CURE

Project Team:

Charles “Steve” Stamey

Joshua David Brown

Brian Goughner

Chauncey Davidson

Nakava “Guylain” Kibunzi

**Table of Contents**

**1. Project Definition…………………………………………………………………….6**

* Background(Why)………………………………………………………..6
* What is CURE...………………………………………………………….6
* How………………………………………………………………………..6

**2. Project Requirements……………………………………………………………….8**

* Functionality……………………………………………………………....8
* Usability…………………………………………………………………...8
* System…………………………………………………………………….8
* Security…………………………………………………………………....9

**3. Project Specification………………………………………………………………...10**

* Focus / Domain / Area…………………………………………………....10
* Libraries / Frameworks / Development Environment………………….10

**4. System – Design Perspective**……………………………………………………….11

* Identify subsystems – design point of view
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices (Optional)
* Sub-System Communication (Diagram and Description)
  + Controls
  + I/O
  + DataFlow
* Entity Relationship Model (E-R Model)
  + Example - <https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model>
* Overall operation - System Model
  + Simplified Sub-system to System interaction

**5. System – Analysis Perspective** – *Group responsibility*

* Identify subsystems – analysis point of view
* System (Tables and Description)
  + Data analysis
    - Data dictionary (Table - Name, Data Type, Description)
  + Process models
* Algorithm Analysis
  + Big - O analysis of overall System and Sub-Systems

---------------------------------------------------------------------------------------------------------------------

04/11

------------------------------------------------------------------------------------------------------------------

**6. Project Scrum Report -** *Group Responsibility*

* Product Backlog (Table / Diagram)
* Sprint Backlog (Table / Diagram)
* Burndown Chart

**7. Subsystems**

**7.1 Subsystem 1** – Name 1 - *Individual responsibility*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

**7.2 Subsystem 2** – Name 2 - *Individual responsibility*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

**7.3 Subsystem 3** – Name 3 - *Individual responsibility*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

**7.4 Subsystem 4** – Name 4 - *Individual responsibility*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

-----------------------------------------------------------------------------------------------------------------

**8. Complete System** – *Group responsibility*

* Final software/hardware product
* Source code and user manual – screenshots as needed - Technical report
  + Github Link
* Evaluation by client and instructor
* Team Member Descriptions

**1. Project Definition**

**Background**

According to the NC Disease Event Tracking and Epidemiologic Collection Tool (NC DETECT), emergency department (ED) visits for opioid drug overdoses in Guilford County jumped from 47 in 2010 to 392 in 2017. While heroin overdose ED visits increased from 9 in 2010 to 291 in 2017.. GCES calls relating to opioid overdoses increased from 157 in 2013 to 1,015 in 2017, a year in which GCES conducted nearly 700 Naloxone overdose reversals. Of the 90 accidental poisoning deaths in Guilford County involving ICD-10 codes X42-X44, 43.3% involved heroin, 36.7% synthetic opioids and 21.1% other opioid drugs. GCES has reported 175 opioid-related deaths in 2017 based on local law enforcement preliminary death data.

This indicates a need for easily accessible resources and tools for substance addicts to help encourage them and help them to get the help they need. When people don’t know they have help, they usually don’t seek it. This app can help inform and open families and addicts themselves to the resources and help them to plan for outcomes and notify them of events in their area. It can also provide people with incentives to stay on track with their program(s) and goals.

**What is the CURE Aid Tool?**

The CURE application is a tool which users can utilize to help themselves or someone they know with harmful substance addictions. The application will allow users to plan events and meetings with helpful organizations and care centers, make and organize gradual recovery steps for each week, and get immediate help in case of emergencies. We want this to be a one-stop useful tool to aid addicts and help them stay on track with their goals.

**Goals for the CURE Aid Project**

We will be fleshing out a whole new design template for the CURE application. We will be working closely with members of the original Android team to redesign the app from the ground up to be more interactive. Some of the planned features include:

* Location services for finding centers and resources near you.
* Auto-dial for getting help in an emergency.
* Calendar Appointments.
* Goal tracking with notification updates.
* Text Help
* Event updates for community outreaching.
* Achievements for meeting goals

We’re currently aiming to make this application uniform between both Android and iOS platforms. We would like to ensure it passes all privacy and store regulations so that it’s available to all who need it.

**2. Project Requirements**

**Functionality**

The CURE Application will be designed for ease of use with specific user functions for finding, setting up, and managing appointments. Users will be able to quickly build calendars and set up plans and goals.

Functional requirements include:

* Calendar API via Meteor
* User Interface with sliding tabs
* Event System for modularity
* Map API (likely from google)
* Authentication protocols for login
* CSV files and/or Database for mapping nodes and user accounts
* Event management/scheduling
* Hot Dial to emergency services
* Calendar viewing/editing
* Progress Report/Log of what the user has done
* Connection to the website for more information
* Anonymity
* Ability to locate nearby events on a map
* Ability to view information about events close by
* Notification setup for scheduled events

**Usability**

The user interface will consist of a primary menu with the important options available up front. Each major option can be swiped through like a tab. Within these tabs are sub-level features which relate to the respective main option for which they are under.

As we do not have a runnable instance of the application there is no testing data sets yet. In future will include use cases.

The performance will be smooth transitions between tabs. By using APIs for location search and map usage, related functionality should be very efficient. With regards to calendar events, they will likely be light-weight background notifications which keep track of dates and times for which the events need to notify the user.

**System Details**

The app will be run primarily on Android and iOS devices. The goal is to be compatible on Android 4.4 and above. This lower end compatibility is important in order to include as many users as possible. The goal is for this app to run under 32MB of RAM while in direct use, while only using about 12MB of RAM for background processes.

There will be necessary APIs for integrating map functionality (ArcGIS), and the user may need a web browser for accessing information from website links (one button call or text feature. if we decide to provide them. Otherwise the application is meant to be an all-in-one aiding tool for users.

A database may be necessary if only for login information, however it is likely that this information can be stored via Google’s account information tools.

**Security**

The login pages will be fitted with Google’s authentication protocols with regards to login procedures, since it uses tokens from Google’s API. For when users can choose to share their information with known contacts, information will be encrypted before being sent to them. Information can be authenticated for integrity using a SHA-256 hashing on data.

Privacy concerns are also important as many users find confidentiality of health information as a standard. It’s a standard we will uphold through encryption or hashing of any data that needs to be sent to a database. Any records will not be visible by us or outside parties. And with the Text Help module any information sent will only be visible by the two chatting parties.

**3. Project Specification**

**Concentration and Target**

The primary focus of this application is to create a web application that can be ported to mobile devices for Cure-NC Greensboro Area Health Education Center (GAHEC). The target audience for our application are families of the opioid crisis and especially individuals suffering from it. CURE Aid Tool will be an application which is part of the self-help and organization genre.

**Libraries, Frameworks and Development Environment**

For the web application we will be utilizing Meteor API to develop the application in JavaScript, Mongo DB (noSQL database), Node.js, Adobe PhoneGap, and Cordova. Google Maps API or ArcGIS Maps for node integration for interactive maps. The development environment for the project includes both Android and iOS platforms where we will develop the original configurations in JavaScript.

Javascript will be used to code the backend functionality and database connections. Database objects from the Mongo database will be passed around the system as needed via an event-based system. Cordova can help us with multiplatform support as it has capabilities to extrapolate a single codebase into both Android and iOS applications for integration of both iOS and Android.

**4. System – Design Perspective**

**Identifying the Subsystems**

The CURE Web Application is a self management helper and tool. It’s main functionality is built for the individual users who need scheduling, event management, and privacy for the logging and notifications. This is accomplished through various subsystems which are designed to be modular for future additions to and removals from the application. These subsystems aggregate the many necessary components through an overarching event-driven system controlled through a Controller class. This is a brief overview of each subsystem. Later in this section the subsystems are broken down further with relevant inputs and outputs. The main subsystems for CURE are as follows:

1. **Calendar**

The Calendar subsystem is designed to store events, meetings, and other program related goals into a typical calendar format. The need for such a system

arrives from potential privacy concerns and for the integration of these events

into other key subsystems, such as the Goal Tracker. Below in *Figure 4.1*, are examples of interactions of the made by the Calendar to the database. *Figure 4.9* expresses a system-level diagram for how the Calendar interacts with other components.

1. **Map/Location**

The Mapping subsystem is melded together with a location service which

can make database queries for nodes which will populate the map. The mapping technology utilizes Google maps, which is easily integrated into Meteor’s API. From the map, the nodes can be selected. Nodes contain important information about the location, pulled from the location service and database.

1. **Hot Dial**

The Hot Dial subsystem is a relatively simple system which can automate making of calls to important places and entities. Some initial numbers are saved into the system, but more can be added/removed at the user’s discretion. The Hot Dial subsystem is self-contained and doesn’t need to call from other services for information, other than built-in Android/iOS functionality for phone calls.

1. **Goal Tracking / Logging**

The Goal Tracker subsystem handles storing of logs and user-entered

data. This subsystem integrates with other subsystems to store historical information about previous visits to care centers, events and meetings. Also users can insert private or public information and set it as an event to be stored in the log later, once the goal is completed. *Figure* 4.1 further lays out some movement through the system from the Goal Tracker and Logging functionality.

1. **Settings**

The Settings subsystem is designed around account management and

preferences. Like many other applications, settings can be altered to the user’s likings. Some examples include: editing security settings, adding/removing profiles, and blocking/enabling notifications. The Settings subsystem has strong interaction with other subsystems, as it can disable/enable features which other systems use.

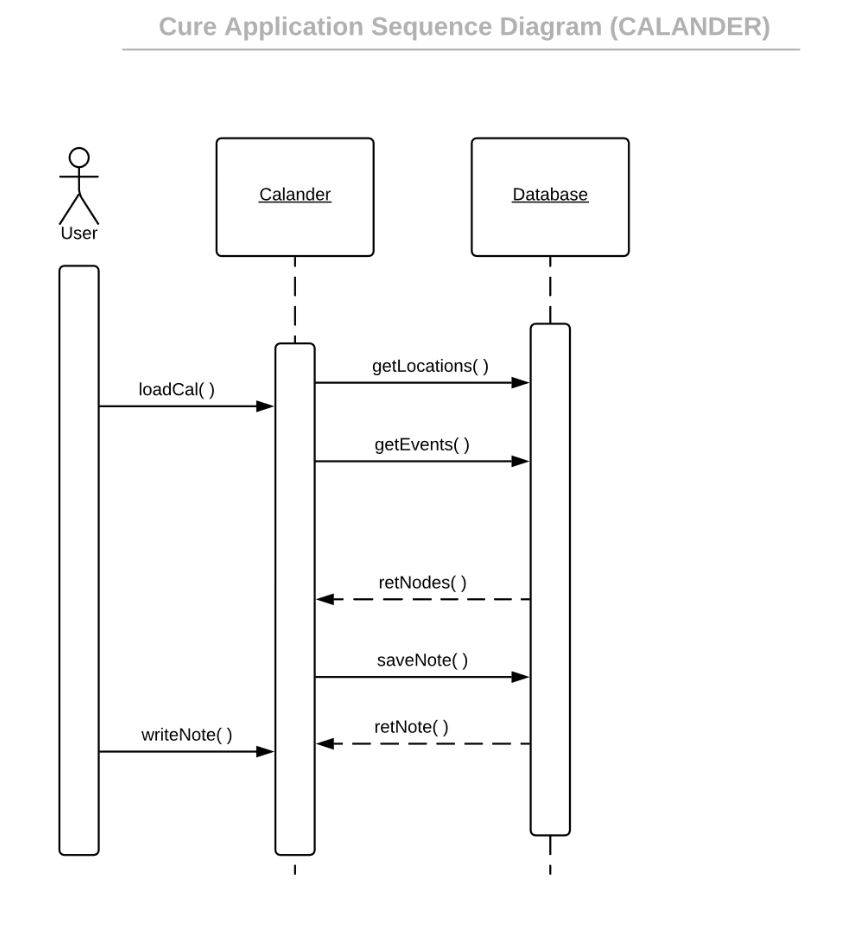
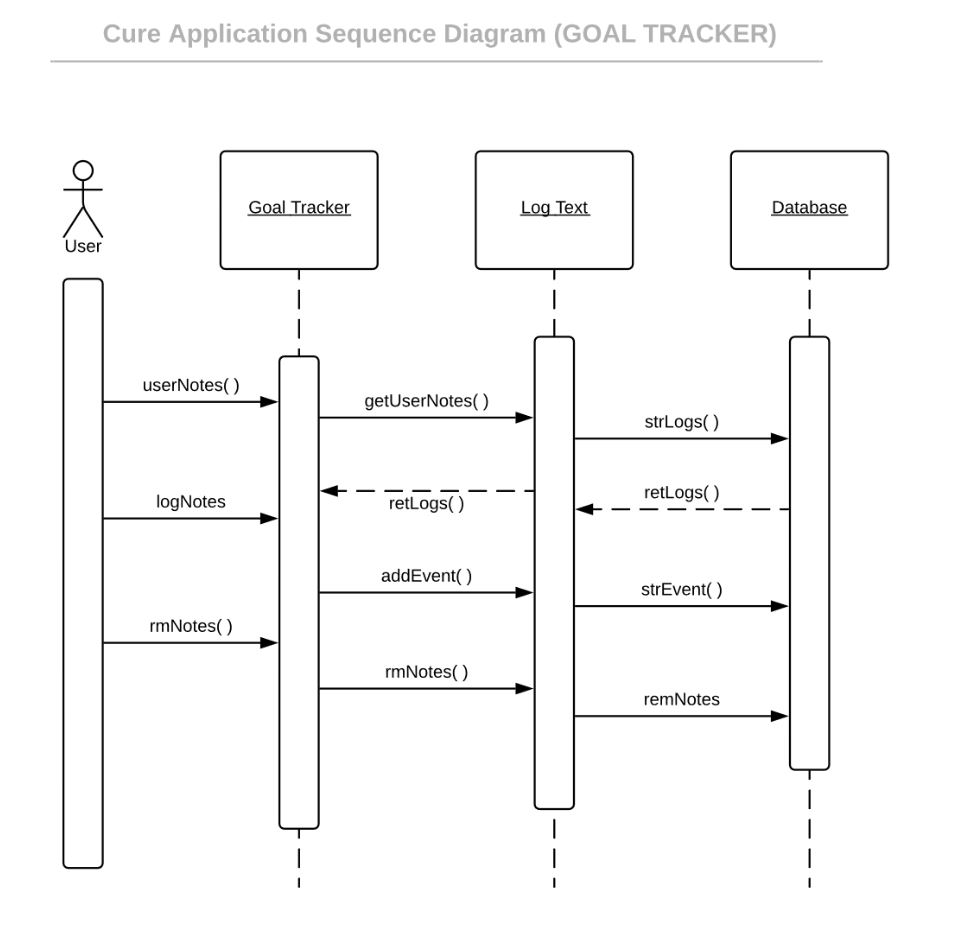
****

Figure 4.1 Figure 4.2

**(Brian Goughnour)**

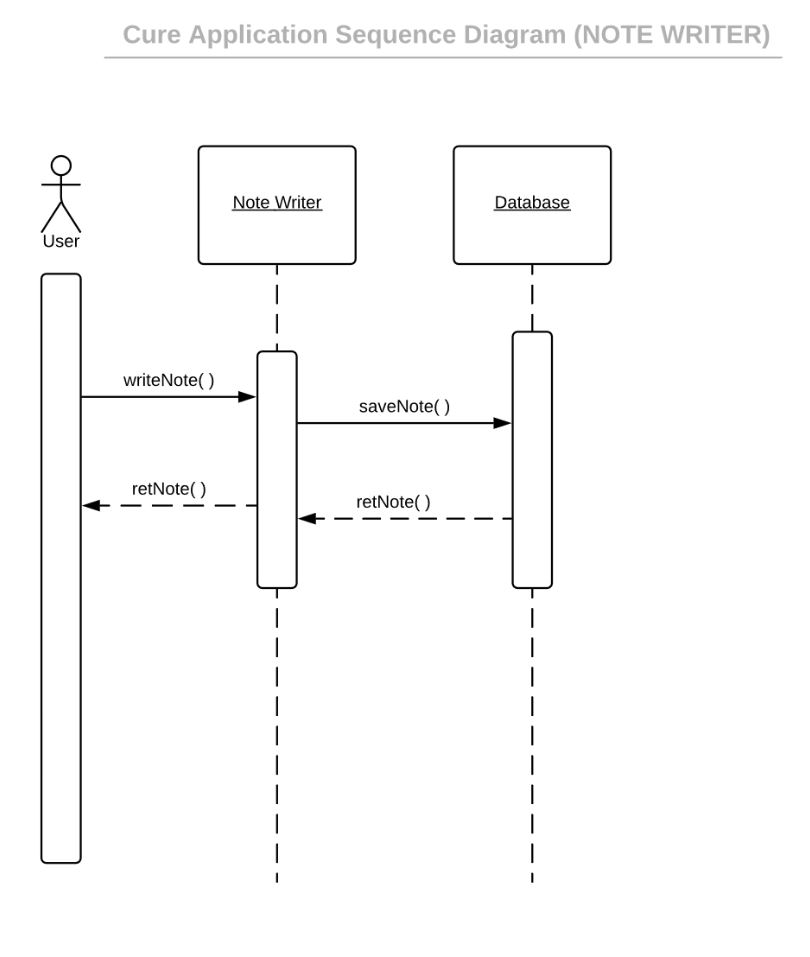
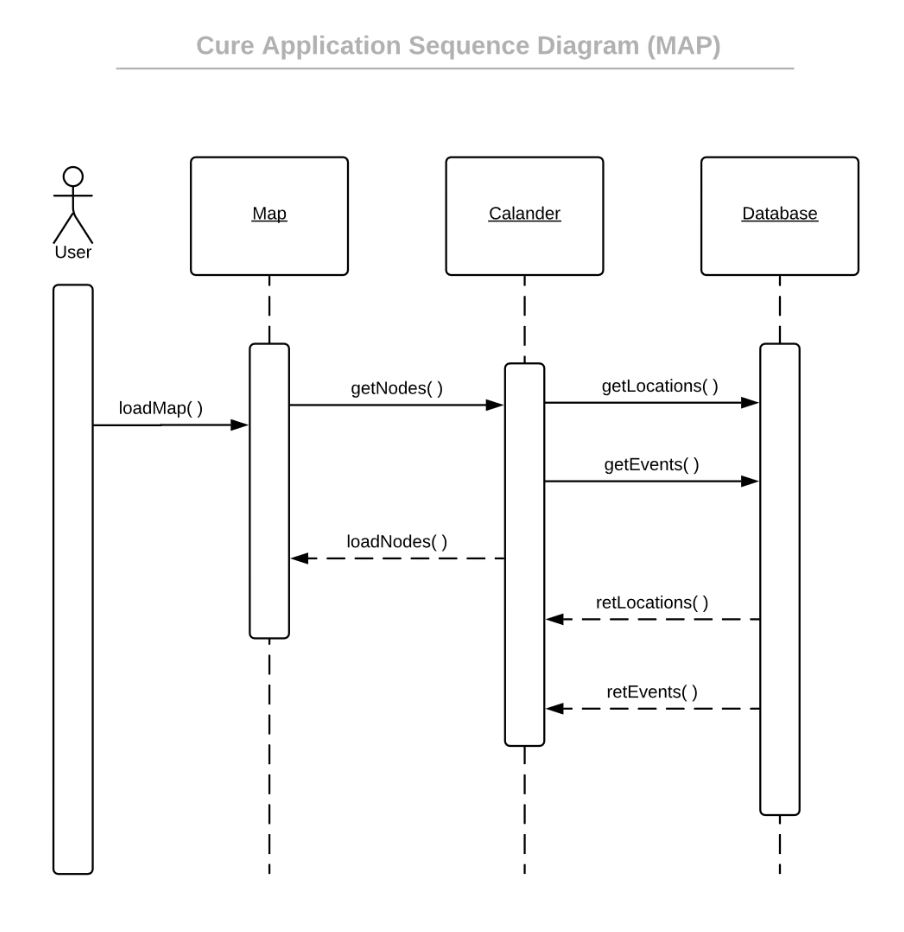
****

Figure 4.3 Figure 4.4

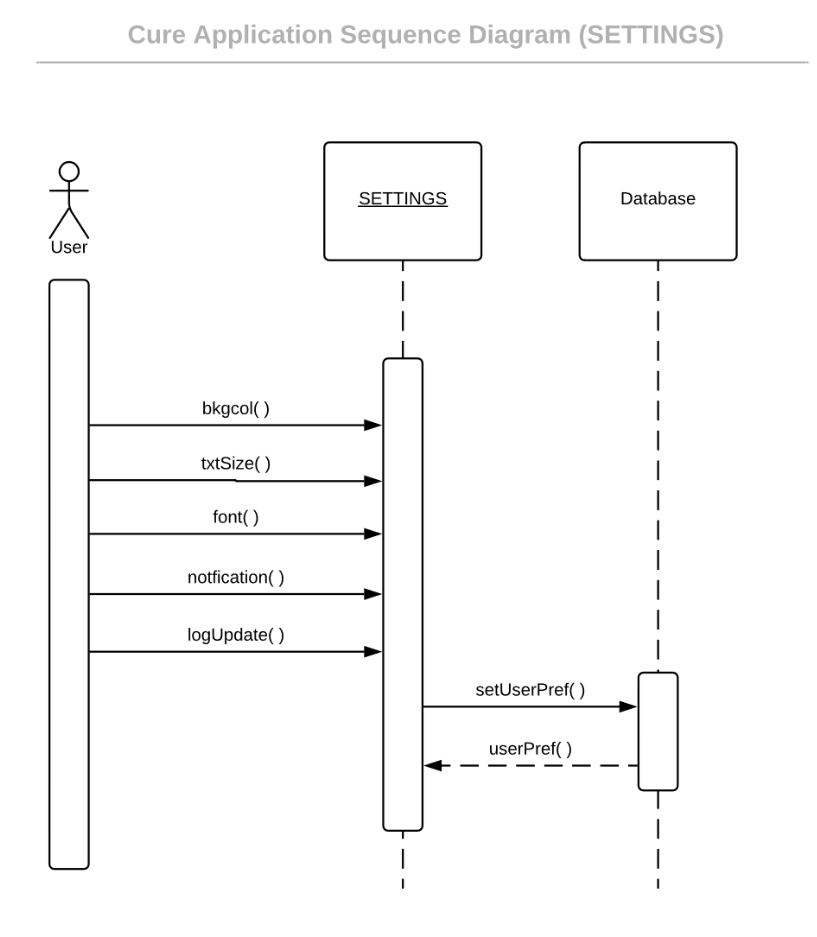
****

Figure 4.5

**(Brian Goughnour)**

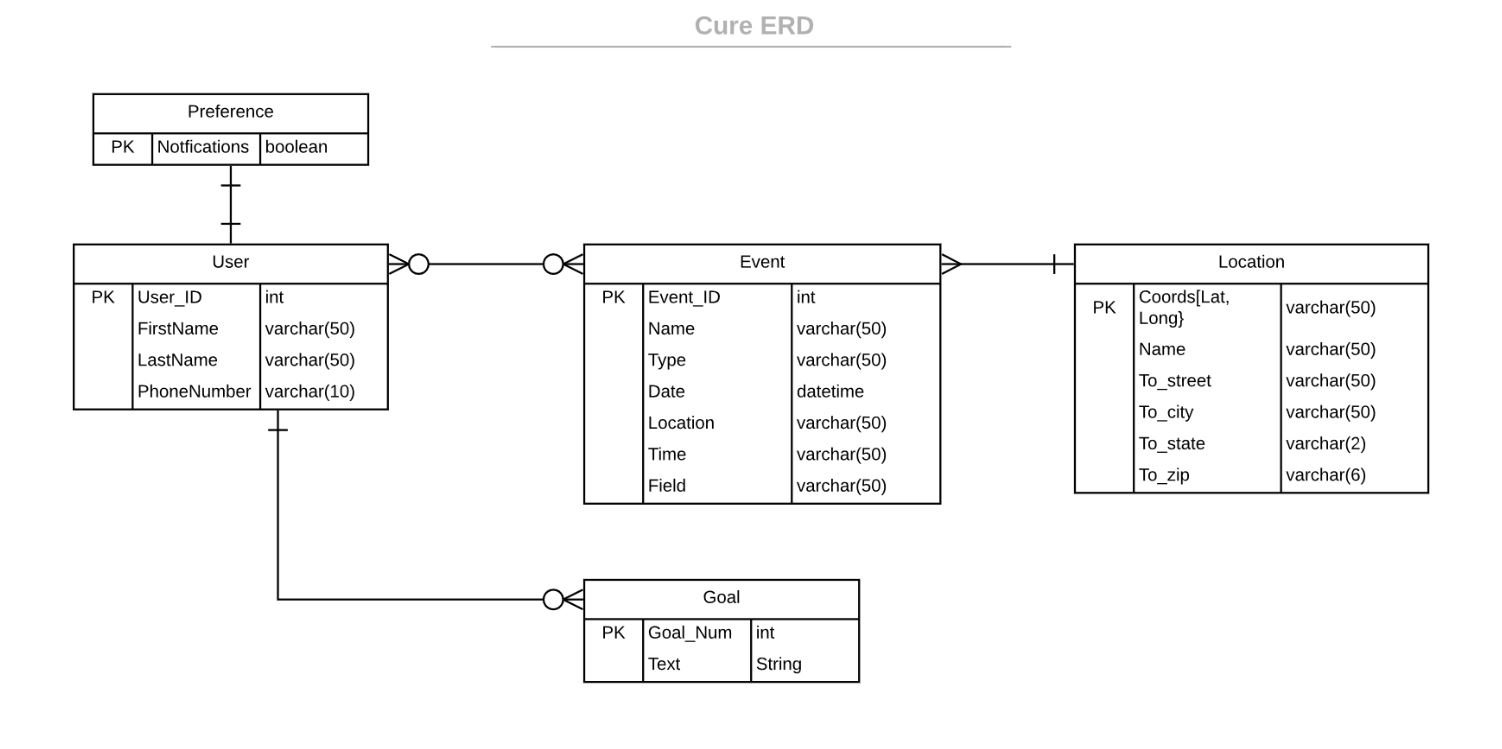
****

Figure 4.6

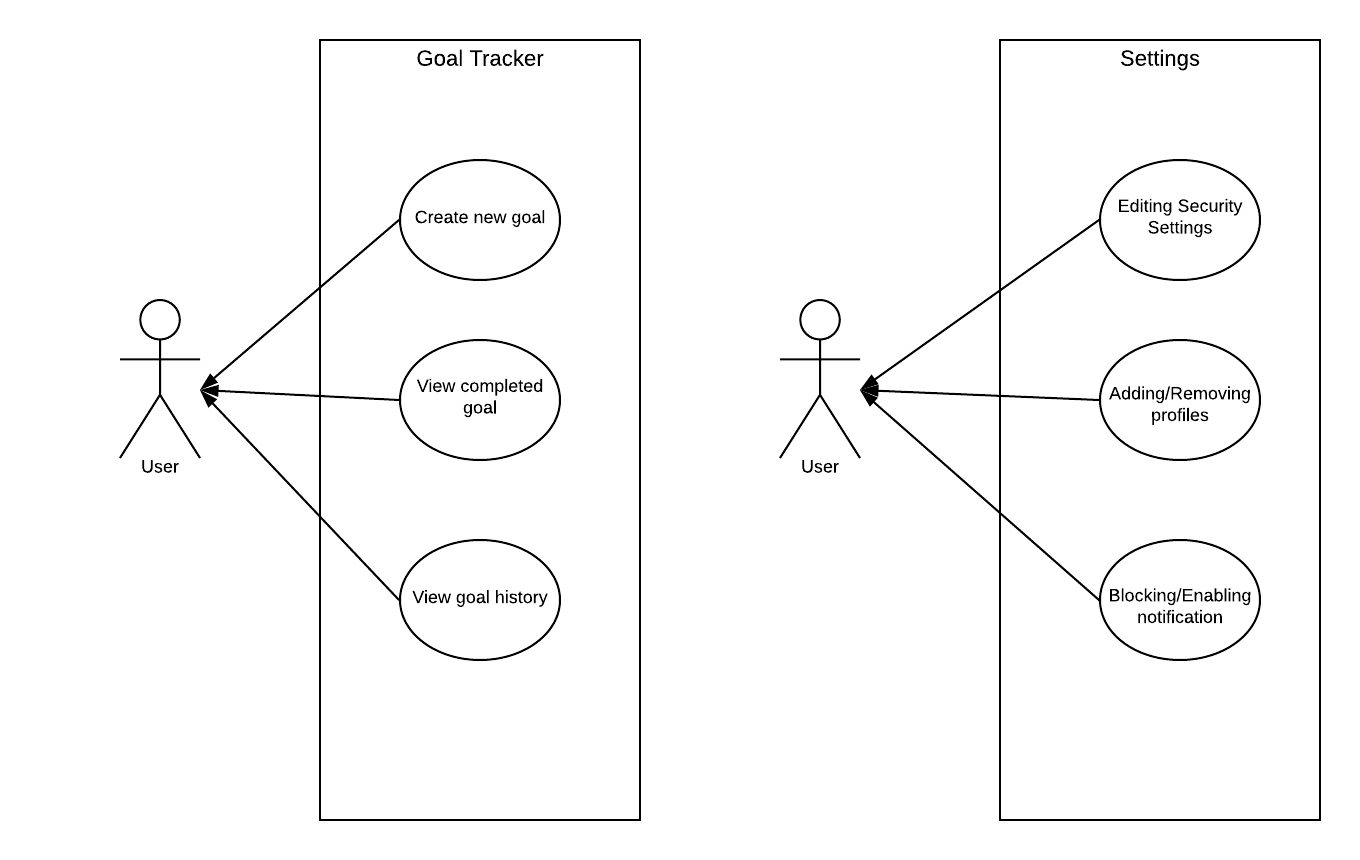
****

Figure 4.7

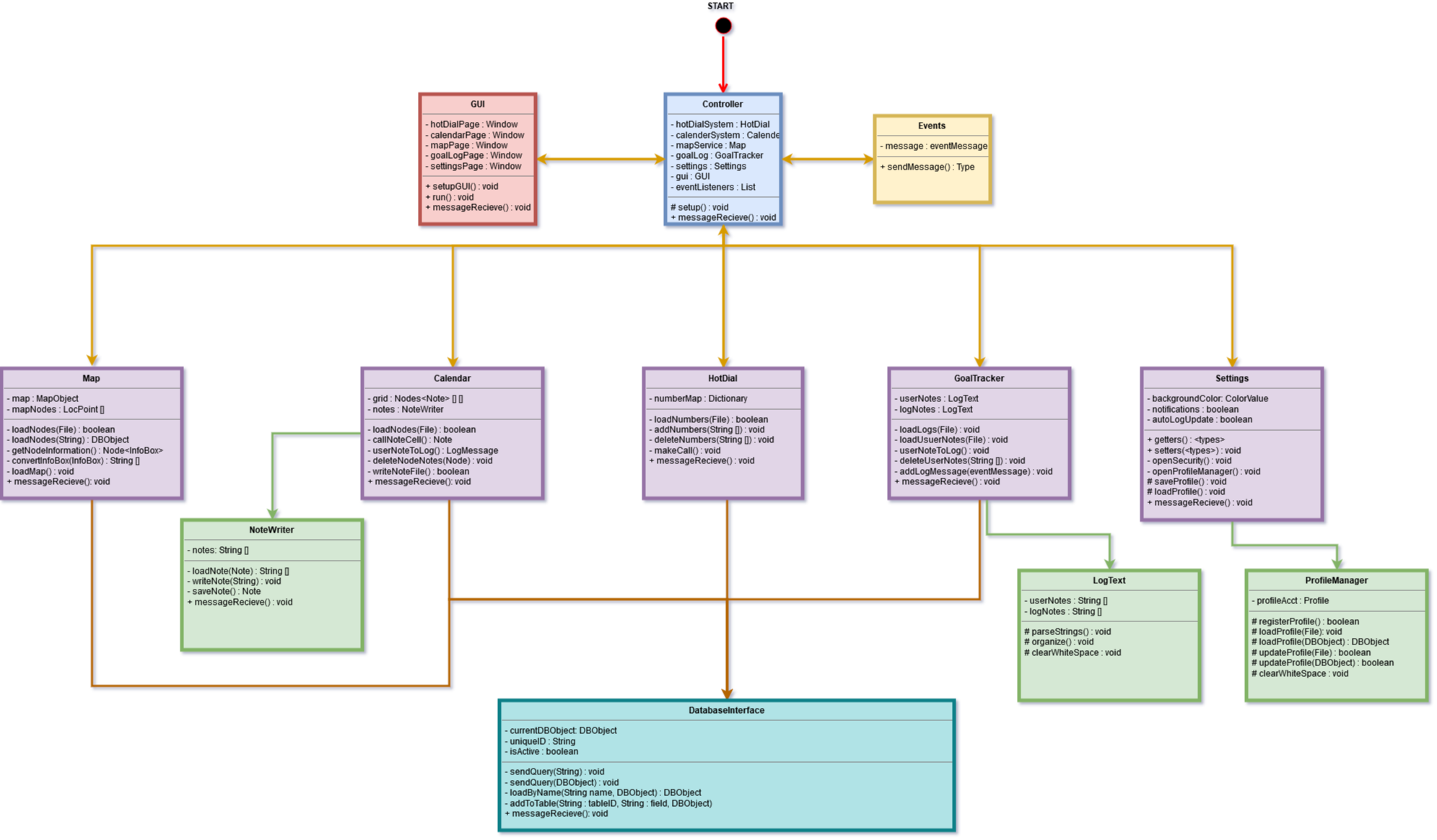
****

Figure 4.8

**Calendar Subsystem**

* Controls
  + If a user touches a cell they can enter a new event.
  + If a user long touches a cell they can delete from a cell.
* I/O
  + A cell of the calendar can take in a data structure which holds key information in a header, such as Date, Time, Message.
    - The result of this is a modified cell structure which contains the newly entered information.
  + The user accepts a notification prompt to insert this data into their Chronicles/Log
    - Data is sent out to other subsystems via an event message.
* DataFlow
  + The message data from an entry can be moved from the Calendar to the GoalTracking module to store in their Chronicles.
  + The Database API receives a query to update the personCount of an event in the Events table whenever the user has added a meeting/event to their calendar that is publicly recognized on the Map/Locations service.

**Map/Location Subsystem (connected)**

* Controls
  + If the user touches a node on the map information can be viewed.
  + If the user touches the “add event” button, then it can be added to their personal calendar and log.
  + Granted an internet connection is accessible and the database can be reached, data points and nodes can populate the map.
* I/O
  + Location services render points on the map through data from the database.
    - The result of this is blips/nodes on the map correlating to the appropriate position of an entity on the map.
  + The user accepts a notification prompt to insert node data into their Chronicles/Log and/or Calendar
    - Data is sent out to the other subsystems via an event message.
* DataFlow
  + The data sent to other subsystems includes the time the event was pinged, the time/date that the event begins, and a small description of the event. This is packed neatly into a generic data structure accessible to the Calendar and Goal Tracker subsystems.

**Hot Dial Subsystem**

* Controls
  + If the user touches a contact, the phone is triggered to make a call.
  + If the user touches the plus icon and gives an appropriately formatted phone number, a new contact can be stored.
* I/O
  + User’s input from the keypad/digital keyboard
    - Results in a new contact generated with the name and/or phone number of a contact.
* DataFlow
  + All data for this subsystem remains within the subsystem except for a call out to the Android/iOS system to make a phone call.

**Goal Tracking / Logging Subsystem**

**Settings Subsystem**

* Controls
  + If a user profile is accepted an account can be created.
  + If a user profile is logged into, then all settings saved are only for that profile.
  + If notifications are turned on, then all subsystems can send notifications to the user.
  + If notifications are turned off, then subsystems cannot communicate with the user via pop-up notifications
  + If the user turns on auto-storing of log information, then updates from the calendar will be logged automatically within the Goal Tracker.
  + If the user turns on (on by default) security settings, then any messages are stored in an encrypted format.
* I/O
  + User’s input from the keypad/digital keyboard within the profile manager subclass
    - Results in a new profile being generated.
  + User turns on notifications
    - Results in all subsystems being able to display notifications.
    - The opposite is true when turned off.
  + User turns on Log auto-storage (on by default)
    - Results in most calendar events being automatically stored in the logging system whenever a task is completed or the event’s date has passed.
* DataFlow
  + Whenever the notification setting is turned on/off the Controller is sent an event message, which asks to set all subsystems’ notifications accordingly.
  + When log auto-storage is on, a message is passed out to related systems that logs may be stored within the Goal Tracker’s logging system.
  + Profile information that was entered is sent and stored as a hashed value within the database.
* Entity Relationship Model (E-R Model)
  + Example - <https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model>

**System Model - Interactions**

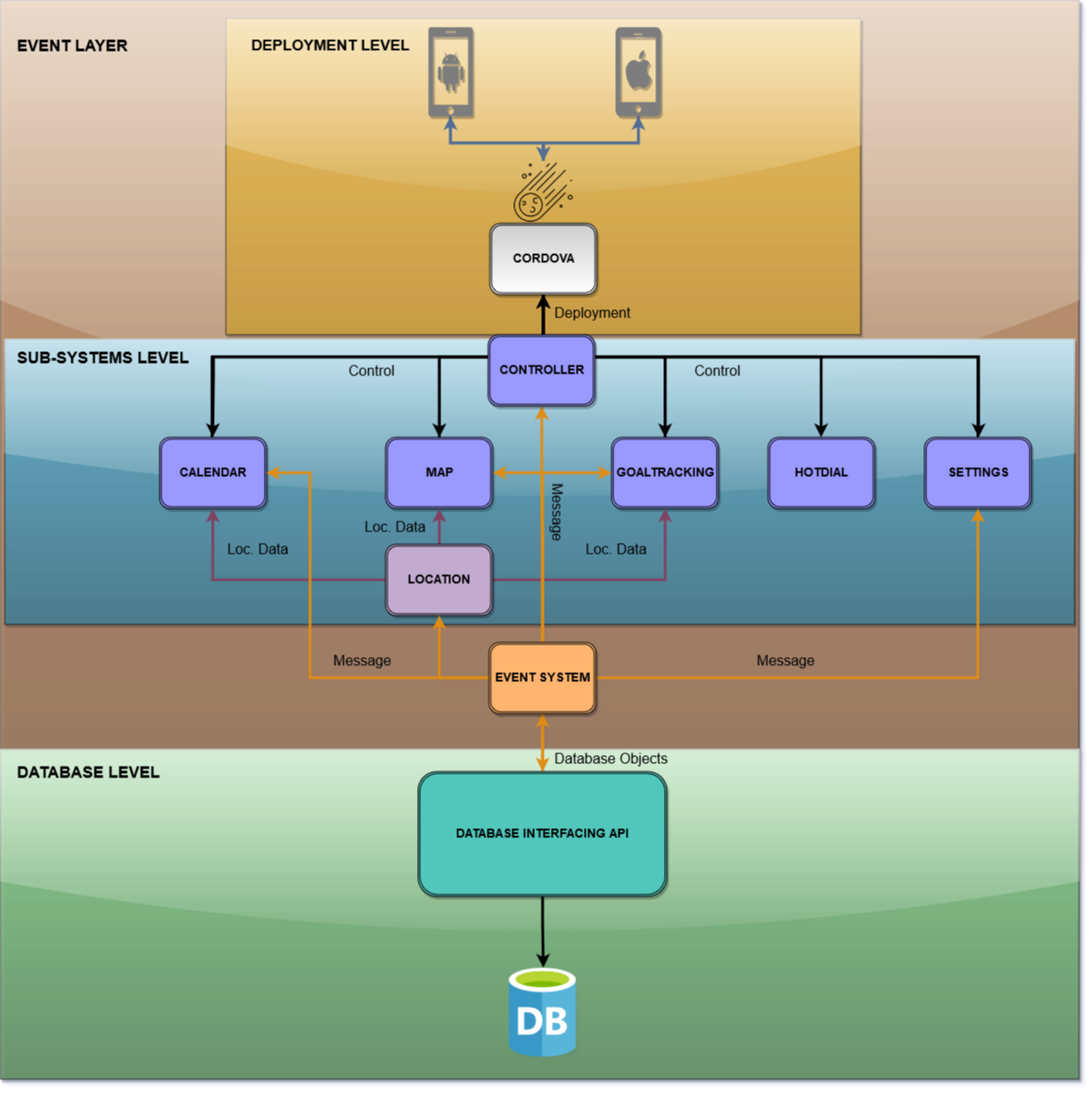


Figure 4.9

**(Joshua Brown)**

**5. System – Analysis Perspective** – *Group responsibility*

* Identify subsystems – analysis point of view (CJ)
  + Location Services (CJ)
    - Grabs information from Event System and displays on a map provided via Google Maps
    - Map that displays different locations that hold upcoming events (up to 3 months)
  + Calendar
  + Goal Tracker
* System (Tables and Description)
  + Data analysis (CJ)
    - Data dictionary (Table - Name, Data Type, Description)
      * LocationTable Events - Get Data
      * Table for scheduled events, count people going
      * Contained in Events
  + Process models (Josh, Brian, Nakava)
    - Using charts from
* Algorithm Analysis (CJ)
  + Big - O analysis of overall System and Sub-Systems - Make Chart
    - Database for event storage (Queries will be O(N))
    - Database for location queries, O(N)
    - Everything else internal, O(1)

|  |  |
| --- | --- |
| Algorithms/Operations | Time Complexity |
|  |  |
| Phone Operations | O(1) |
| Location/Event Query from Database | O(n) |
| Event Attendance Query | O(1) |

**Presentation for 2/12/14**

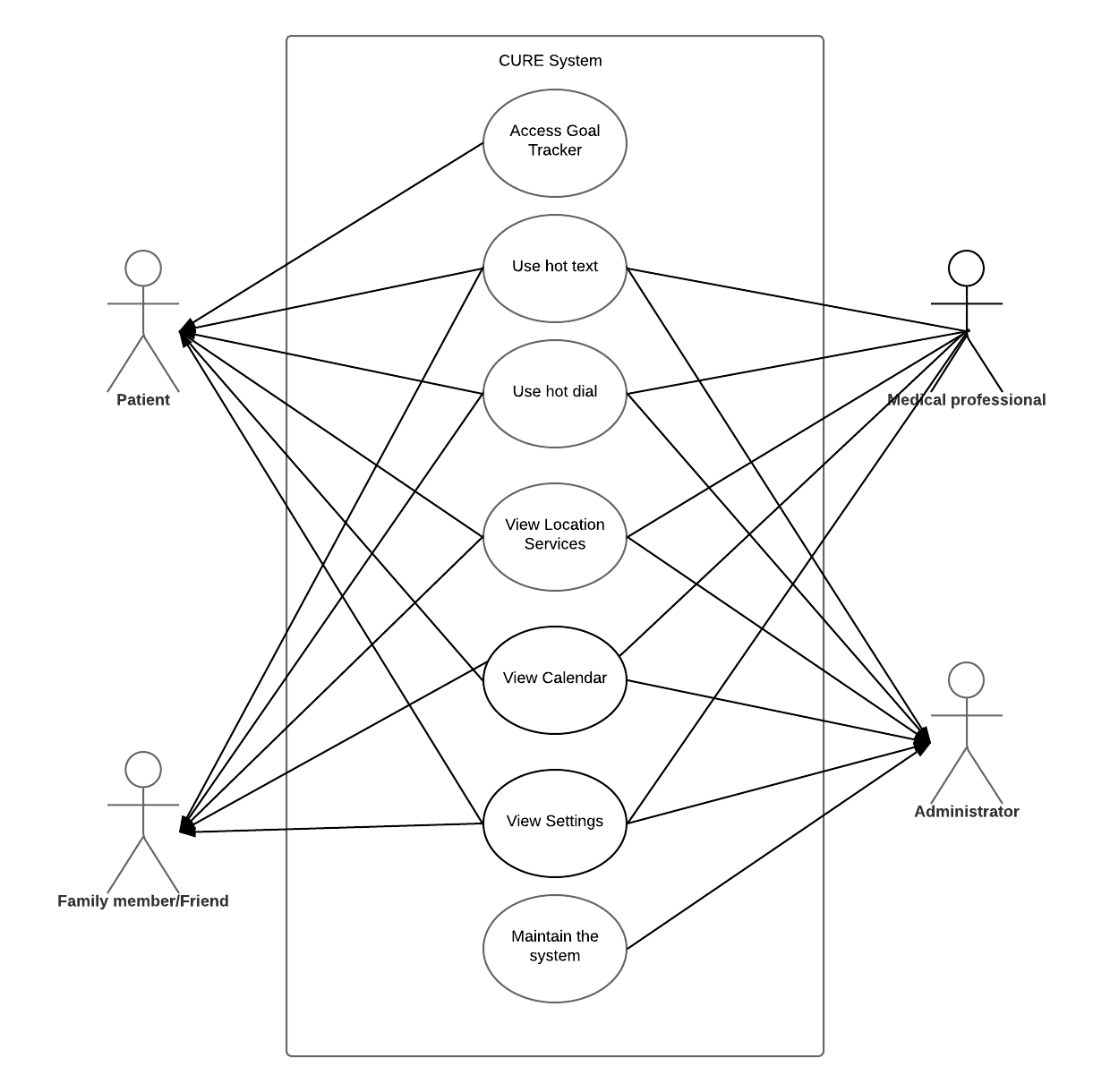


Figure 4.10

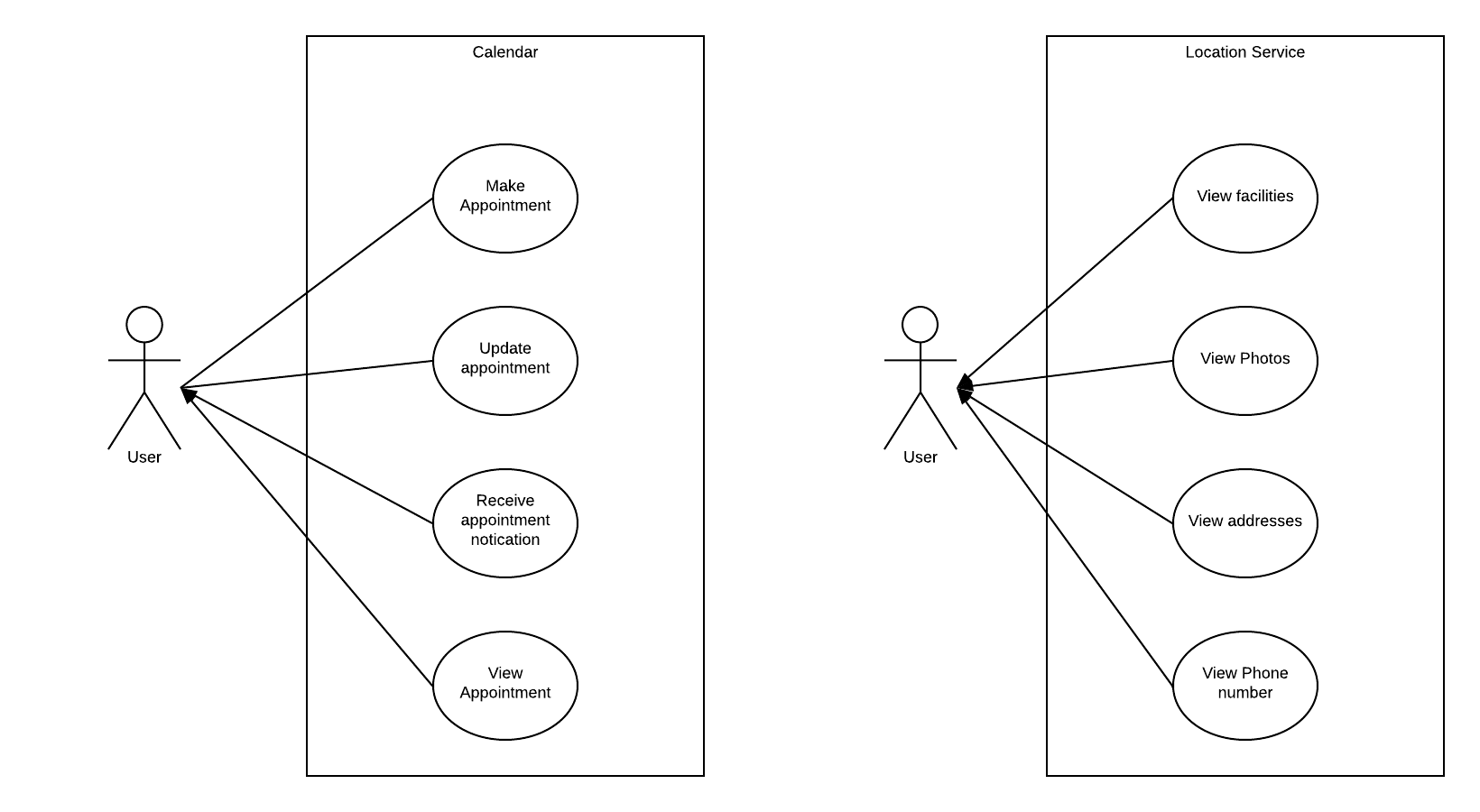


Figure 4.11

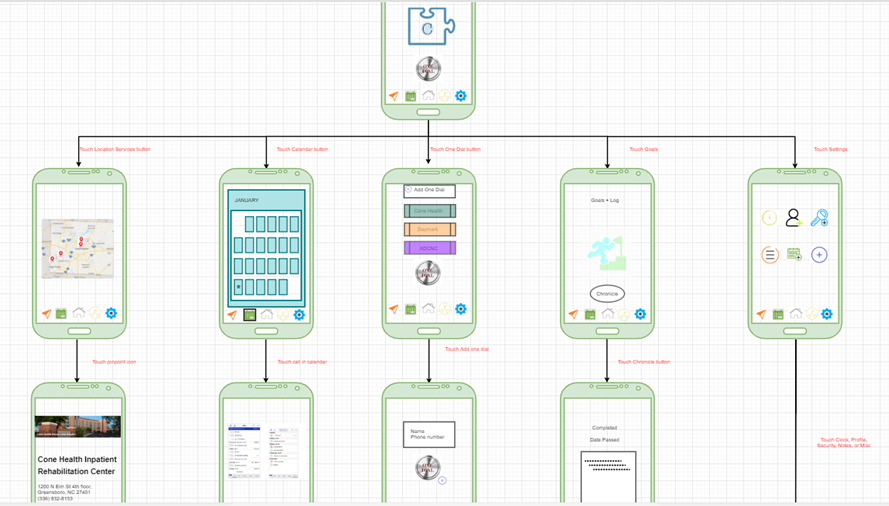


Figure 4.12