. All features of the app will be in one place for easy access.



Sleep Analysis View

Serenova will offer a *Sleep Analysis* page where users will be able to see their sleep metrics both graphically and textually.

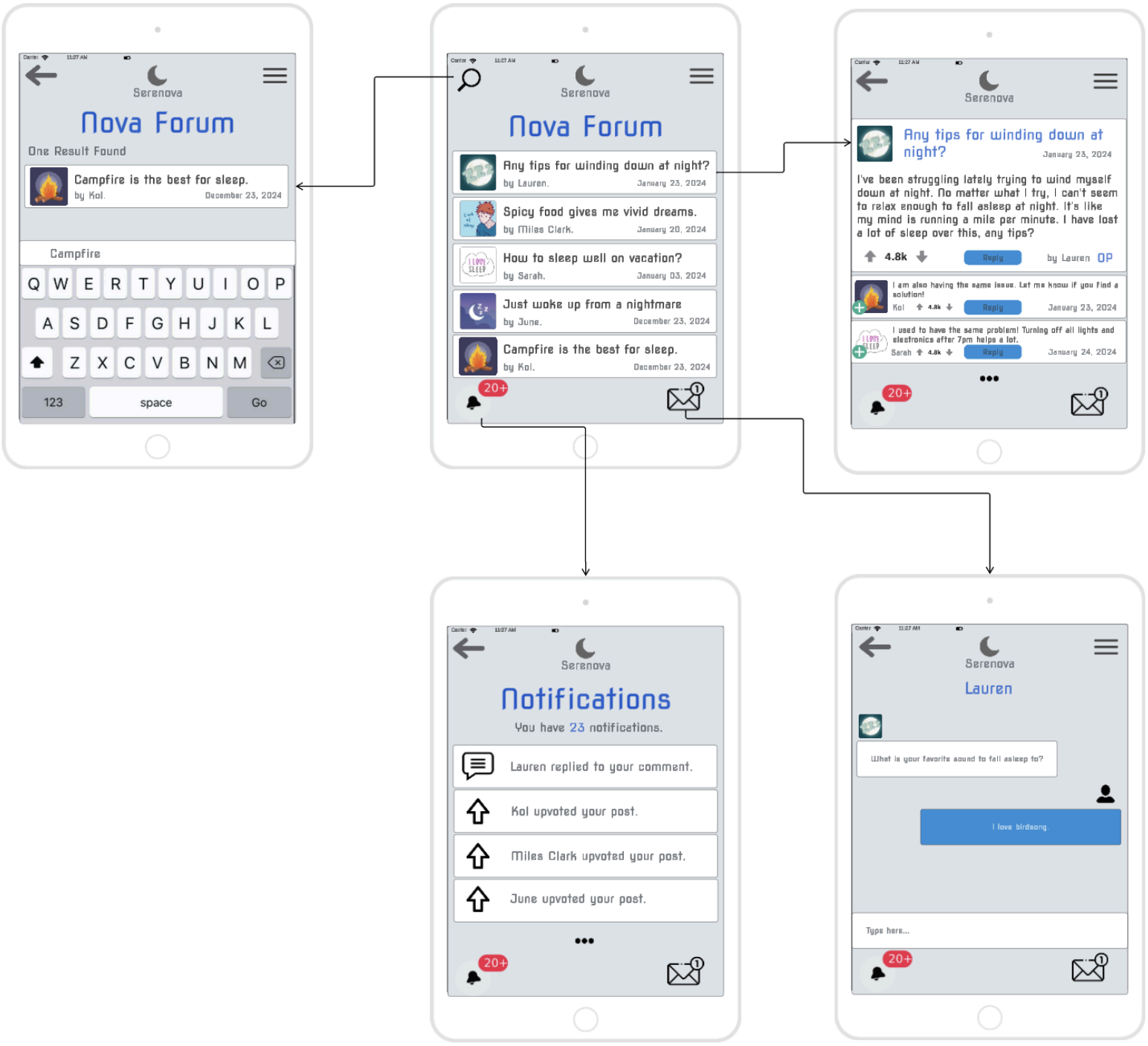


Profile View

Serenova users will have their own personal profiles where they can view user information and personalize their own settings. The sleep reward system will also be accessible through the user's profile page, a feature that will motivate users to be consistent with working to improve their sleep schedule.

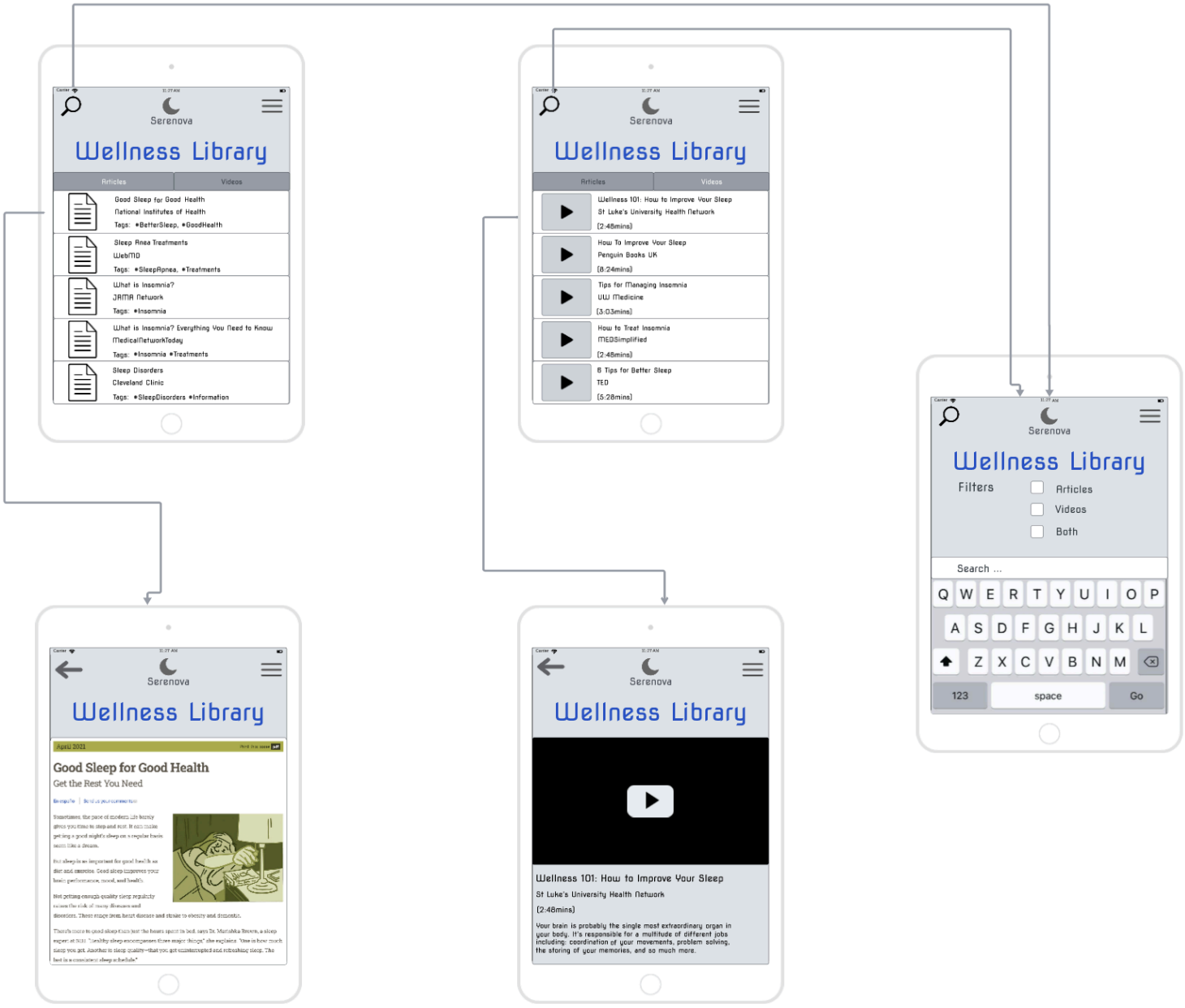
Community Forum View

Serenova will offer *Nova Forum*, a feature that will allow users to discuss sleep-related topics. Users will be able to post threads for others to see, reply to those threads, and even direct message other users.



Sleep Resources View

Serenova will give users access to various sleep resources through the app's *Wellness Library*. Users will be able to view relevant articles and videos that will assist in improving their sleep patterns.

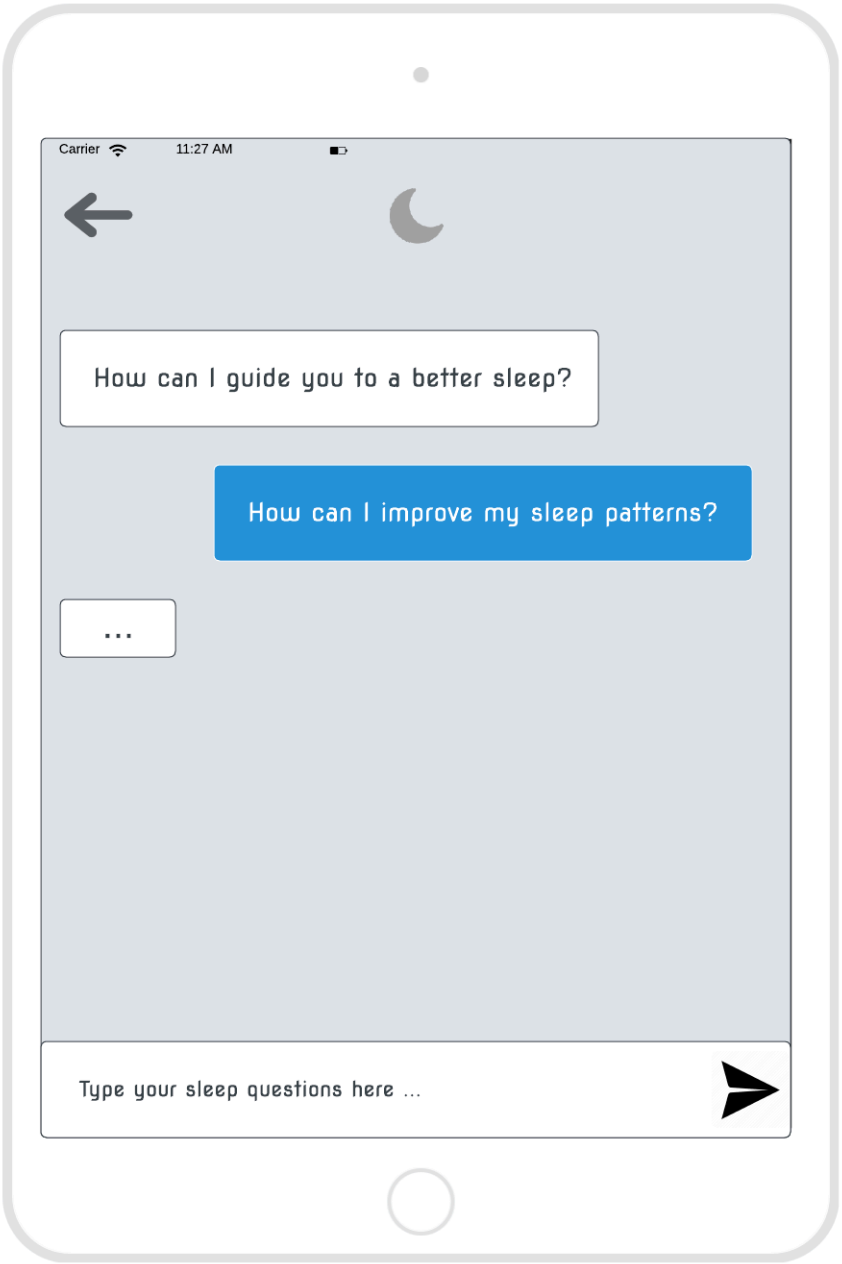


Sleep Mode

Serenova will have a *Sleep Mode* where users will be able to effortlessly transition to a calm state for nighttime. Users will have access to resources such as relaxing sounds and guided meditations. When the user wakes up, they will be greeted with a screen that gives them options to review and reflect on their previous sleep.

Chat Bot View

Serenova will have a page dedicated to hosting interactions between the user and an online AI Chat Bot for users to receive sleep guidance. Below is the chat interface for the app.



4.6 Database Design

Serenova will be utilizing the Firebase Realtime Database to store each user's sleep data and facilitate real-time data synchronization. To take advantage of Firebase's instantaneous data syncing abilities, we have decided to structure the database using **flattened data structures**. All of the classes will serve as their own collection, with entries to each document within the collection being a unique identifier. By avoiding nested data structures, we will be able to increase the efficiency of our search queries and reduce latency by only loading data from the database when we need it. Additionally, we will use Firebase Cloud Functions for data processing and triggering events in real-time for users, such as alerting users of a received message from other Serenova app users.