CURE

Project Team:
Charles Stamey
Joshua David Brown
Brian Goughner
Chauncey Davidson
Nakava “Guylain” Kibunzi

Table of Contents

    - 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 6 – Charles Stamey - *Setting Subsystem*

- 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

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 App?

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.
- 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

Software

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.

Hardware

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. 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.

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.

Platform

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.

Genre

CURE Aid Tool will be an application which is part of the self-help and organization genre.

4. System – Design Perspective

(Joshua Brown - intro and first 3 subsystems)

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.

2. **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.

3. **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.

4. **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.

(Steve Stamey)

5. **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.

(Nakava Kibunzi)

6. **Educations**

   Education and resources subsystem provide education and resources to the user in order to assist and support health care professionals. It facilitates community based clinical training experience for students and residents. It educates community members about health career choices and recruit future health care professionals.

(Brian Goughnour)

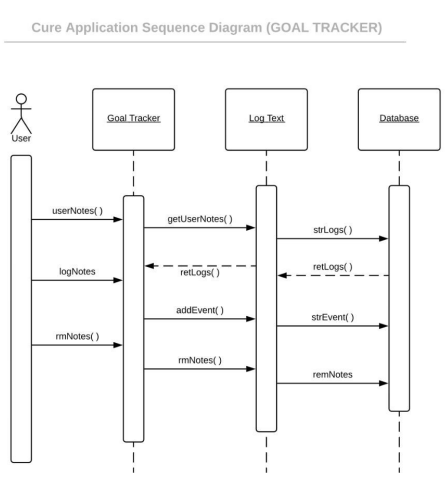


Figure 4.1

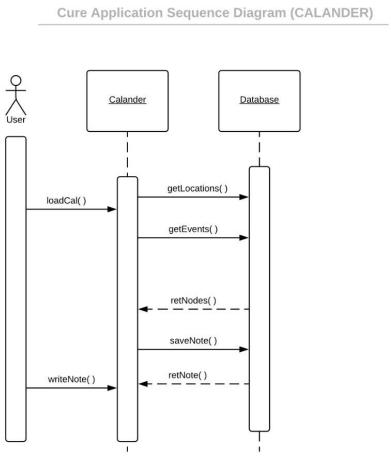


Figure 4.2

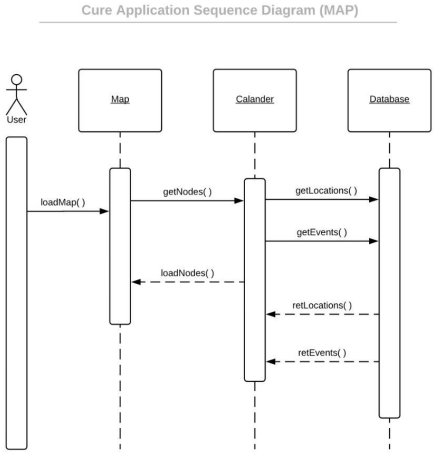


Figure 4.3

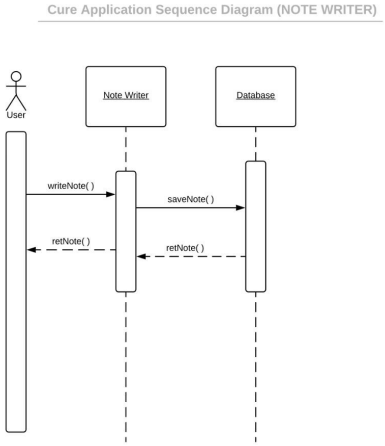


Figure 4.4

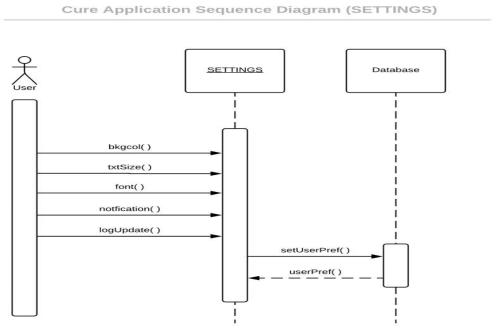


Figure 4.5

(Brian Goughnour)

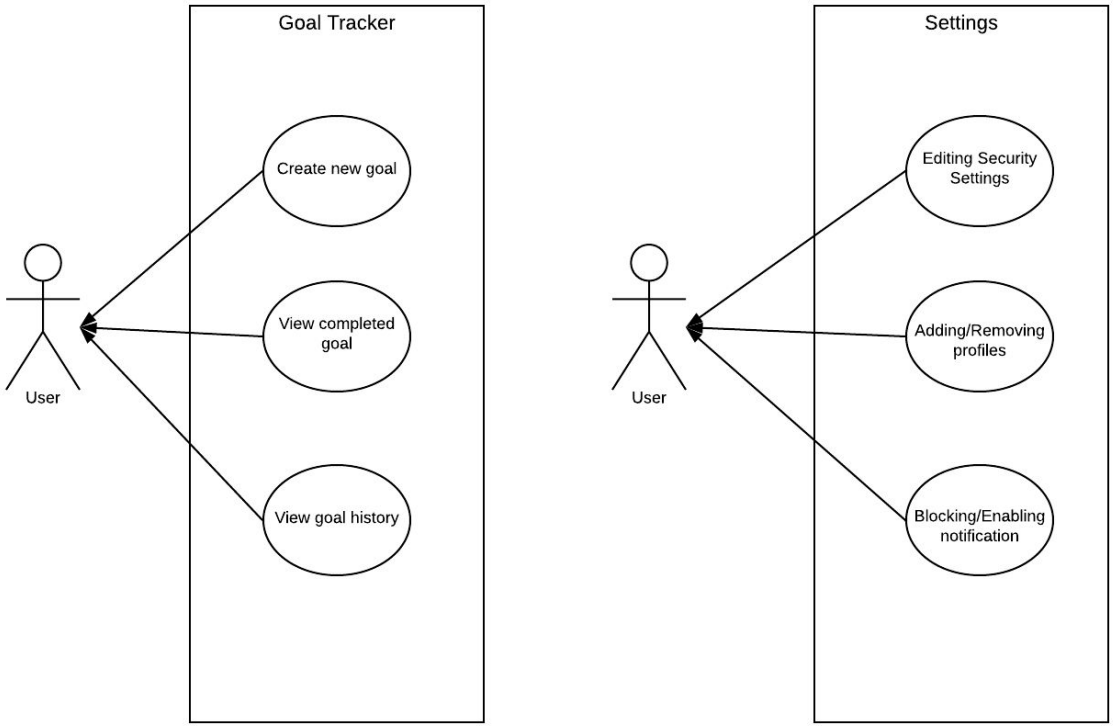


Figure 4.7
(Nakava Kibunzi)

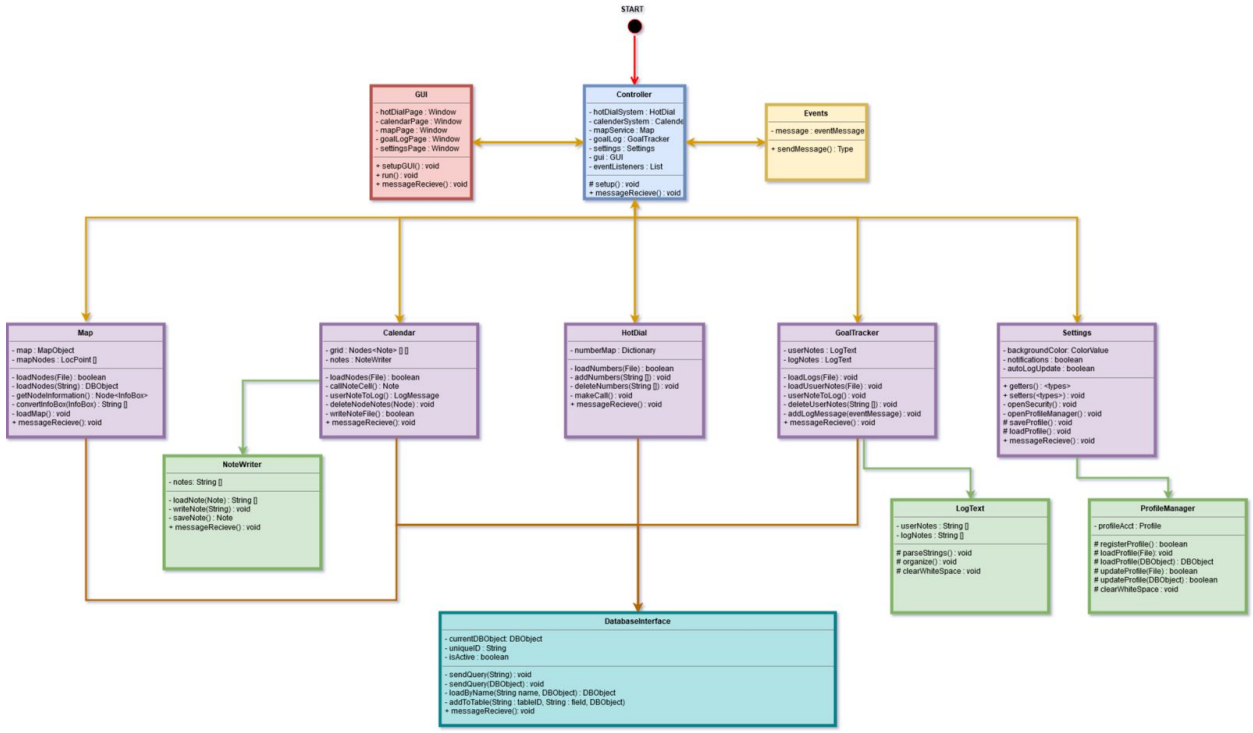


Figure 4.8

(Joshua Brown)

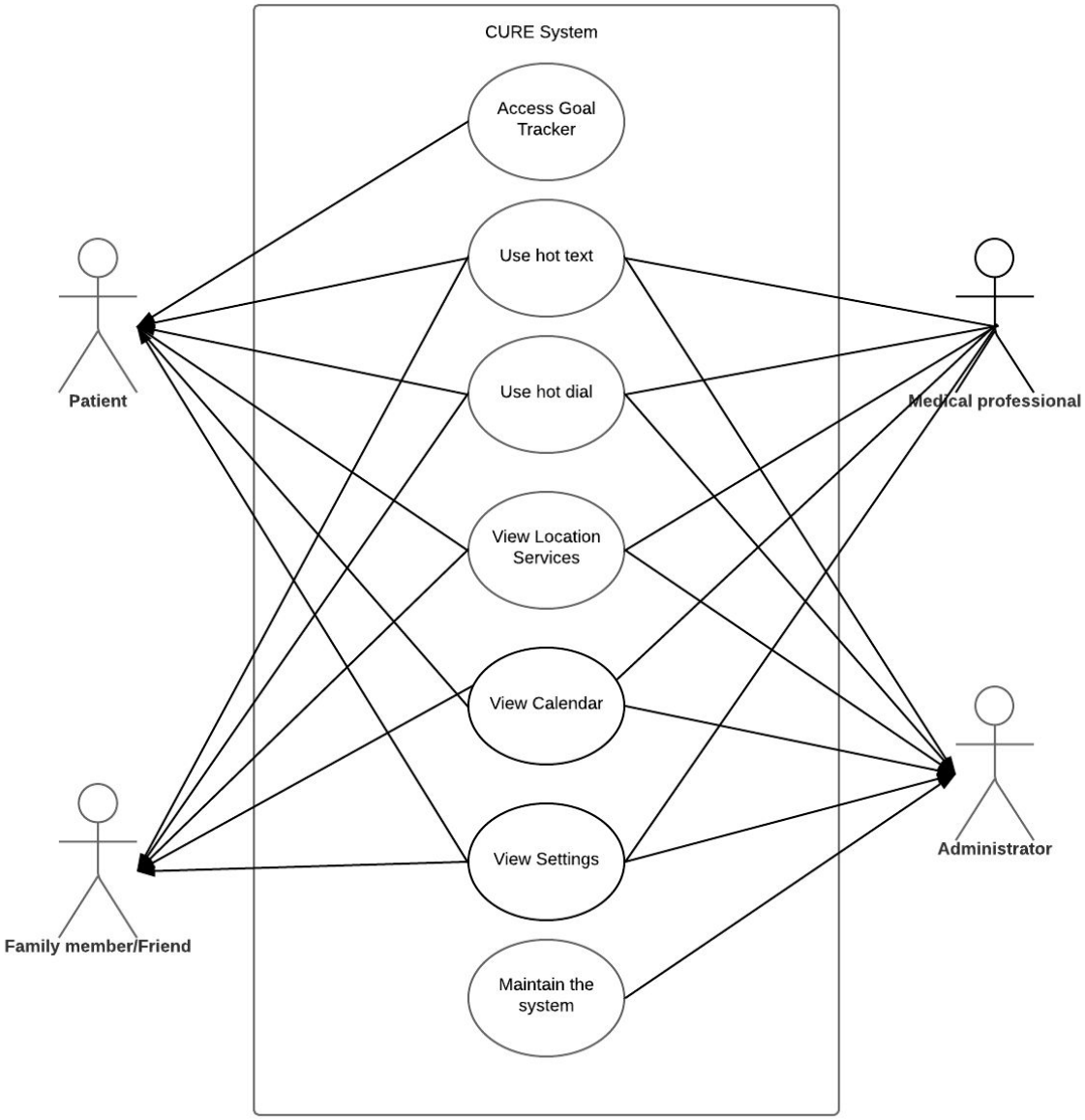


Figure 4.10
(Nakava Kibunzi)

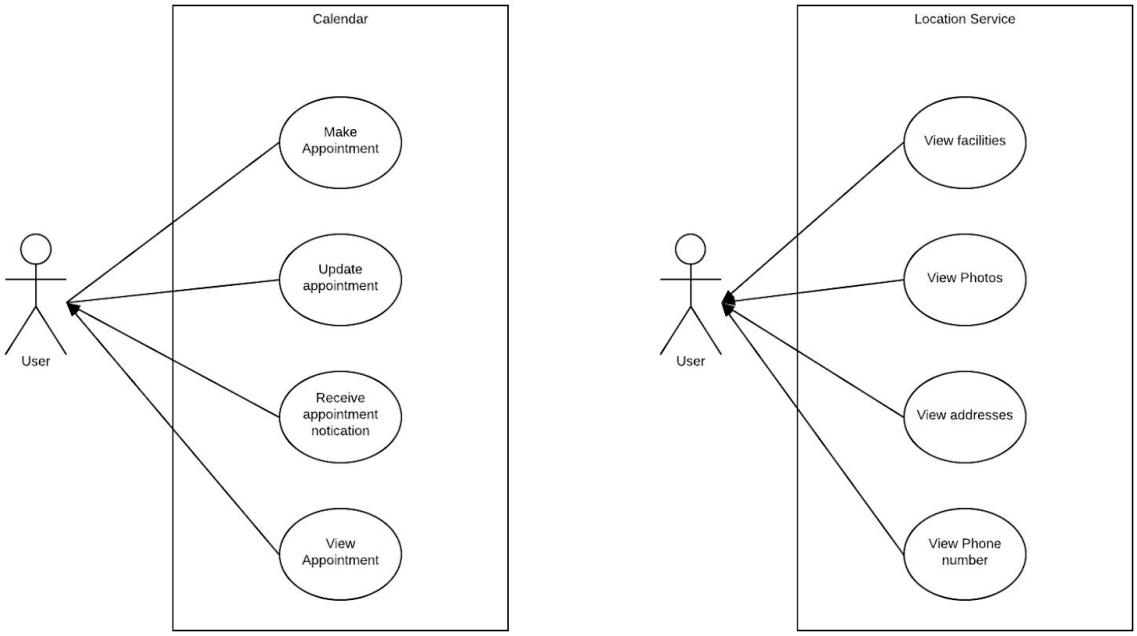


Figure 4.11
(Nakava Kibunzi)

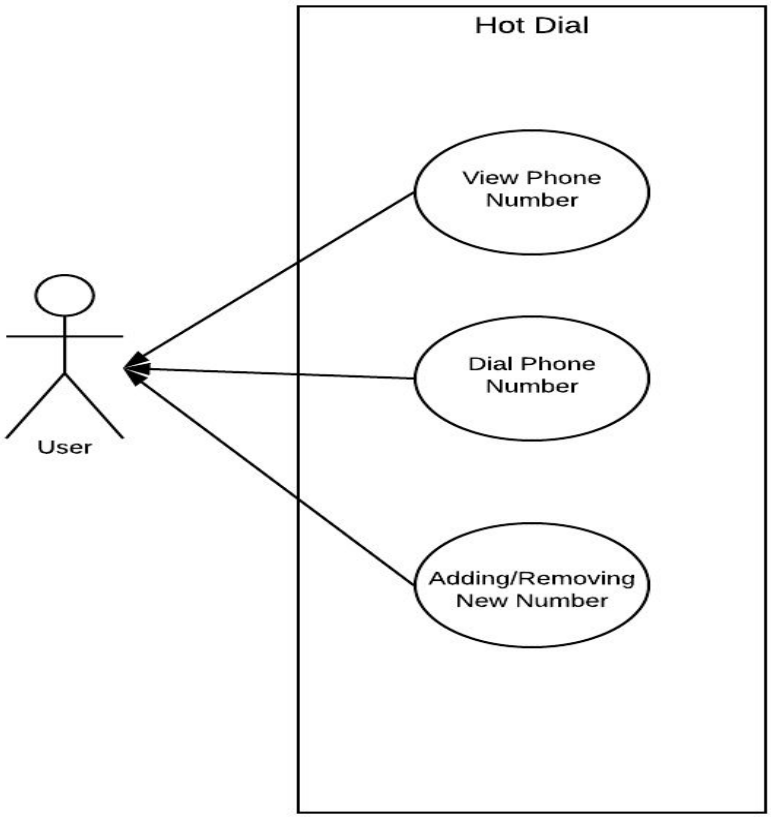


Figure 4.12
(Nakava Kibunzi)

(Steve Stamey)

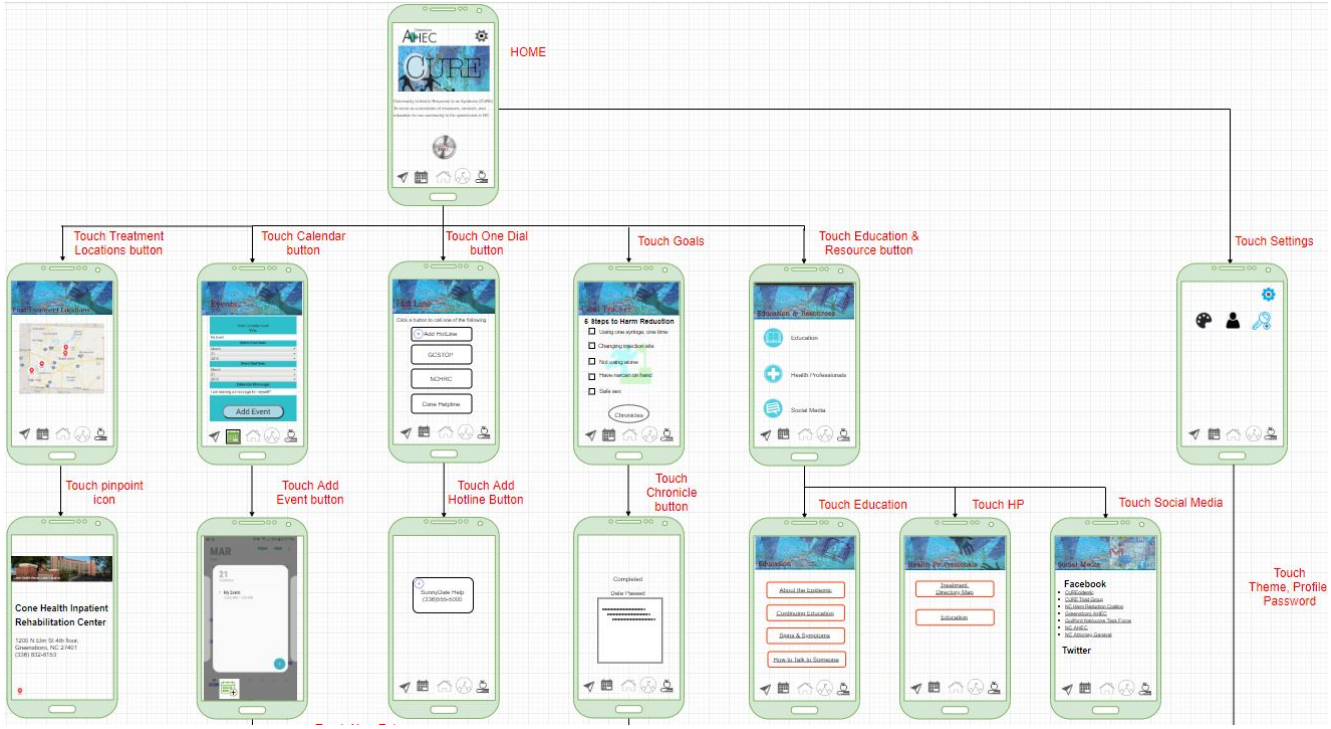


Figure 4.13a Screenshot of Mockup Subsystems Diagram

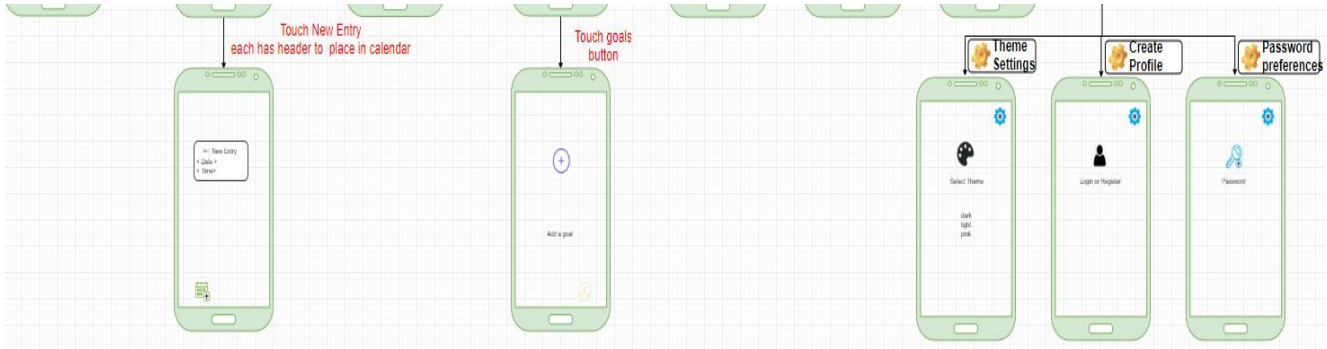


Figure 4.13b Continued Screenshot of Mockup Settings Diagram

(Joshua Brown)
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.

(Joshua Brown)
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.
    - Given 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.

(Joshua Brown)

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.

(Brian Goughnour)

Goal Tracking / Logging Subsystem

- Controls
    - User can touch plus icon which creates new goal.
    - User can touch Chronicles button to view completed goals and goals history.

- I/O
    - The user accepts a notification prompt to insert this data into their Goals/Chronicles.
    - User's input from keypad/digital keyboard
        - The user can export new events to be placed in users' Goals from their calendar.

        - Given whether an event is made or not, will notify user.
    - User prompts database to browse Goals completion and history.

- DataFlow
    - The message data from an entry can be exported from the Calendar to the GoalTracking module to store in their Chronicles.
    - The Database API receives a query to update the goals 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.
    - The Database API receives a query to view the history.

(Joshua Brown)(Steve Stamey)

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.

(Nakava Kibunzi)

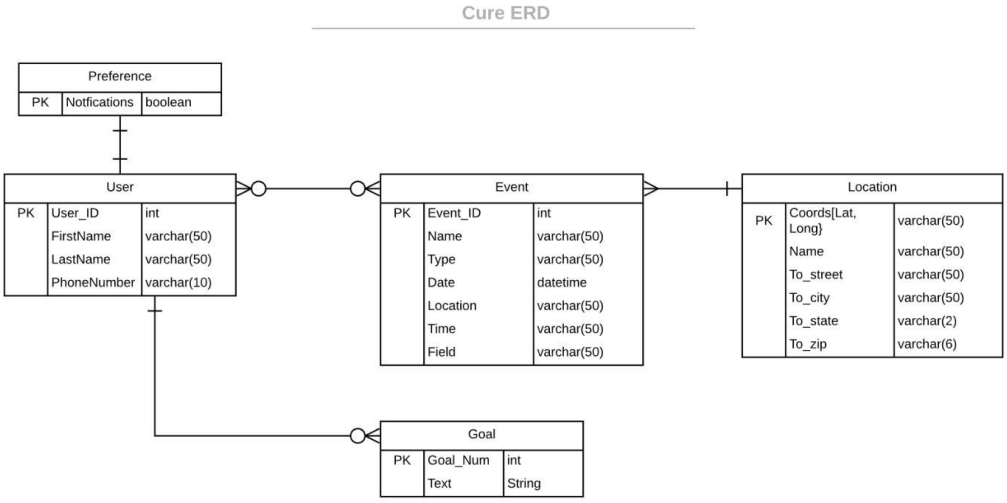
Education and Resources Subsystem

- Controls
    - User can touch the dropdown menu and it'll display the list of the link .
    - User can touch on the link to view the contents of the page.
    - User can touch the education icon to go back to education main page
- I/O
    -
    -
- DataFlow
    - All data for this subsystem remains within the subsystem.

(Brian Goughnour)

- Entity Relationship Model (E-R Model)
    - For the implementation of the database Mongo DB will be used to store location data, event times, and specific users attending certain functions as well as calendar dates.

- User may also have a settings preferences plane that can also be stored within the database. This should all be stored locally on the users phone.
- Further details regarding the database will be discussed with Dr. Sells during a meeting March 1st.



(Brian Goughnour)

(Joshua Brown)

System Model - Interactions

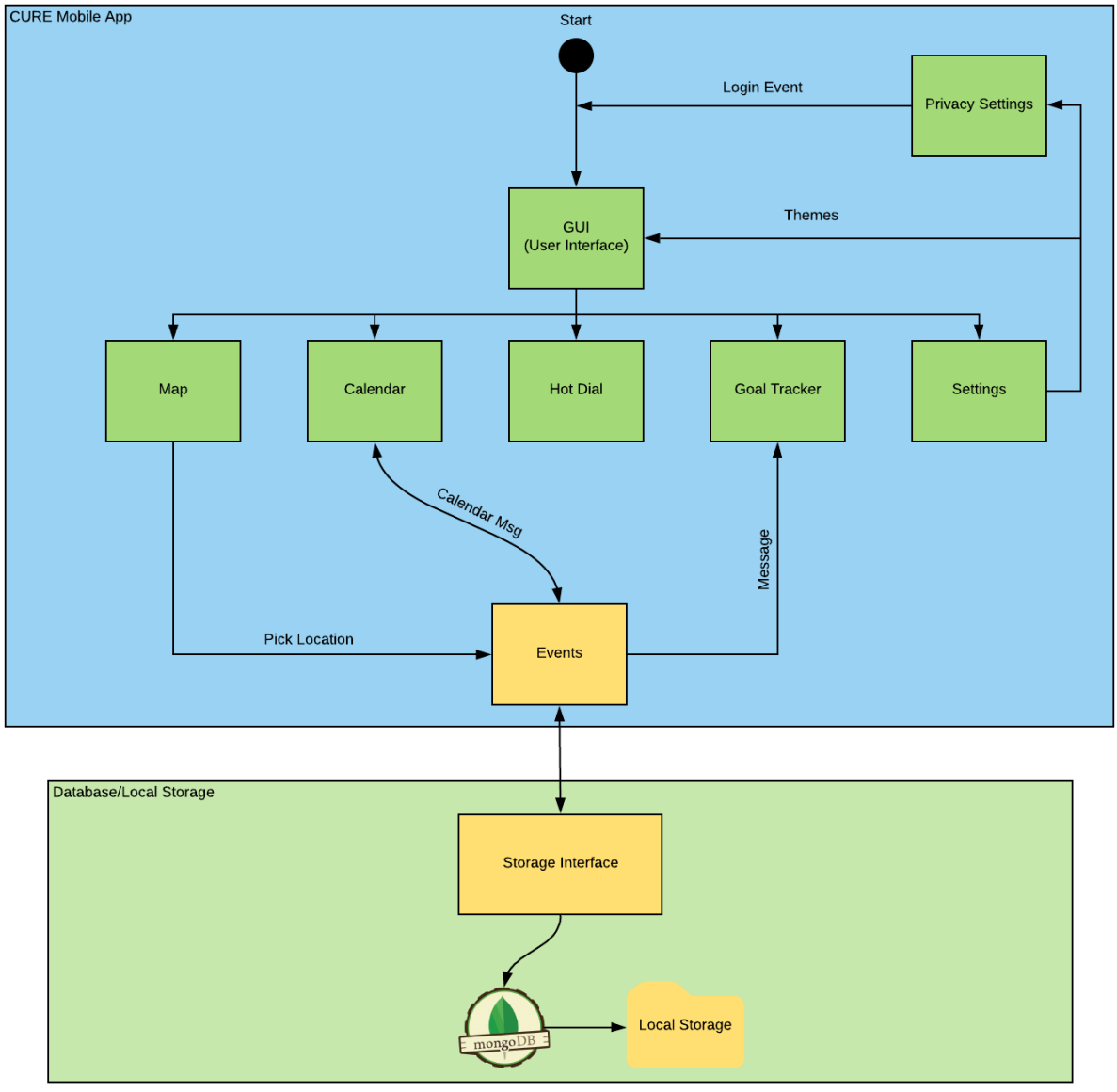


Figure 4.9

5. System – Analysis Perspective – *(Chauncey Davidson*

- Identify subsystems – analysis point of view **(Chauncey Davidson)**
  - **Identified functions of subsystems**
  - Location Services **(Chauncey Davidson)**

        - 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)
        - Touch menu shows event details and number of attendees under each event

        - Transfer chosen events to update Goal Tracking and Calendar when an event is scheduled, rescheduled, or cancelled
    - Calendar
        - Inputs received by Map/Location service when user signs up for an event; displays in cell
        - Message passing to Goal Tracker/Chronicle
        - User modifies cells via tap-and-edit interaction
        - Displays future calendar months
    - Goal Tracker
        - Displays Goals agenda with requirements to complete goals
        - Event Goals are uneditable, and are controlled through events through Location
        - Users have ability to create new goals, manually
- System (Tables and Description)
    - Data analysis **(Chauncey Davidson)**
        - Data dictionary (Table - Name, Data Type, Description)

Events

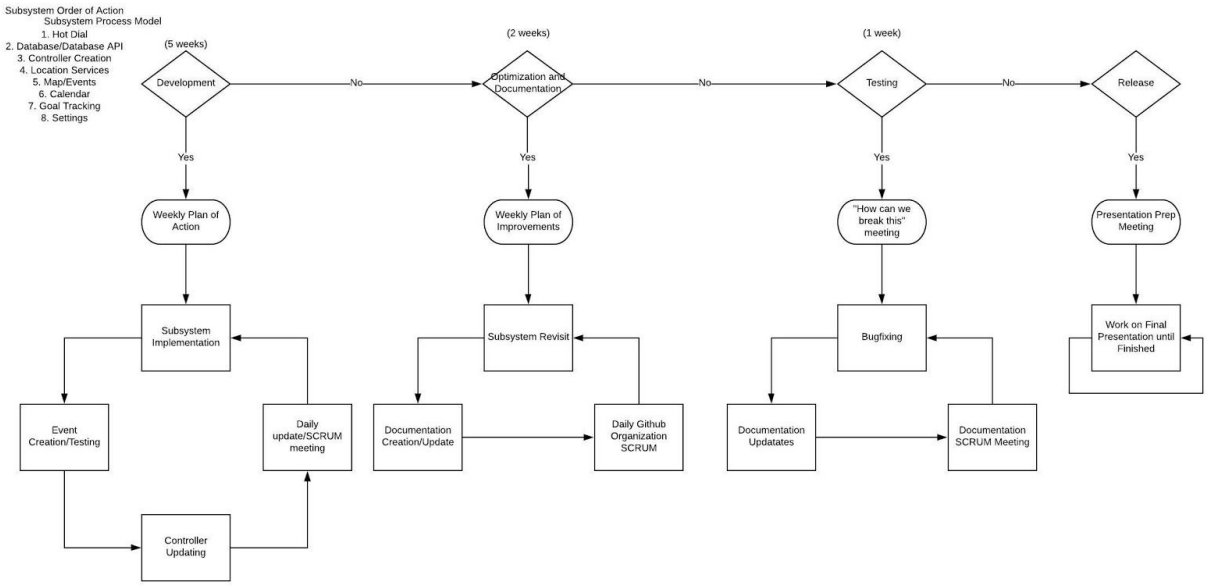
Variable	Variable Name	Measurement Unit	Allowed Values	Description
Event Identification Number (PK)	ID	int	00000-99999	ID number assigned to events in sequential order
Event Name	name	char(50)	String of Characters	Name of the event
Event Type	type	char(50)	String of Characters	Displays what kind of event to expect

Event Date	date	mm/dd/yyyy	01-12/1-31/2000-3000	Date of event
Event Time	time	int	0000-2359	Time of event
Event Location	location_name	char(50)	String of Characters	Name of event venue, from location table

Locations

Variable	Variable Name	Measurement Unit	Allowed Values	Description
Location Coordinates (PK)	location_lat_long	(double, double)	(0.0-9999.9,0.0-9999.9)	The location on the map so that Google Maps can find and place it
Location Name	name	char(50)	String of Characters	Name of the event location
Street Location Name	loc_name	char(50)	String of Characters	Name of the street of the event
State Name	state	char(2)	String of 2 Characters	Two-character string showing state
City Name	city	char(50)	String of Characters	Name of city
ZIP Code	zip	int	00001-99999	ZIP Code of event

-
    - Process model (Agile Model) **(Chauncey Davidson)**



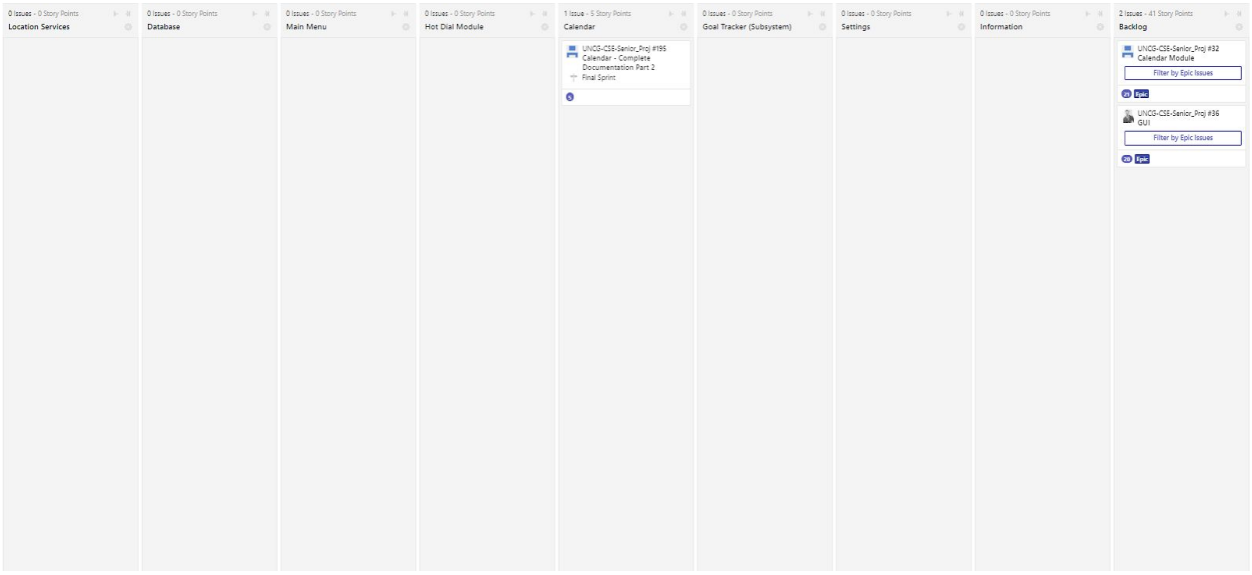
- Algorithm Analysis **(Chauncey Davidson -- including diagram)**
  - Big - O analysis of overall System and Sub-Systems - Make Chart
    - Database for event storage (Queries will be O(*log*N))
    - Database for location queries, O(N*log*N)
    - Everything else internally, O(1)

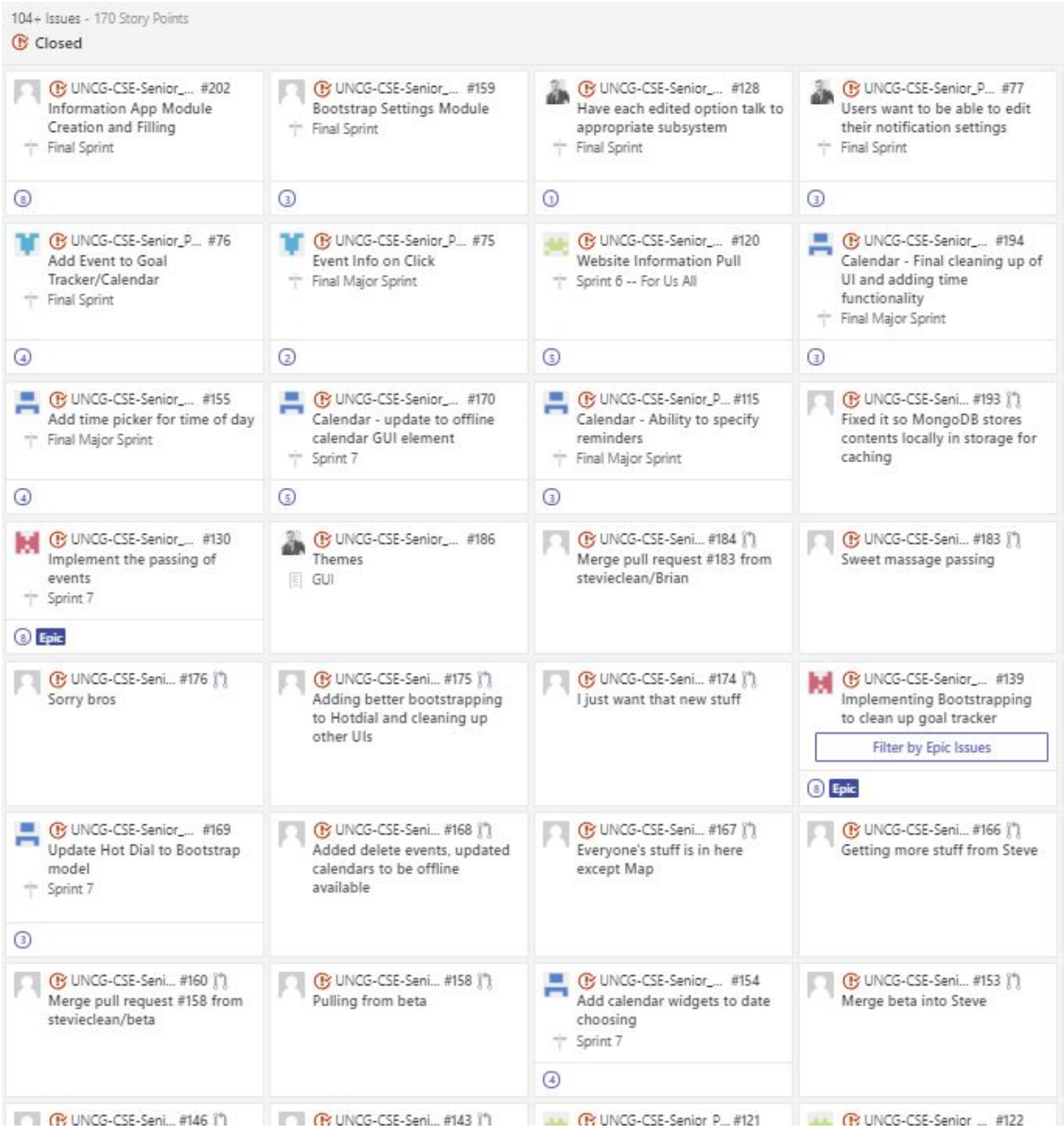
Algorithms/Operations	Time Complexity
Phone Operations	O(1)
Location/Event Query from Database	O(n*log*n)
Event Attendance Query	O(1)

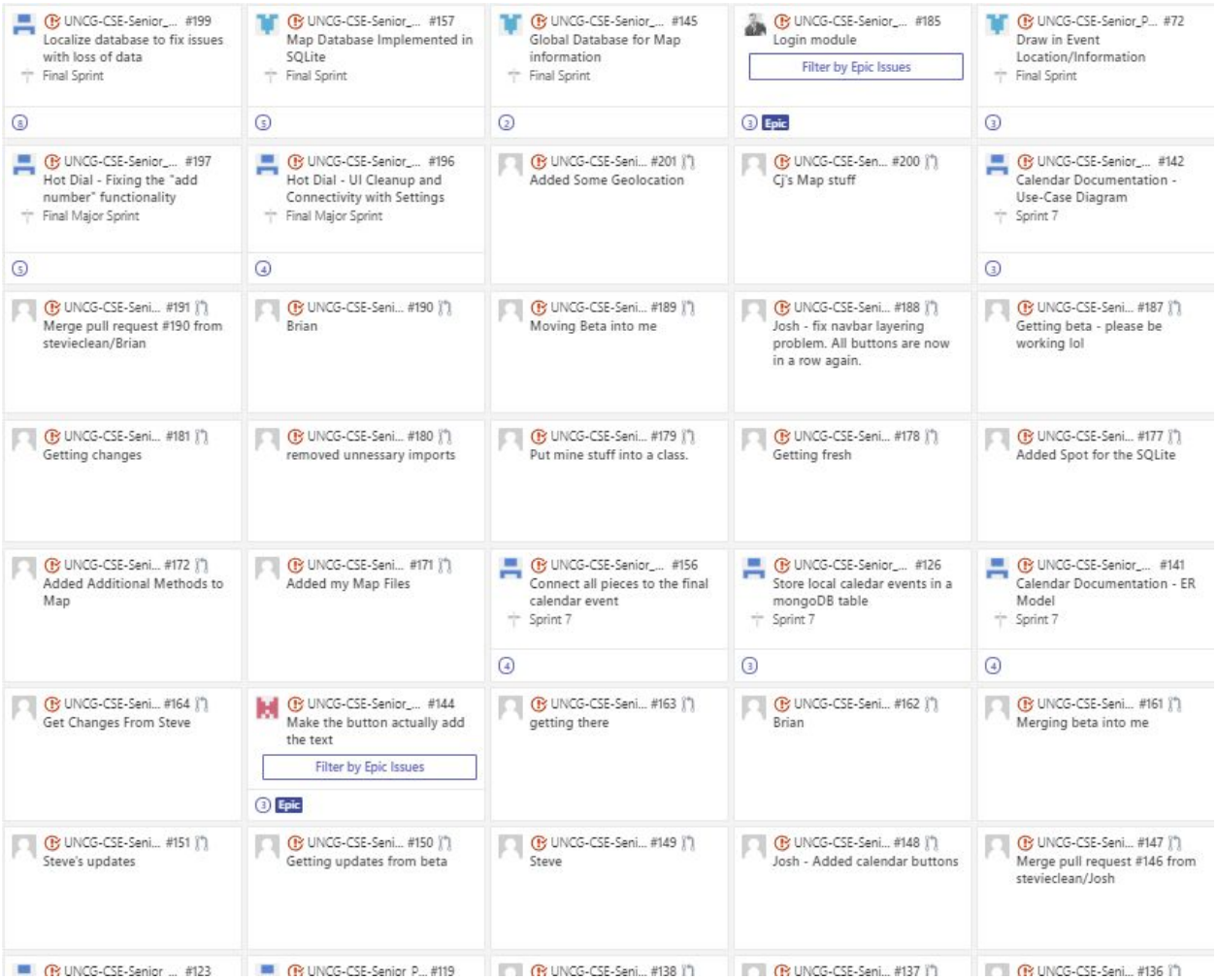
Section 6: SCRUM Reports

Product Backlog

Current Issues (4/16/19)

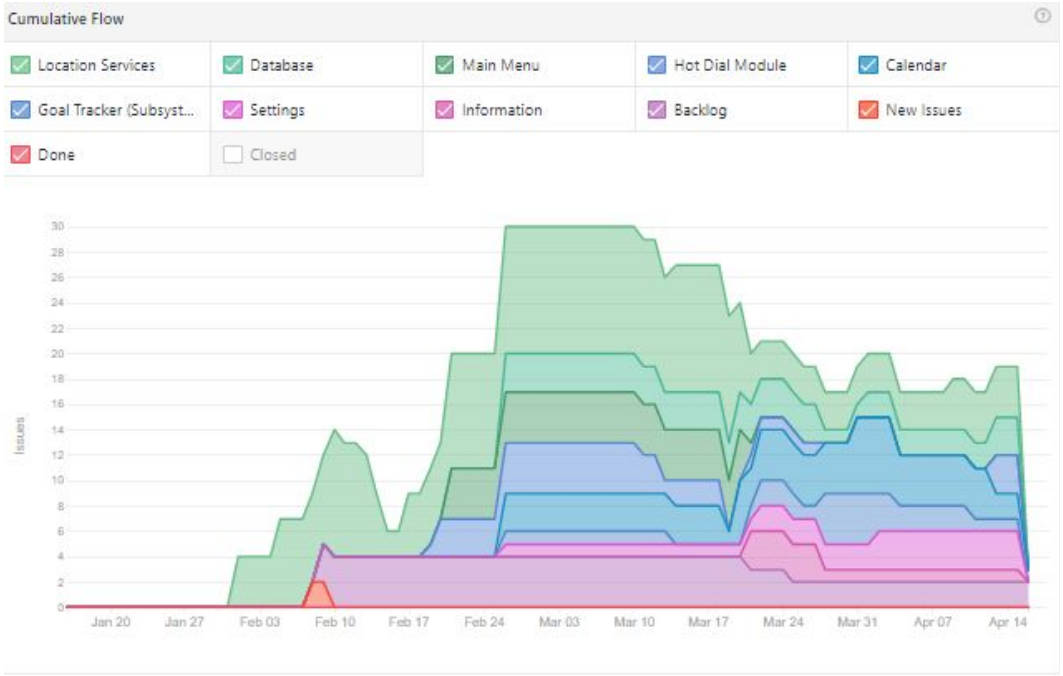



104+ Issues - 170 Story Points
Closed
UNCG-CSE-Senior_... #202
Information App Module Creation and Filling
Final Sprint
UNCG-CSE-Senior_... #159
Bootstrap Settings Module
Final Sprint
UNCG-CSE-Senior_... #128
Have each edited option talk to appropriate subsystem
Final Sprint
UNCG-CSE-Senior_P... #77
Users want to be able to edit their notification settings
Final Sprint
UNCG-CSE-Senior_P... #76
Add Event to Goal Tracker/Calendar
Final Sprint
UNCG-CSE-Senior_P... #75
Event Info on Click
Final Major Sprint
UNCG-CSE-Senior_... #120
Website Information Pull
Sprint 6 -- For Us All
UNCG-CSE-Senior_... #194
Calendar - Final cleaning up of UI and adding time functionality
Final Major Sprint
UNCG-CSE-Senior_... #155
Add time picker for time of day
Final Major Sprint
UNCG-CSE-Senior_... #170
Calendar - update to offline calendar GUI element
Sprint 7
UNCG-CSE-Senior_P... #115
Calendar - Ability to specify reminders
Final Major Sprint
UNCG-CSE-Seni... #193
Fixed it so MongoDB stores contents locally in storage for caching
UNCG-CSE-Senior_... #130
Implement the passing of events
Sprint 7
Epic
UNCG-CSE-Senior_... #186
Themes
GUI
UNCG-CSE-Seni... #184
Merge pull request #183 from stevieclean/Brian
UNCG-CSE-Seni... #183
Sweet massage passing
UNCG-CSE-Seni... #176
Sorry bros
UNCG-CSE-Seni... #175
Adding better bootstrapping to Hotdial and cleaning up other UIs
UNCG-CSE-Seni... #174
I just want that new stuff
UNCG-CSE-Senior_... #139
Implementing Bootstrapping to clean up goal tracker
Filter by Epic Issues
Epic
UNCG-CSE-Senior_... #169
Update Hot Dial to Bootstrap model
Sprint 7
UNCG-CSE-Seni... #168
Added delete events, updated calendars to be offline available
UNCG-CSE-Seni... #167
Everyone's stuff is in here except Map
UNCG-CSE-Seni... #166
Getting more stuff from Steve
UNCG-CSE-Seni... #160
Merge pull request #158 from stevieclean/beta
UNCG-CSE-Seni... #158
Pulling from beta
UNCG-CSE-Senior_... #154
Add calendar widgets to date choosing
Sprint 7
UNCG-CSE-Seni... #153
Merge beta into Steve
UNCG-CSE-Seni... #146
UNCG-CSE-Seni... #143
UNCG-CSE-Senior P... #121
UNCG-CSE-Senior ... #122



These are most of the issues that were closed over the four month period. Ranging from getting used to meteor to implementing different modules, these are the basis of the issues that were closed over the period of development.

Sprint Backlog (4 months)



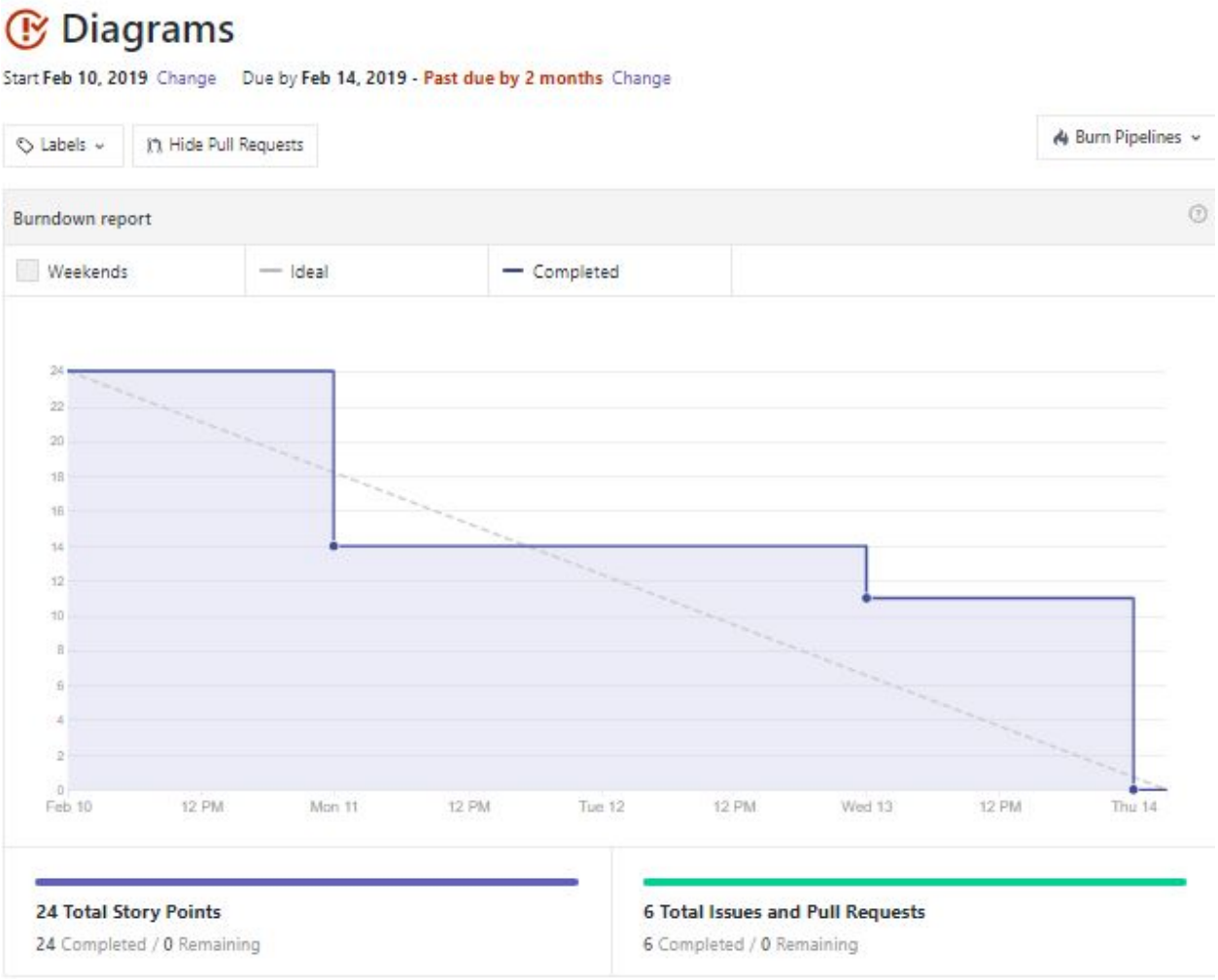
Issues in Pipeline	April 13th	April 14th	April 15th	April 16th
Location Services	4	4	4	0
Database	3	3	3	0
Main Menu	0	0	0	0
Hot Dial Module	3	3	3	0
Calendar	2	2	2	1
Goal Tracker (Subsystem)	1	1	1	0
Settings	3	3	3	0
Information	1	1	1	0
Backlog	2	2	2	2
New Issues	0	0	0	0
Done	0	0	0	0
Closed	180	180	180	199

In January, zenhub was still new and was difficult to work around, but this month was used as the planning phase for the project; this is where the group met and discussed how the project would be addressed going forward.
In February, in order to get used to the way zenhub operates, issues were made for the various charts and models for the project. Some of the pipelines were made and edited at this point for organization's sake. Towards the end of February, the project's skeleton

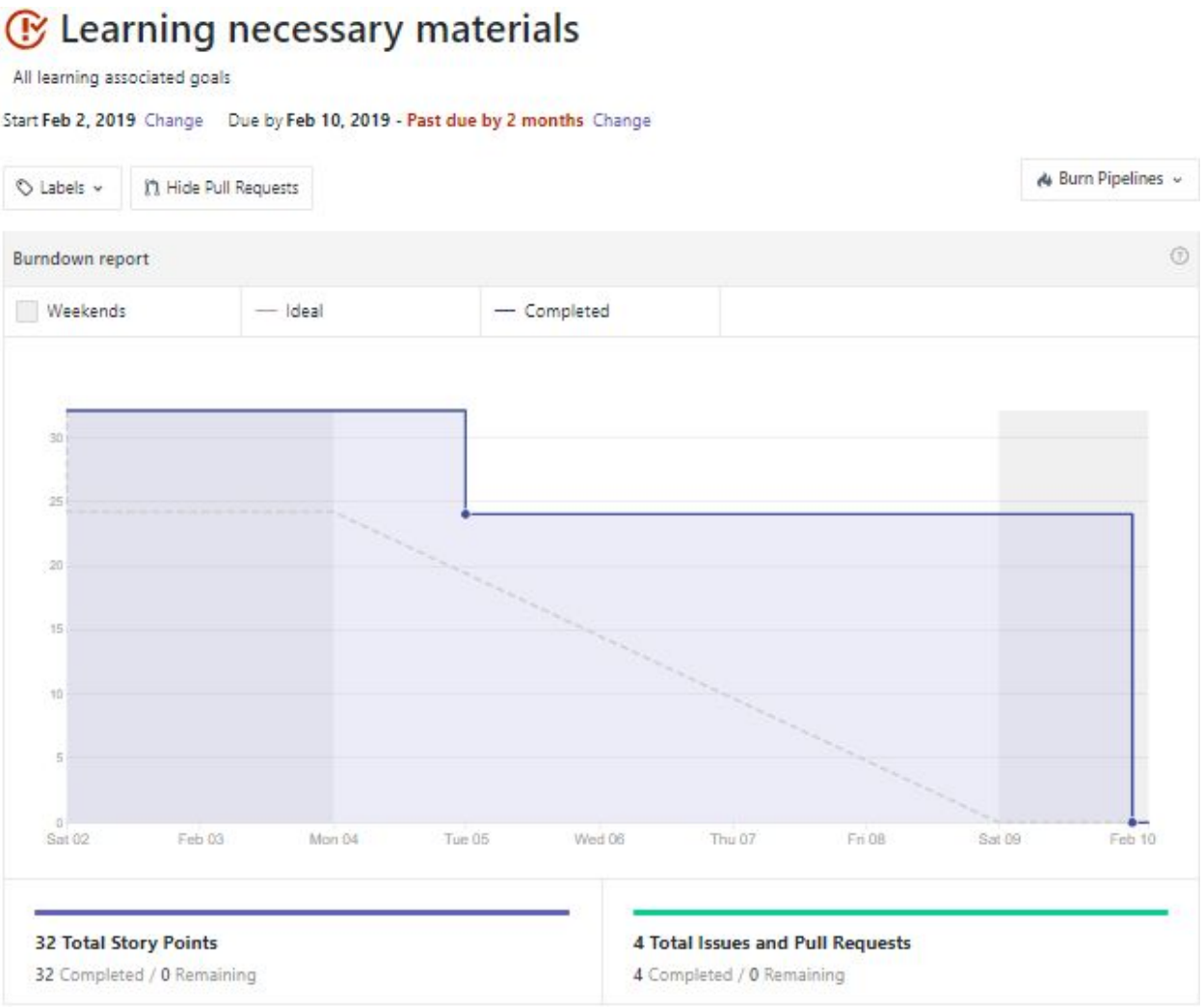
was made, and more issues and pipelines were made to accommodate the new issues that arose.
Near the end of March is where most of the issues were taken care of and closed. There were many issues surrounding the mobile version of the app, which were difficult to pinpoint. Despite the problems, the issues started to be taken care of.
Currently in April, in the project's final stages, the final issues are starting to be closed. Most of the development is done, there are only minor bug fixes and UI updates to make to the project before release.

Burndown Charts - Developmental Sprints

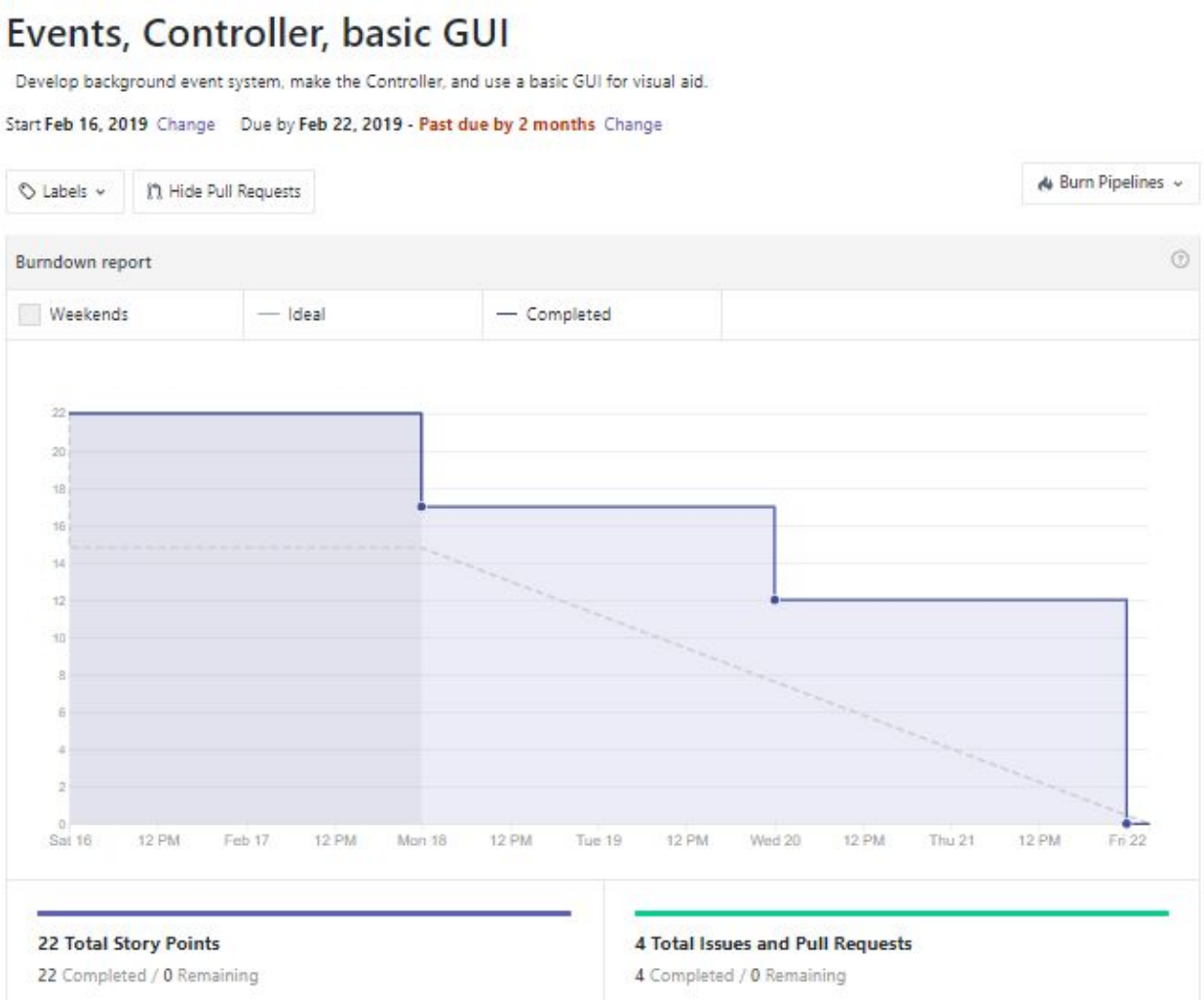
Week 1 - Finishing Diagrams



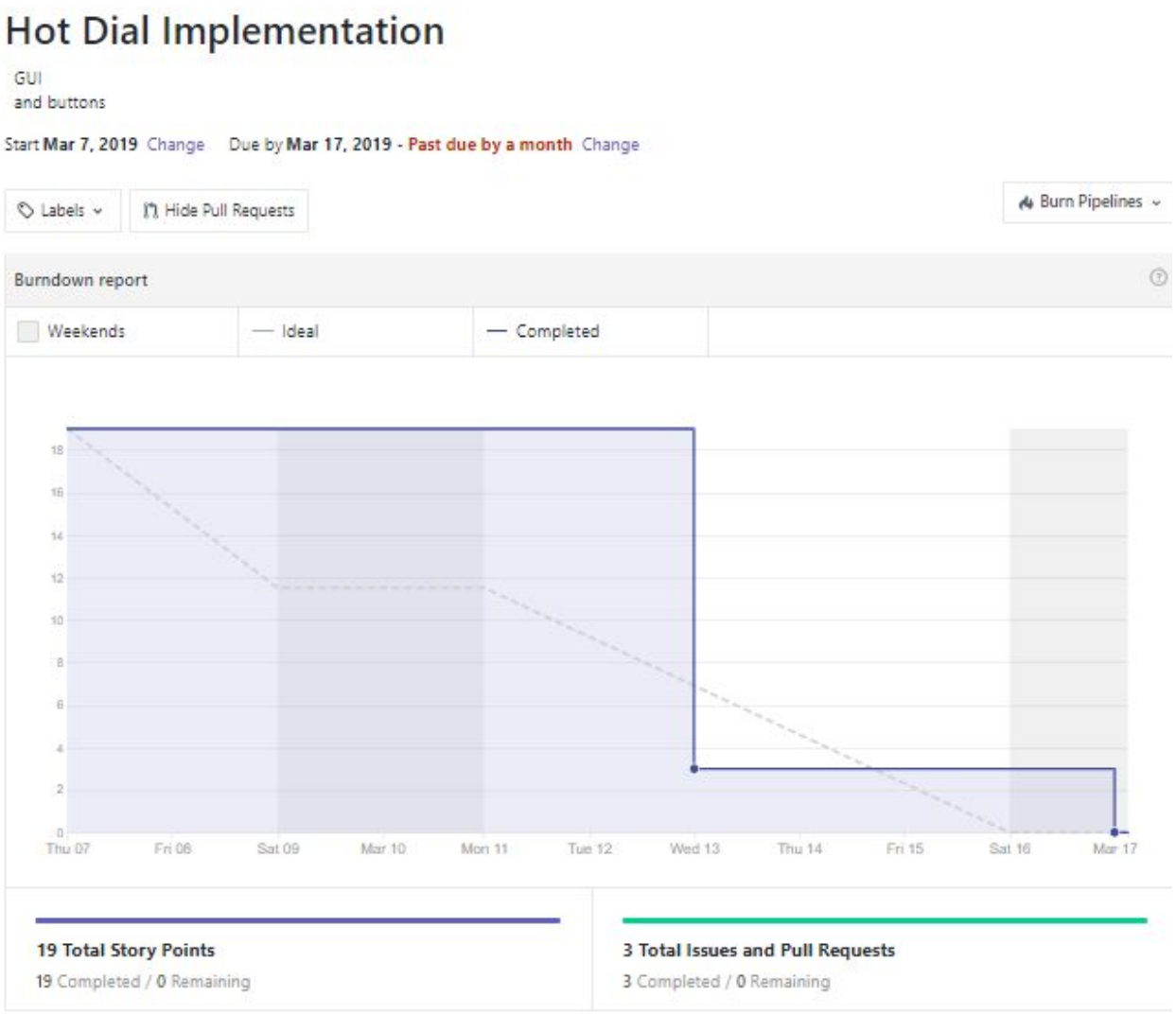
Week 2 - Research Week



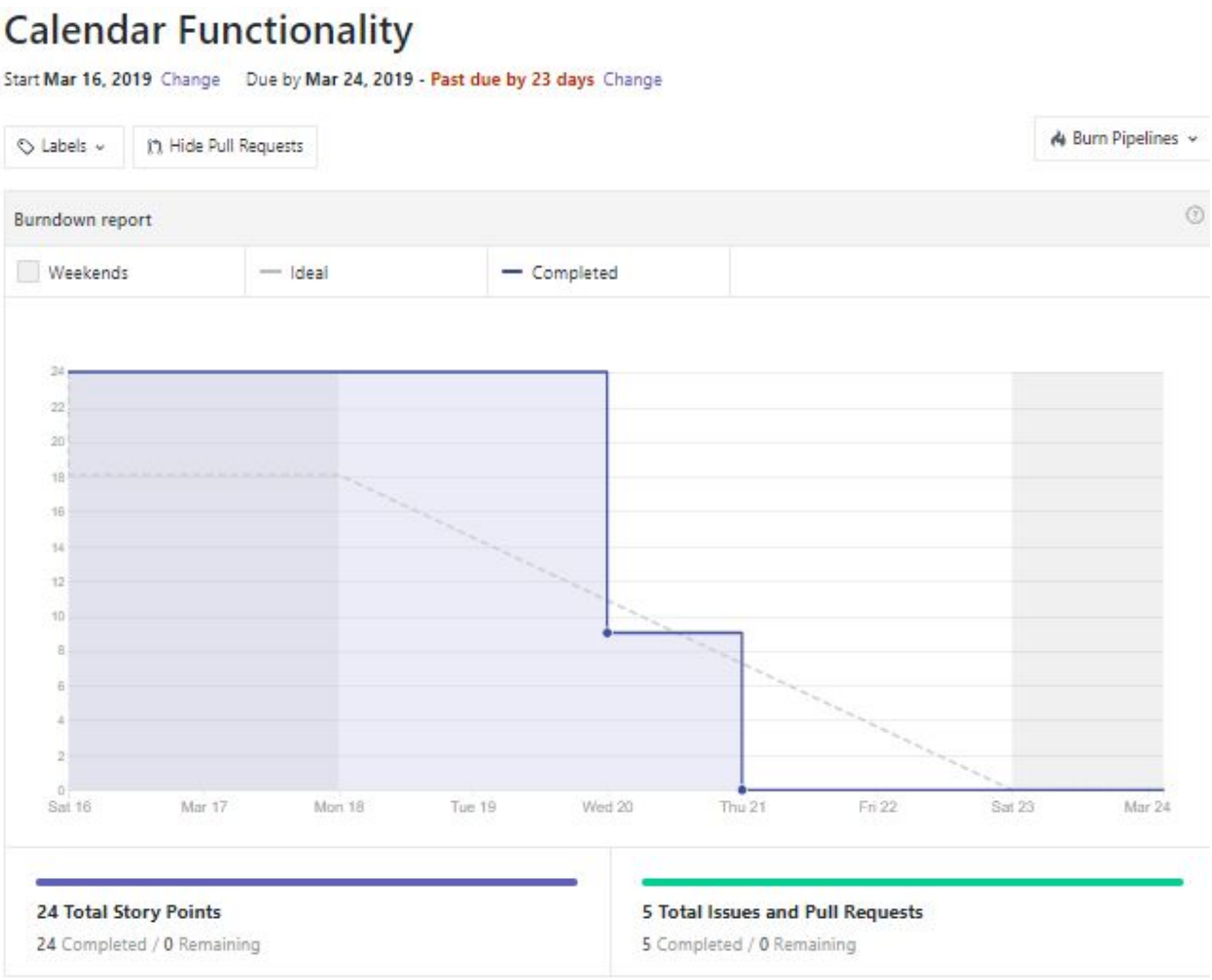
Week 3 - Application Skeleton Setup



Week 4 - Collaboration on Hot Dial



Week 5 - Module Implementation



Week 6 - Module Implementation pt. 2

Sprint 6 -- For Us All

Updating Calendar to include notifications, allow cancelling of events, storing events in a local MongoDB table.

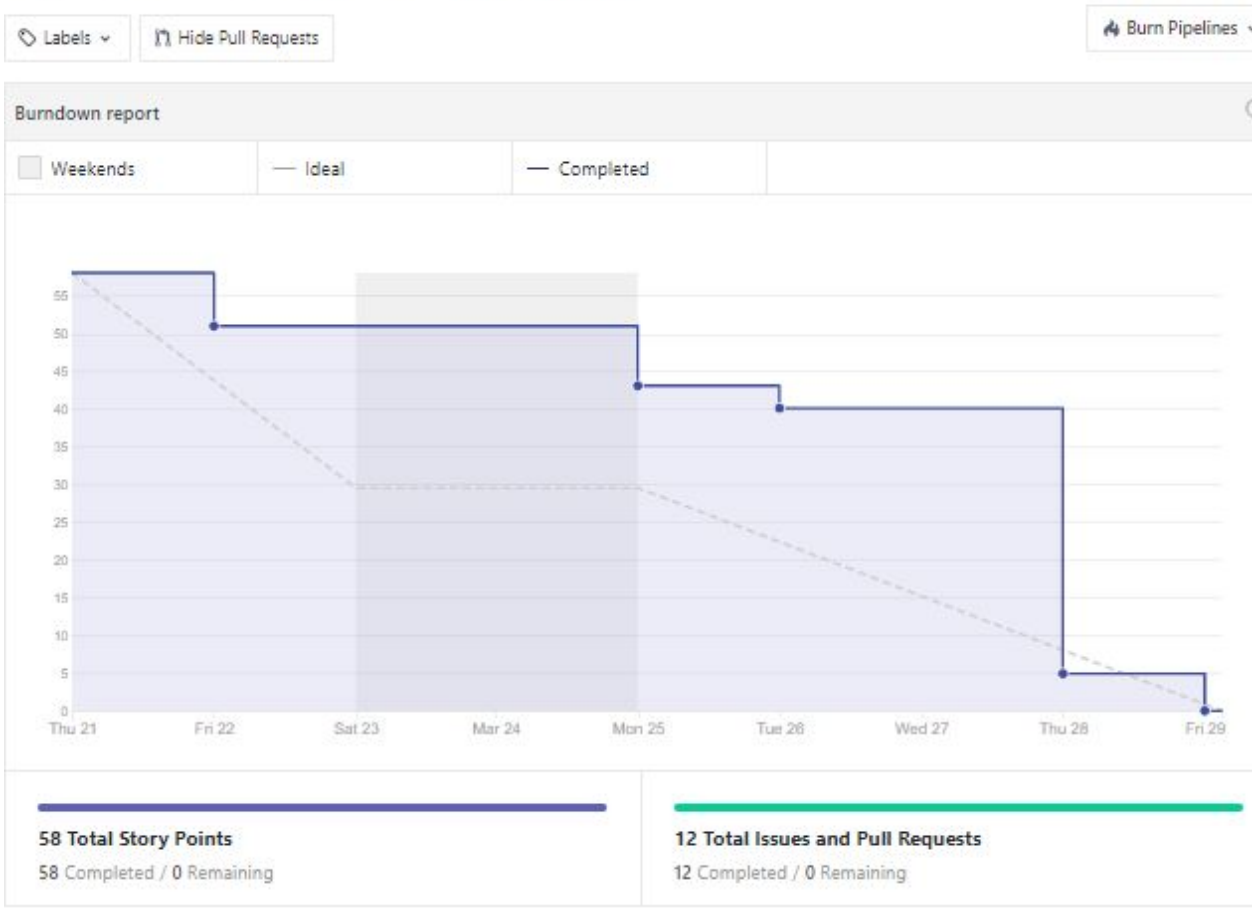
Updating Settings menu to include new menus for some of the buttons and a removal of unnecessary components. Make settings menu communicate with other subsystems.

Write database to be handled for necessary subsystems. Relevant

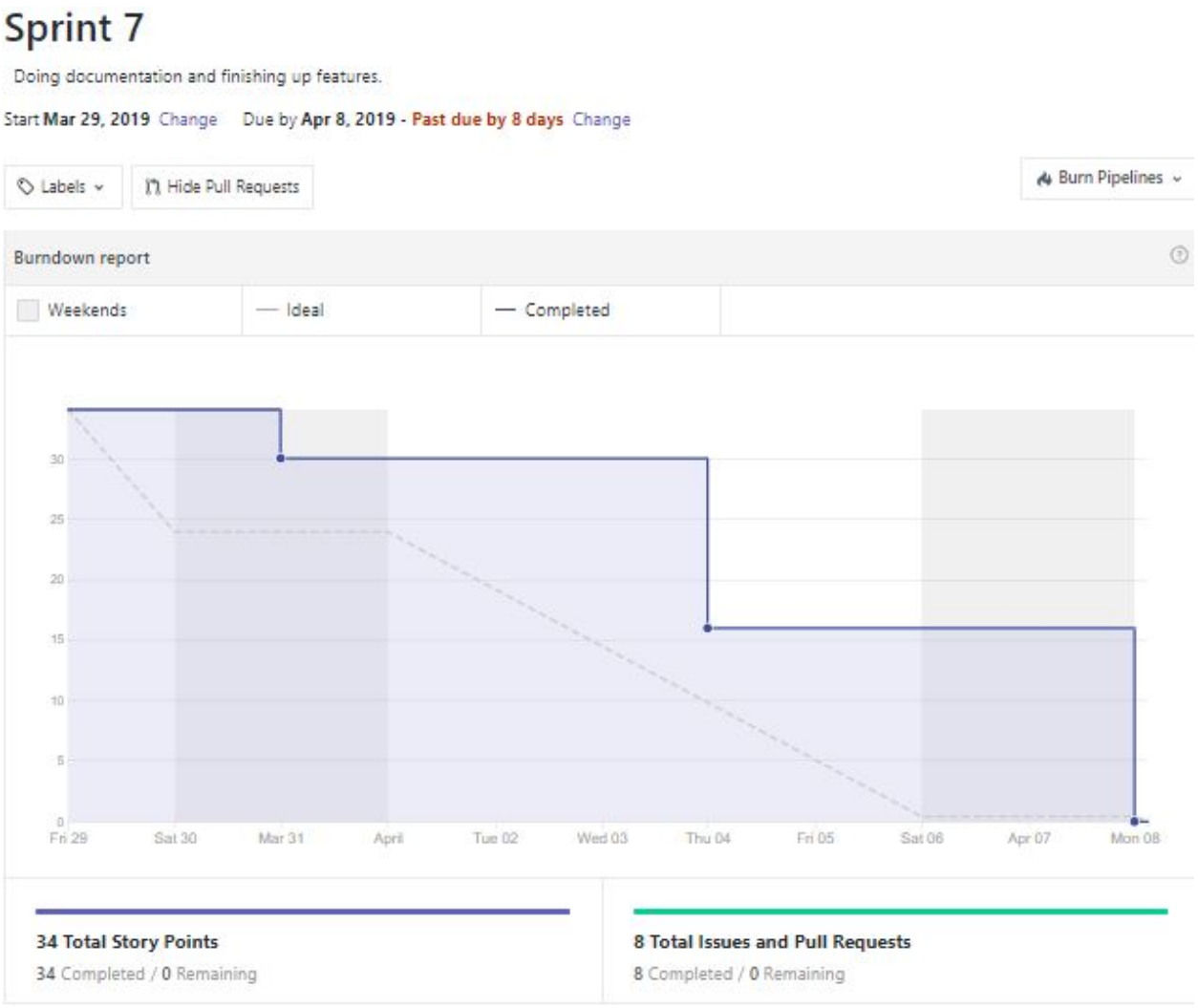
Get goaltracker integrated into main system. Include base options for user as potential goals. Store goals into a local MongoDB table. Communication between calendar and goaltracker.

Leaflet maps needs to be imported. Add blips from a csv file. Once a blip has been selected on the map information will be displayed about the location If the user wishes, they can add this location to their calendar as an event, which will subsequently also be asked to become a goal in their goaltracker.

Start **Mar 21, 2019** Change Due by **Mar 29, 2019** - **Past due by 18 days** Change



Week 7 - Start Documentation, Bug fixing

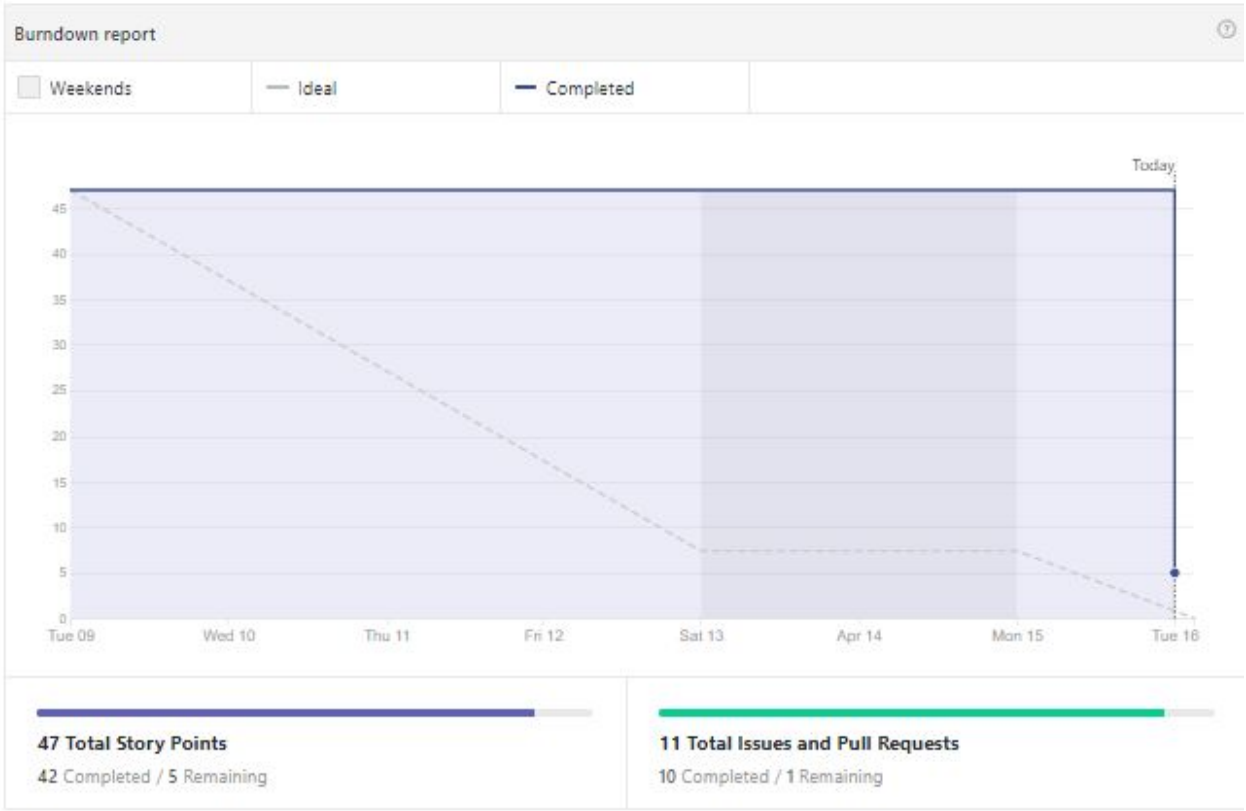


Week 8 - Additional Bug Fixes

Final Sprint

Start **Apr 9, 2019** Change Due by **Apr 16, 2019 - Due today** Change

Labels Hide Pull Requests Burn Pipelines



47 Total Story Points

42 Completed / **5** Remaining

11 Total Issues and Pull Requests

10 Completed / **1** Remaining

7.1 Subsystem 1 Hot Line (Joshua Brown)

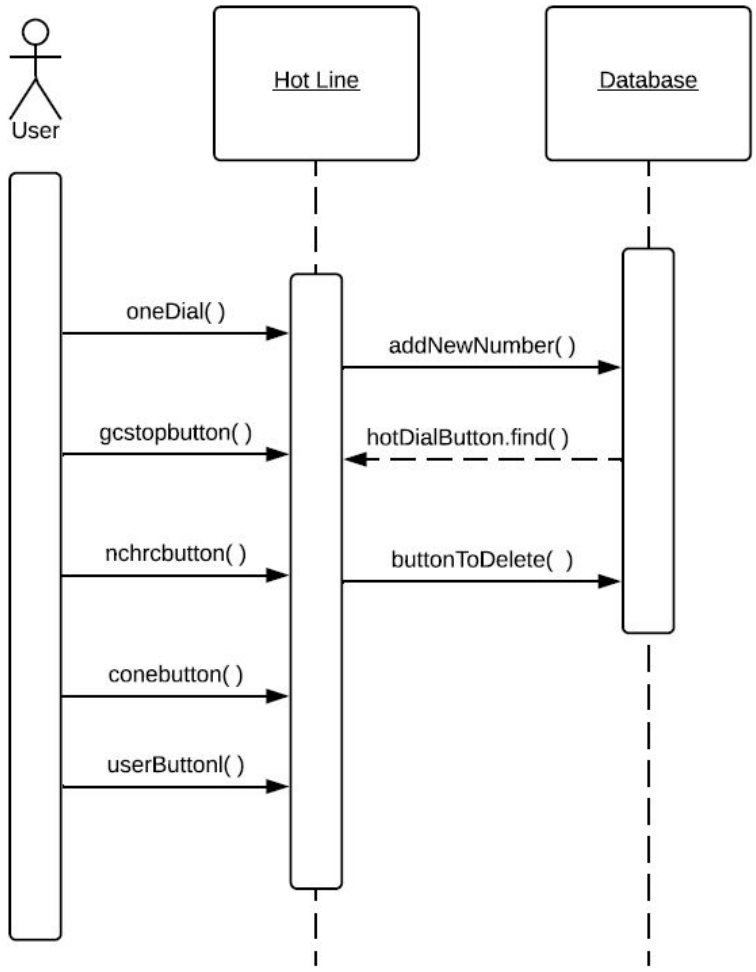
Initial design and model

The initial design of the hot line subsystem was to have fixed numbers that the user could touch in order to automatically call emergency services related to the opioid crisis such as GCSTOP. The user simply accesses this subsystem from the main screen at all times by touching the red phone icon.

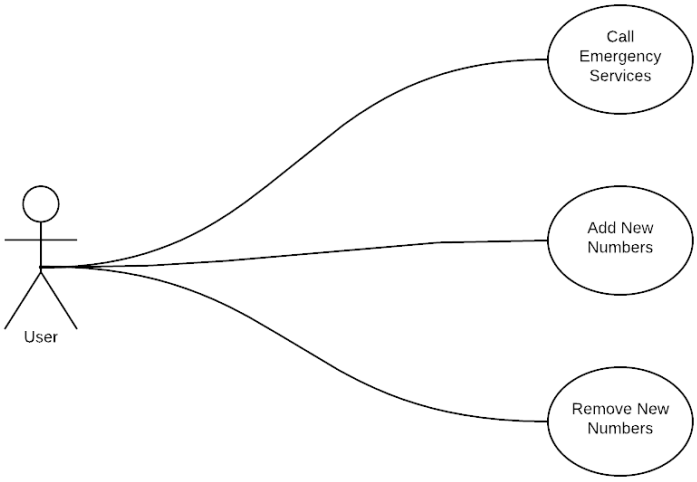
UML Diagram

Hot Line
-name : String -phonenumber : String -templateHelper : events() -templateRender : onRendered()
+get listeners() : list +handleMessage(message) +sendMessage(message)

Cure Application Sequence Diagram (HOT LINE)



Use Case Diagram



Design Choices

Design choices were made across time for this module. The main, original concept allowed users to quickly call needed services in the local community. Later, it was decided that we should allow users to define their own emergency numbers and remove them as needed.

Data dictionary

Name	Data Type	Description
Name	varchar(10)	The name of the place or person
Phone Number	varchar(10)	10-digit phone number for this entry, which is used to make a phone call

Refinements

The hot line submodule has only had a few necessary tweaks over the course of the application. The first change was to implement bootstrap and theming into the subsystem. This made it much more appealing to operate and view. The other major

change was the implementation of the “add” and “delete” function for new phone numbers.

The “add” and “delete” functionality includes implementation of MongoDB combined with a local storage manipulation of the server data. This was an important change as it improved run time a lot compared to the original implementation. Originally, the implementation used local file storage at every call. The current implementation uses MongoDB which will store in RAM (much faster) but will only call local storage at the opening of the program and once insertion is completed.

Scrum Backlog - See Section 6 (pg. 30)

Coding

Approach

The hot dial submodule was built using object-oriented programming. The subsystem itself is a class object created through a Controller. Reactive variables were used via Templates from Blaze in Meteor and the MongoDB collections which were used to create/delete new users.

Language

The languages used for the hot dial were HTML, Javascript, and CSS. HTML was used for user interface design, Javascript was for the backend logic of the module, and CSS was used to aid the design of the user interface. In addition to standard CSS, the bootstrap CSS library was used to add additional functionality to UI elements and provide an all around cleaner look and feel.

User Training (See page 62)

Testing

Hot Line’s user training was relatively straight-forward. The most significant benefit from sample runs was trimming down the time it took the user to get to and make phone calls. Initially users had several menus and even required confirmation before calls. This proved to be too slow and hurt the goal of emergency service for emergency situations.

From further training sessions with hot dial we determined that allowing users the ability to add and remove emergency contacts may be a vital and important feature. This is vital because everyone has loved ones and personal healthcare professionals with whom they may need to immediate contact.

7.1 Subsystem 2 Calendar (Joshua Brown)

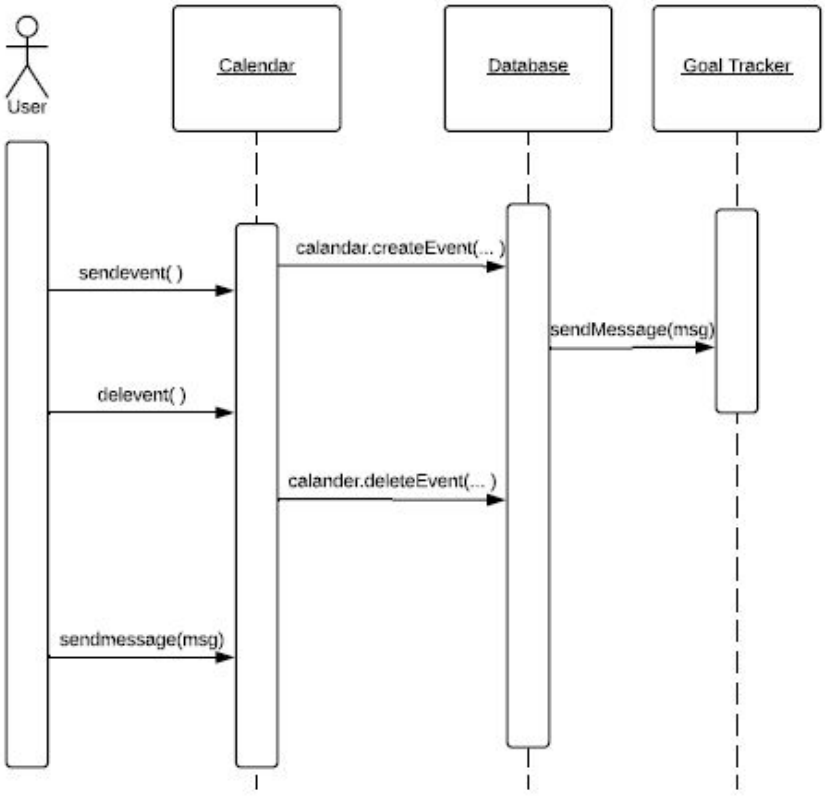
Initial design and model

The initial design of the calendar was to have a fully built and integrated calendar within the subsystem. It would contain normal features of a calendar, such as adding events within dates, editing them, and removing them. The subsystem would also accept Map pinging events which would pass data to the calendar for making a calendar event. The calendar itself would create a simply message which could be passed along to the GoalTracker to be cast as a goal in its list.

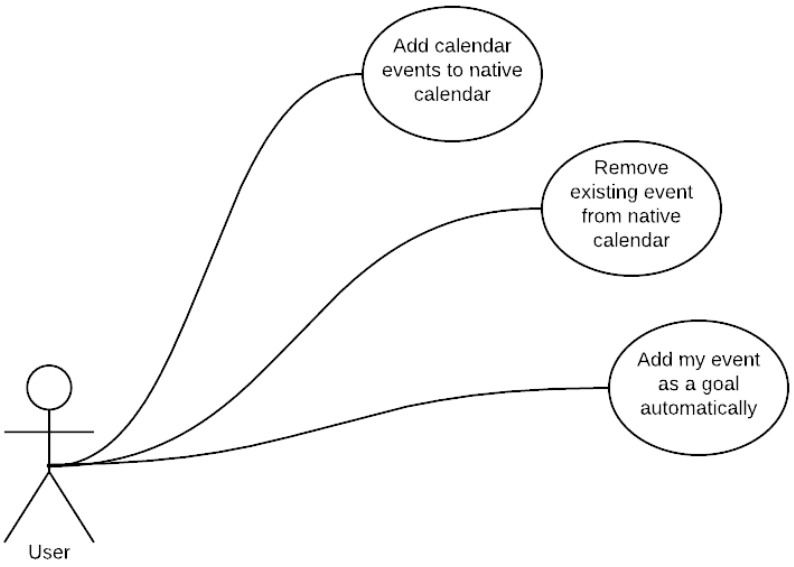
UML for Calendar

Calendar
-title : String -notes: String -templateHelper : events() -templateRender : onRendered()
+get listeners() : list +handleMessage(message) +sendMessage(message)

Cure Application Sequence Diagram (CALENDAR)



Use Case



Design Choices

Originally, the idea was to create a literal calendar that would be available to the user in-app. It was decided later in the project that this was not necessary and it would be better to have the subsystem create calendar events which would be passed to the user's native/preferred default calendar. This allows the user to still maintain control, but whenever they make appointments in the app, the subsystem can automatically generate those events to Google/iOS calendars.

Some key choices that were made during the designing phase were to modify the look and feel of the GUI of the subsystem, utilize calendar pop-ups for time and date settings, and pull events to the goal tracker subsystem with the title of the event being displayed there.

Data dictionary

Name	Data Type	Description
Title	varchar(100)	The title of a calendar event
Message	varchar(500)	A description or explanation of a calendar event
Date	Date	A full date with the format mm/dd/yyyy
Time	varchar(10)	The time with hours and minutes of a calendar event in the format HH:MM

Refinements

The calendar module went from being a full-fledged calendar, into an event creation/deletion module. The reason for doing so was that we really needed the application to utilize the native calendar on phones to meet requirements of Apple's app development guidelines. Applications on the app store must use functionality from the phone itself in addition to other functionalities it may have.

The benefits of this approach were that this subsystem took a lot less time to develop, since creating a full calendar with containers for information events wasn't necessary. We can simply implement calls to phone operations and pass the necessary data to the phone's API to get the desired result.

Some detriments of these refinements include disconnectivity between the application and features which can cause slowdown depending on the current usage of the phone's native operations. Also the possibility of dependencies breaking down may ruin the functionality of this subsystem. While the subsystem is mostly self-contained, the fact that it now depends on the operation/handling of the native calendar, this module may need to be updated in the future to be compatible with future iterations of Android and iOS calendars.

Scrum Backlog - See Section 6 (pg. 30)

Coding

Approach

The calendar submodule was developed using an object oriented model of design. This subsystem was designed to hold calendar events and then make calls to the phone's native calendar system with the appropriate inputs for creating a calendar event. Whenever a user wants to create a calendar event, he/she gives keyboard input for the "Title" and "Message" section of the window. For the "Time", the user will simply enter time as it needs to be entered in order to pick a time for the event to be stored under. As for the "Date" section, the user clicks on the textfield, which automatically generates a small calendar box for the user to pick a date from. A backend message passing, event system was used specifically to pass messages to other subsystems. Whenever an event is crafted a message is passed to the GoalTracker subsystem which may store the event as a goal.

Language

The languages used for the calendar were HTML, Javascript, and CSS. HTML was used for user interface design, Javascript was for the backend logic of the module, and CSS was used to aid the design of the user interface. In addition to standard CSS, the bootstrap CSS library was used to add additional functionality to UI elements and provide an all around cleaner look and feel.

User Training (See page 62)

Testing

For the calendar subsystem there wasn't an opportunity for legitimate user testing, however each of the developers regularly tested the subsystem. Along the training process we changed the user interface multiple times to better illustrate the features and make them as easy-to-use as possible.

In addition, as part of training we discovered that features such as the mini-calendar and time select wheel should be added. Their addition was relevant to the application in order to maintain our goal of easy and quick access to calendar events and to make interoperability between subsystems streamlined.

7.2 Subsystem 3 – CJ - Map Subsystem

Initial Design and Model

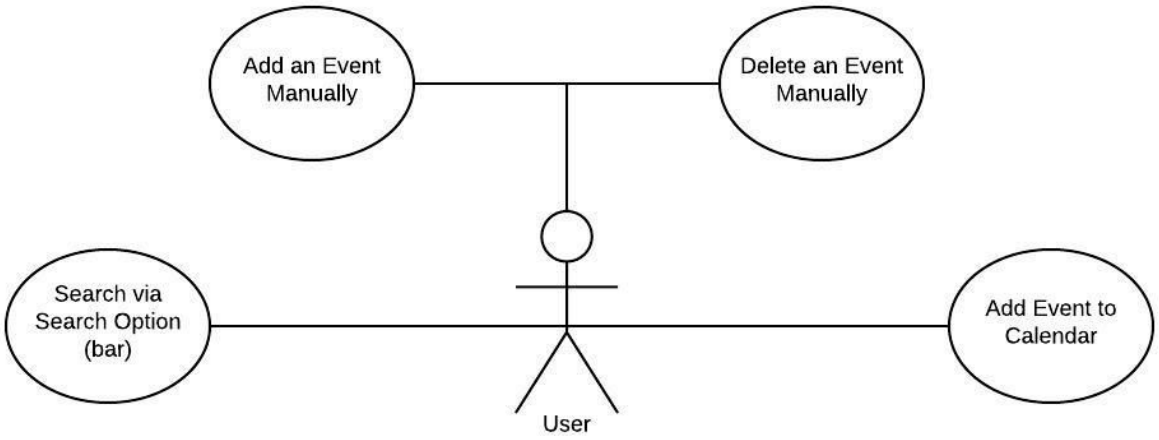
The first design of the program was a pipe dream, but it was feasible. The model was based around a working server with a database that is constantly active, but was based on the assumption that our client's website had a database of events I could draw from.

The basis of the model was to draw events from the database and display them on the map. These events would pop up on the map in terms of markers, and the users would be able to click on them and add them to the local database of their stored events, and they would be able to cycle through them via a tabbed system. Additionally, if they wanted, they would be able to click on a marker and potentially add it to their Calendar. The user should be able to traverse the map freely, also.

This is the UML model for the map:

Map Subsystem
-long_lat : <double,double> -address: String
+long_lat_to_address() : String +address_to_long_lat() : <double,double> +addAddress(): boolean +removeAddress() : boolean +addMarker(): boolean +removeMarker(): boolean +listMarkers(): boolean +sendMessage(message)

Use Case Diagram



Title	Data Type	Description
Address	varchar(500)	Physical address of a location.
Long_Lat	<double, double>	Longitude and latitude. The map uses these as coordinates.
Marker_Status_on_map	boolean	How the map generates its markers.

Refinements

I made a lot of refinements to the model over time, especially after our first initial meeting with Dr. Sill. After inquiring about the database and where to pull the data from, and I was sent a CSV file. This changed my perspective a bit, mostly because I assumed that there was already a database created from the previous team that contained the events that I can pull from, but that did not exist. Because of this, I decided to give the users the option to set the markers themselves, and add the dates themselves to their own markers. Additionally, I had some trouble with Mongo because I thought the functionality of it was global. I was stuck on this fact for a while, but I ended up fixing it by giving the users choice. Additionally, I may add some routing, search bar, and deletion methods that I will attempt to implement.

Coding

Approach

We started coding using Meteor, a platform that easily ‘translates’ web-app code (HTML, Javascript, CSS) using Blaze. I originally had trouble with developing using this platform at first because of its newness, and it was a steep learning curve. The way meteor handles errors, along with web development in general, was all new information for me. I started my coding by creating the database in Mongo, but I was having trouble because I did not understand that we needed to put a database import on the server side as well. After I figured that out, I attempted to work with Leaflet, the open source map that’s being used in the project. The rest of the project has been working with imports from leaflet, including database queries, on-map searches, and traversals.

Languages

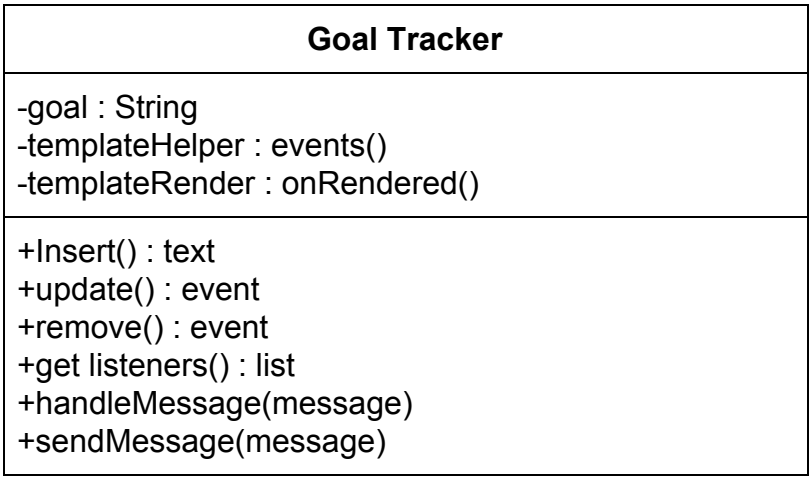
I primarily used HTML and Javascript, along with different commands that work with leaflet. Most of my work has been in Javascript, dealing with leaflet methods and commands, trying to manipulate the map and load things onto it. Some of the HTML commands were used to form the containers for the different sections of Javascript commands and forms.

7.2 Subsystem 4 Goal Tracker (Brian Goughnour)

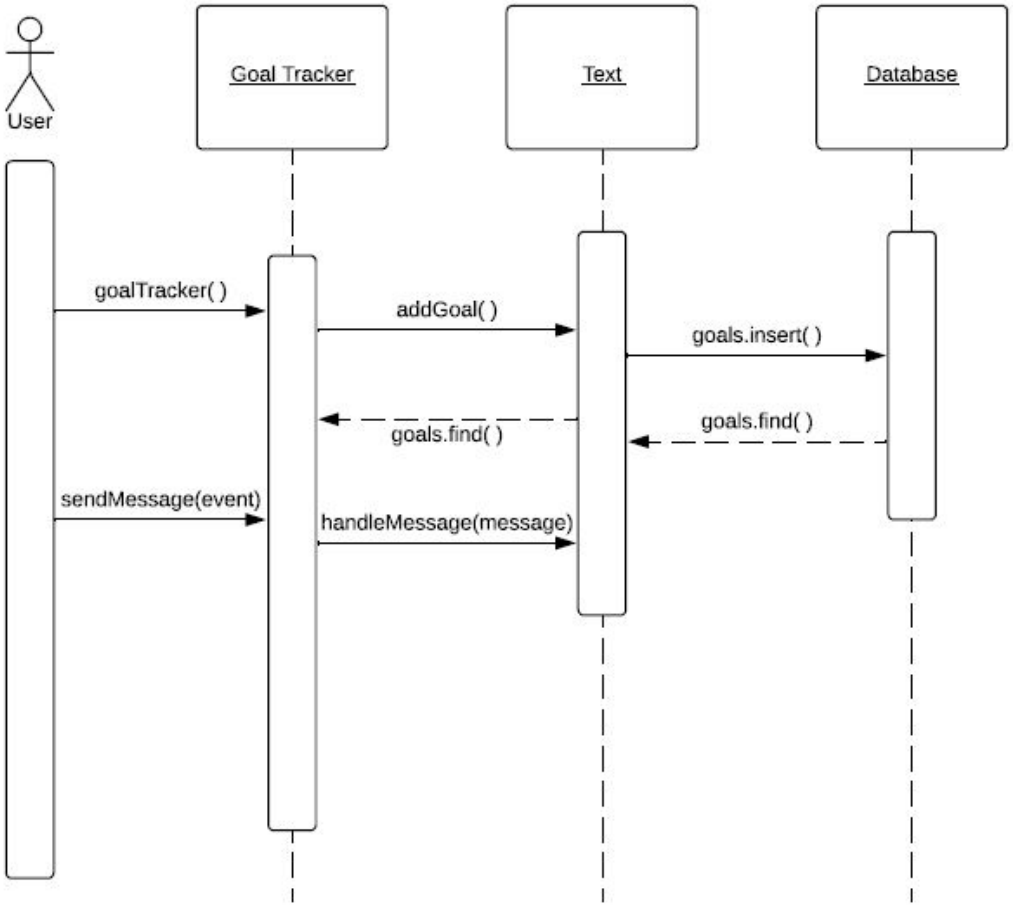
Initial design and model

The initial design of the goal tracker was to have a fully built and contained goal tracker within the subsystem. The goal tracker was to have the standard features of any goal tracker. One feature would be the ability to populate the list of goals by inserting text as a new goal. The next feature would be the ability to update the goal as the user completes them. Finally for the standard features the ability to delete the goal once it was completed. A unique addition to the feature set of this goal tracker was its ability to receive and create a new goal from the calendar subsystem upon creation of new event. Lastly, the goal tracker was given the ability to change themes given changes made within the settings subsystem.

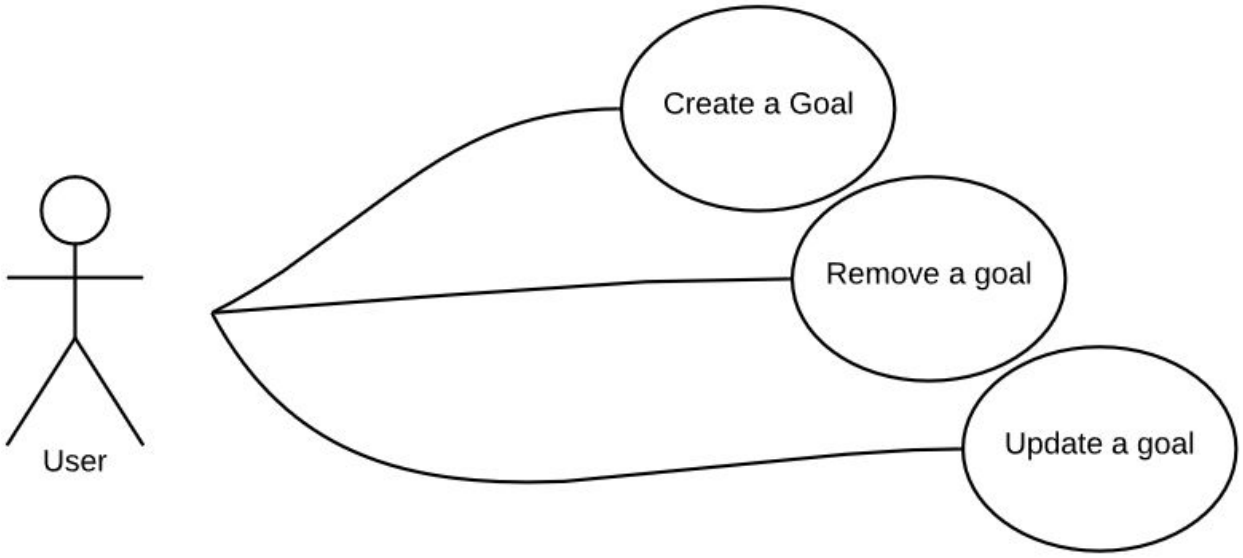
UML for Goal Tracker



Cure Application Sequence Diagram (GOAL TRACKER)



Use Case Diagram for Goal Tracker



Data dictionary

Name	Data Type	Description
Text	varchar(500)	The text of the goal

Refinements

The goal tracker module was designed to take a users text input and create a list of objects. Originally the entire subsystem was created using the react framework. React made managing components much easier and allowed for creating a simple UI, for testing. The original react framework had to be scrapped in favor of Blaze's templating. This made for a more simple integration in the rest of the application.

Next after a meeting with the product owner an implementation of message passing was created for accepting of messages from the calendar subsystem. The handling of messages was necessary for when a user created an event the goal list would then be populated with a new goal. The implementation of message passing was

also necessary to accept a list of premade goals from the settings subsystem per product owner request.

Lastly, aside from the standard CSS libraries, addition bootstrapping libraries were used in order to create better application synergy.

- Scrum Backlog (Product and Sprint - Link to Section 6

Coding

Approach

The approach to creating the goal tracker submodule was designed and implemented using object oriented design. The goal was for the submodule to be completely self contained and accept incoming calendar events and goals from various other subsystems, and store them into a completely separate Mongo DB database.

This approach allowed for the other developers to disregard anything that they may need to pass from their submodule to the goal tracker and ensure that it would be handled appropriately.

When clicking in the text form a user is able to type as much text as necessary to define a new goal. The user can then press the “commit to the Cure” button and the goal list will be populated with the new goal. The list is then sorted with the newest goal being put at the top of the list. A user then is able to click a checkbox to indicate that a goal has been completed and a line is put through the text in the goal for further readability. Lastly, the user may click the “X” beside the goal to delete the goal from the list.

Language

The primary languages used for the design and implementation of the goal tracker were HTML, CSS, and Javascript. ES6 was utilized for all the logic that the submodule was to actually do, with HTML for the interface. Finally, CSS standard, and bootstrap libraries were used to complete the final user interface and elements.

- User training
    - Training / User manual (needed for final report)
- Testing

7.2 Subsystem 5 Education (Nakava Kibunzi)

Initial design and model

The initial design of the education was to have a fully built and integrated education within the subsystem. It would contain normal features of education, such as dropdown menu that contain the buttons that link to the html pages. The subsystem would allow settings subsystem to change the background color of the pages. The subsystem would allow the user to read the contents of each page. The users can registrate to take the course online. They can view the contact information and other resources.

Data dictionary

Name	Data Type	Description
Text	varchar(800)	Text education contents

Refinements

The education module was designed to allow users to read information regarding health care. Meteor was used, it allowed for creating GUI for the main page menu.

The standard CSS libraries, addition bootstrapping libraries were used in order to create better application synergy.

Approach

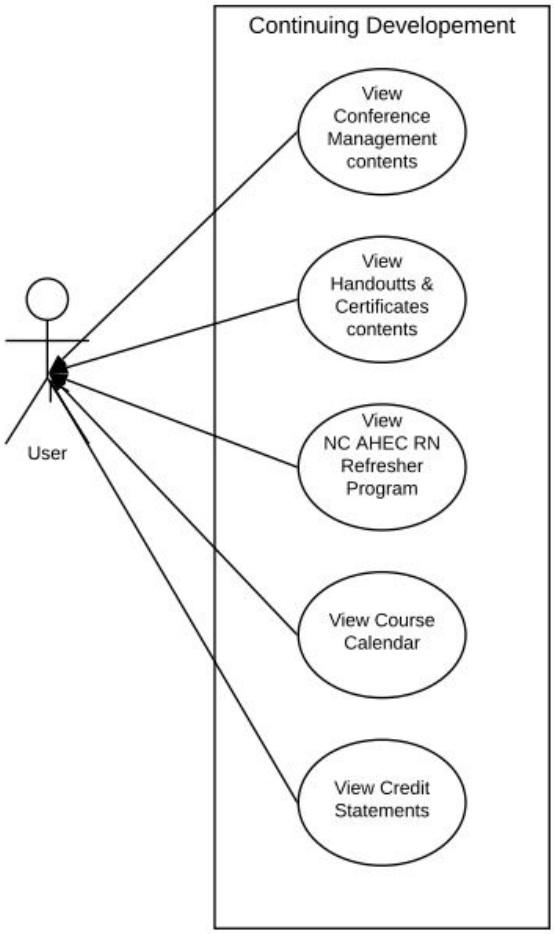
The education submodule was developed using an object oriented model of design. This subsystem was designed to be completely self contained and accept the setting alter the theme when the user wants a new theme.

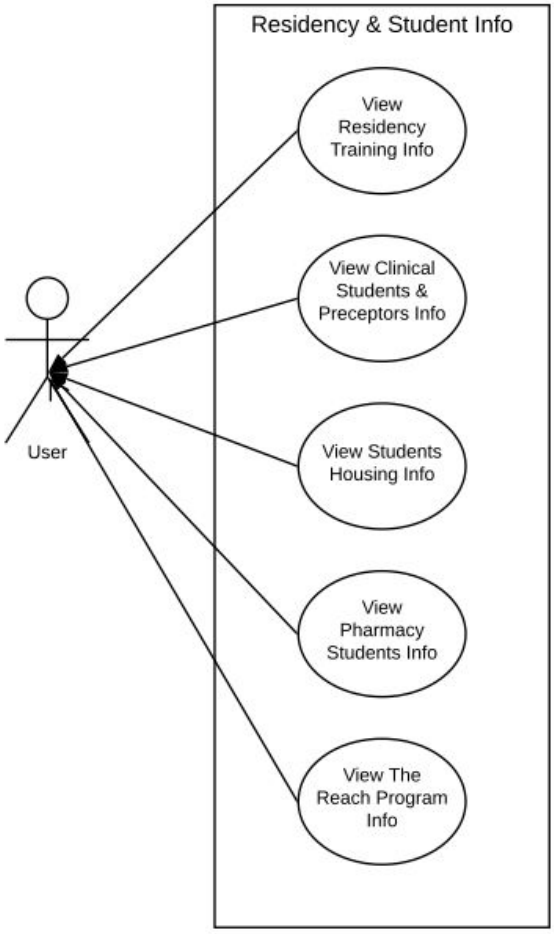
Language

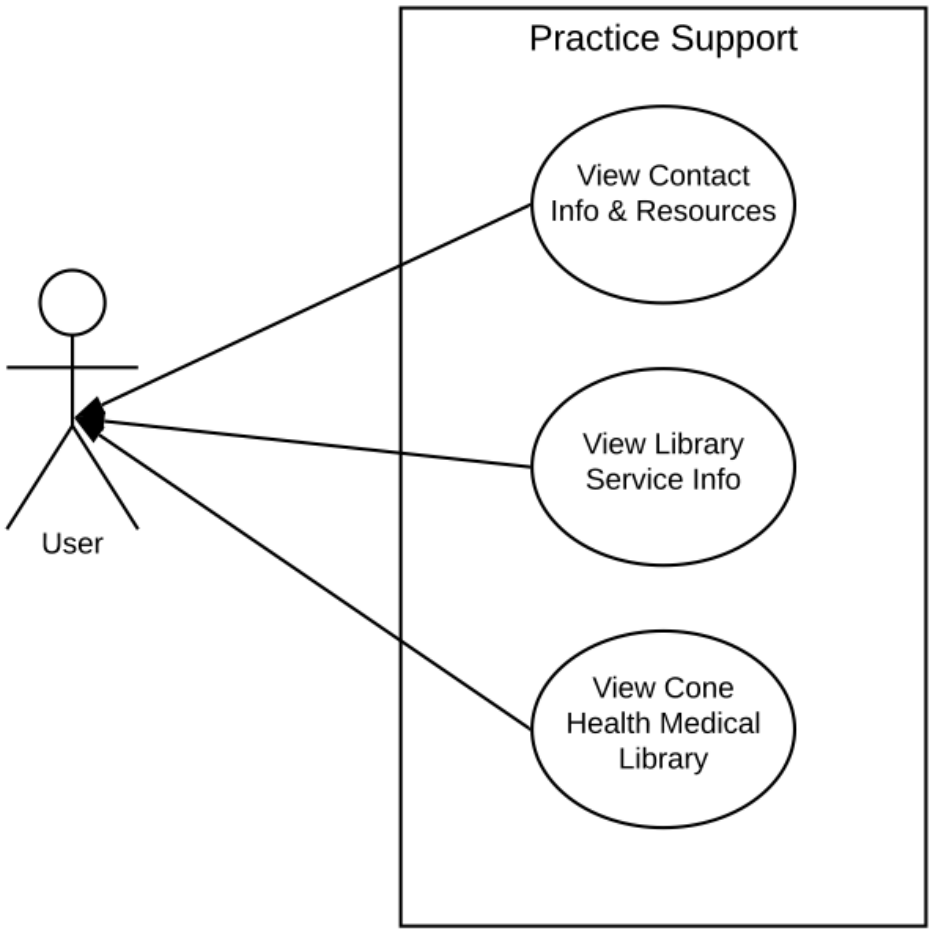
The languages used for the education were HTML, Javascript, and CSS. HTML was used for user interface design, Javascript was for the backend logic of the module,

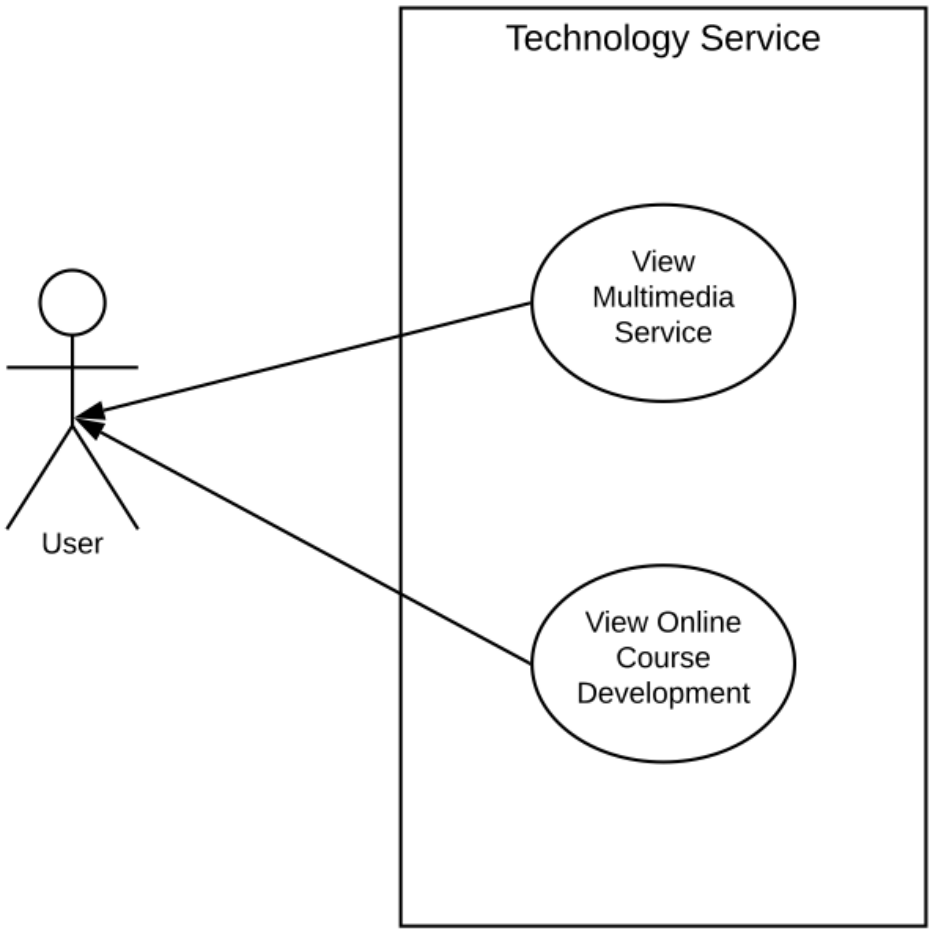
and CSS was used to aid the design of the user interface. In addition to standard CSS, the bootstrap CSS library was used to add additional functionality to UI elements and provide an all around cleaner look and feel.

- User training
    - Training / User manual (needed for final report)
- Testing




Residency & Student Info
View Residency Training Info
View Clinical Students & Preceptors Info
View Students Housing Info
View Pharmacy Students Info
View The Reach Program Info
User


Practice Support
View Contact Info & Resources
View Library Service Info
View Cone Health Medical Library
User


Technology Service
View Multimedia Service
View Online Course Development
User

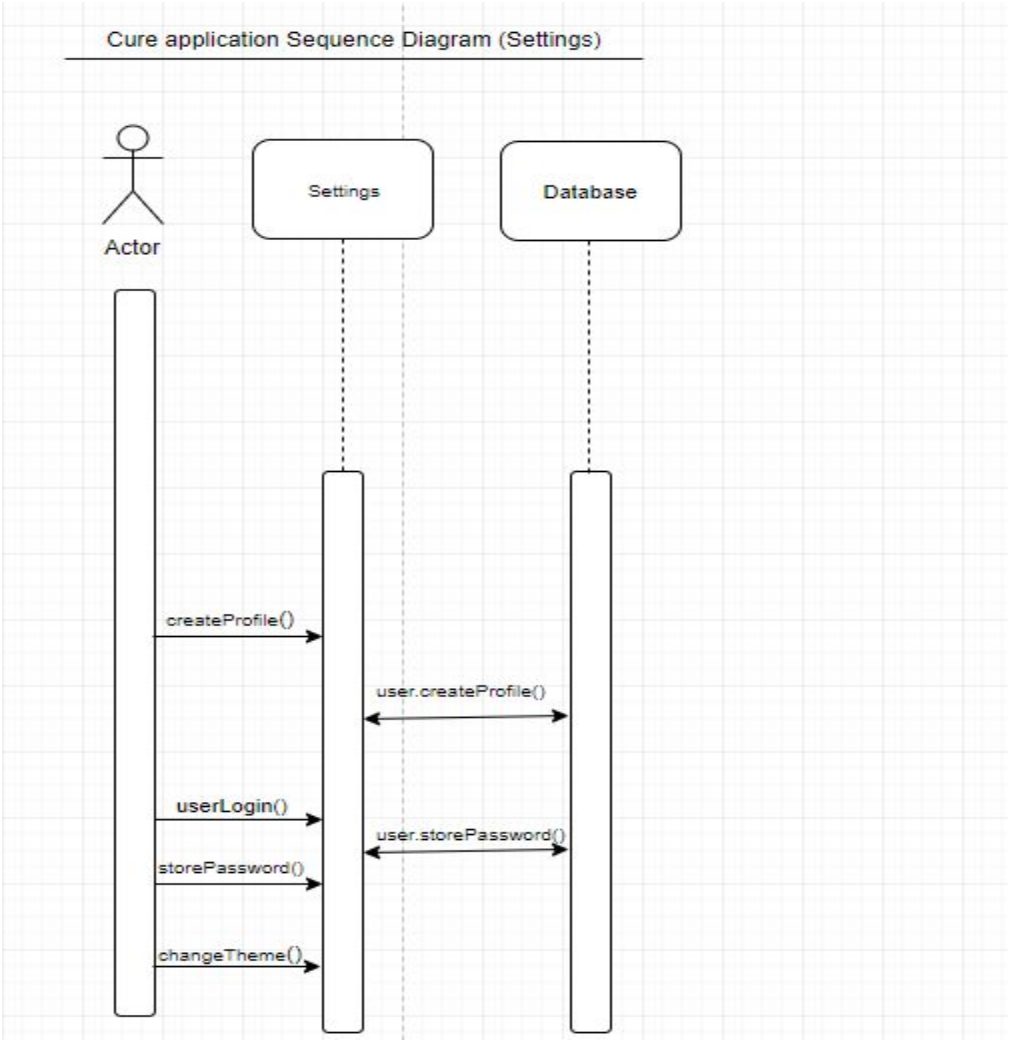
7.2 Subsystem 5 Settings(Charles S. Stamey)

Initial design and model

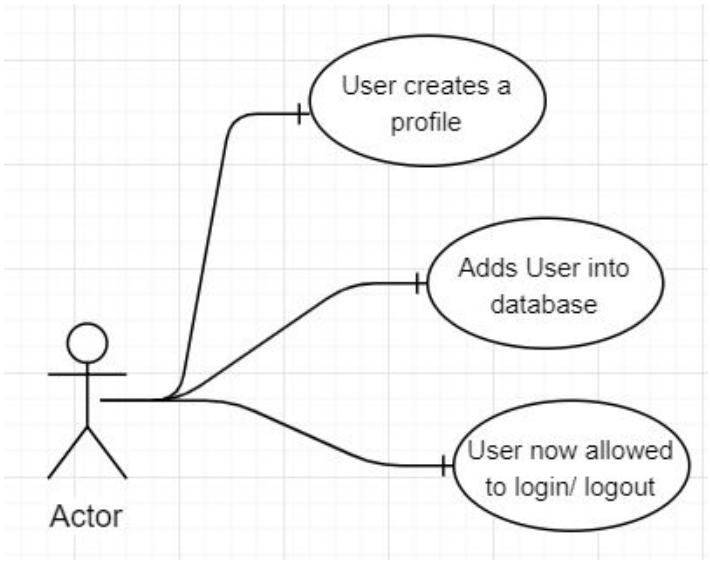
The settings module was designed to give users the ability to customize and tailor the app to their liking. With several themes to choose from, it was felt that the themes would do just that. Also, several built -in functions of Meteor was used. For example, passwords and account-ui were packages utilized for the modeling of user accounts.

UML for Settings

Settings
-Themes : events() -profile: events -password : events() -notes:String
+get listeners() : list +handleTheme(themes) +createProfile(user)


Cure application Sequence Diagram (Settings)
Actor
Settings
Database
createProfile()
user.createProfile()
userLogin()
user.storePassword()
storePassword()
changeTheme()

Use Case



Design Choices

Initially, the idea was that every app needs or has a setting subsystem that would be available to the user in-app. Later in the project some of the settings were not as necessary because they were handled by the device's settings system.However, we still felt as the user needed a way to customize the CURE app, create a profile, and be able to login, once the profile was created.

Some key choices that were made during the designing phase were to modify the look and feel of the GUI of the subsystem, utilize themes setting, and making the look of the GUI global across the entire app. The profile and login pages were consistent with this same design in that what was done for those needed to be made global and applied consistently across the app.

Data dictionary

Name	Data Type	Description
Theme	varchar(4)	Themes
Profile	varchar(n)	Profile creation
Login	varchar(1)	One login per device
Password	varchar(1)	Password format

Refinements

The settings module went from several settings to only one or two settings, back to several settings. The benefits were that this subsystem was simplified, since a lot of the account was streamlined and integrated within meteor.

Some liabilities of these refinements include disconnectivity between the application and and local database which tended to make logging in with the mobile app impossible with a dedicated server. While the subsystem is mostly self-contained, the fact that it now depends on obtaining a server for these purposes.

Final refinements included giving the ability to implement security options with user accounts and passwords. This is optional, but an implementation nevertheless.

Themes were added. A pink theme was added with the hope that in the future a ribbon can be used as the icon to represent honoring breast cancer awareness. Additionally, a blue ribbon theme in future updates to honor addiction and recovery awareness would be a good gesture.

Scrum Backlog - See Section 6 (pg. 30)

Coding

Approach

The settings submodule was developed using an object oriented model of design. This subsystem was designed to hold themes, user accounts, and passwords. Whenever a user wants to utilize the option and create a profile, the keyboard is used to input that. For the email placeholder, the user will simply enter their email as it needs to be entered and becomes their login credentials. The “Password” placeholder, the user clicks on the textfield, which then prompts user to Sign In or Register. It would then be stored in a database for future use.

Language

The languages used for the calendar were HTML, Javascript, and CSS. HTML was used for user interface design. Javascript was for the backend logic of the module, and CSS was used to aid the design of the user interface. In addition to standard CSS, the bootstrap CSS library was used to add additional functionality to UI elements and provide an all around cleaner look and feel.

User Training (See page 68) *for User Manual*

Testing

For the Settings subsystem there wasn’t an opportunity for legitimate user testing, however each of the developers regularly tested the subsystem. We did discover that there needed to be an outside database implemented for user accounts, as Meteor only handled web-based accounts.

User Manual

Presented by
Members of Senior Capstone
University of North Carolina Greensboro
Computer Science Department

Table of Contents

Page #

1.1 System Overview -The CURE App aims to serve survivors of the opioid crisis with mostly local, privacy protecting, easy-to-use tools.

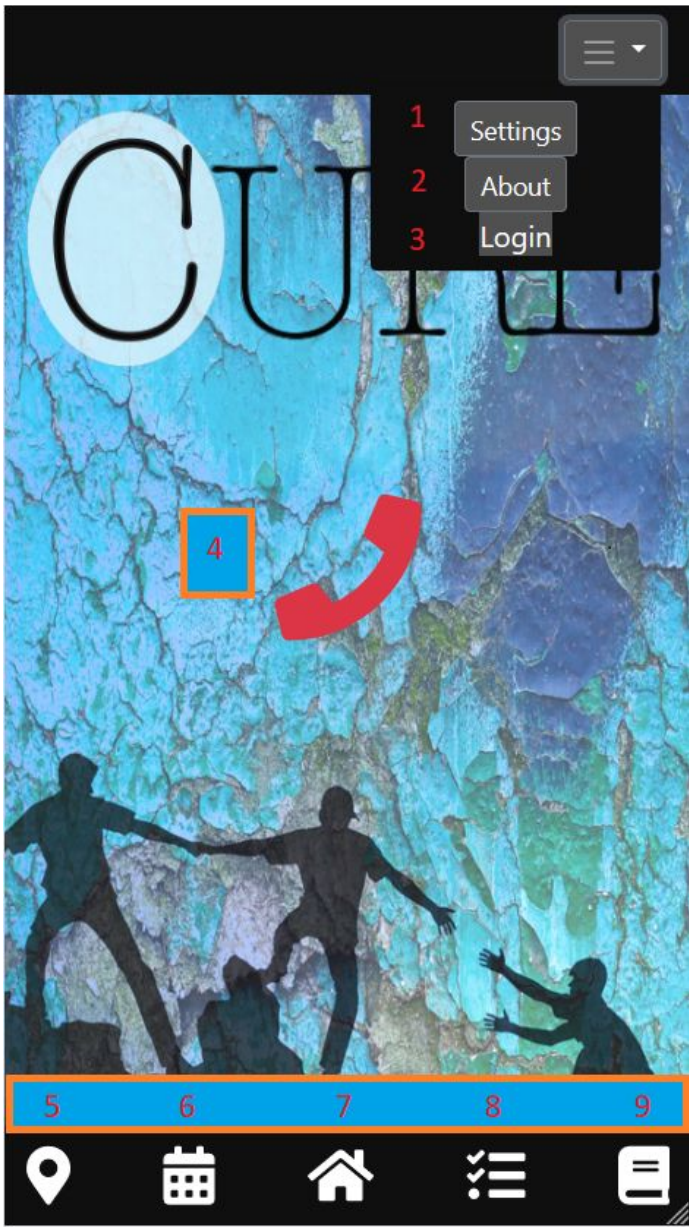
1.2 Project References

References that were used in preparation of this document in order of importance to the end user.

https://guide.meteor.com/accounts.html (*page 10*)

2.1 Getting Started

Main Screen



From the main screen a user can click any of the nine buttons available.

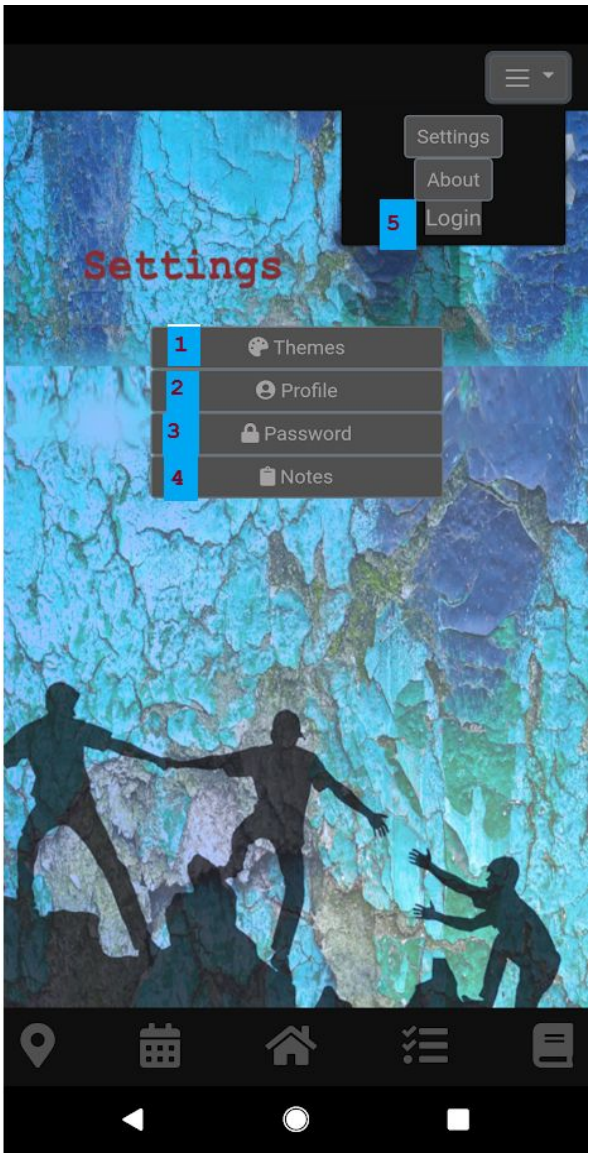
Starting from the top and going down each of the buttons travels to the following pages:

1. Settings
2. About CURE
3. Login Account
4. Hot Line (one dial)
5. Services/Location Map
6. Calendar Events
7. Home (this page)
8. Goal Tracker
9. Education and Information

The main screen is where the application will always start. The navigation bars at the top and bottom of the screen

are available from anywhere in the application. The function unique to the home screen is the Hot Line (4) button. This will open the Hot Line menu which allows users to quickly contact emergency help.

①. Settings

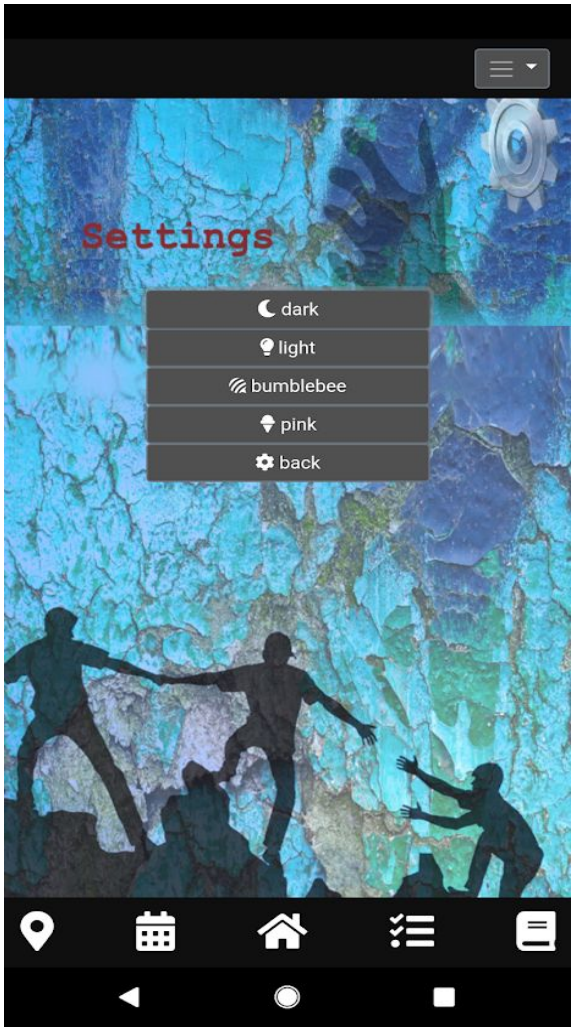


① Themes
② Profile
③ Password
④ Notes
⑤ Login

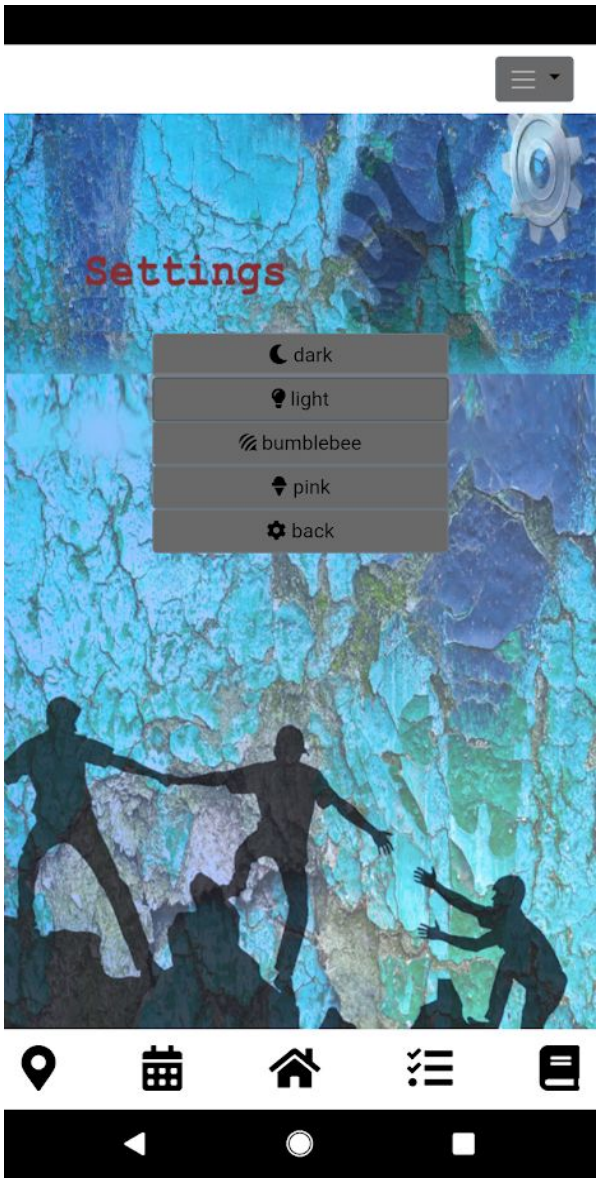
The main setting screen begins with allowing users to customize the app with Themes(1)! Included are four default themes.

① Themes

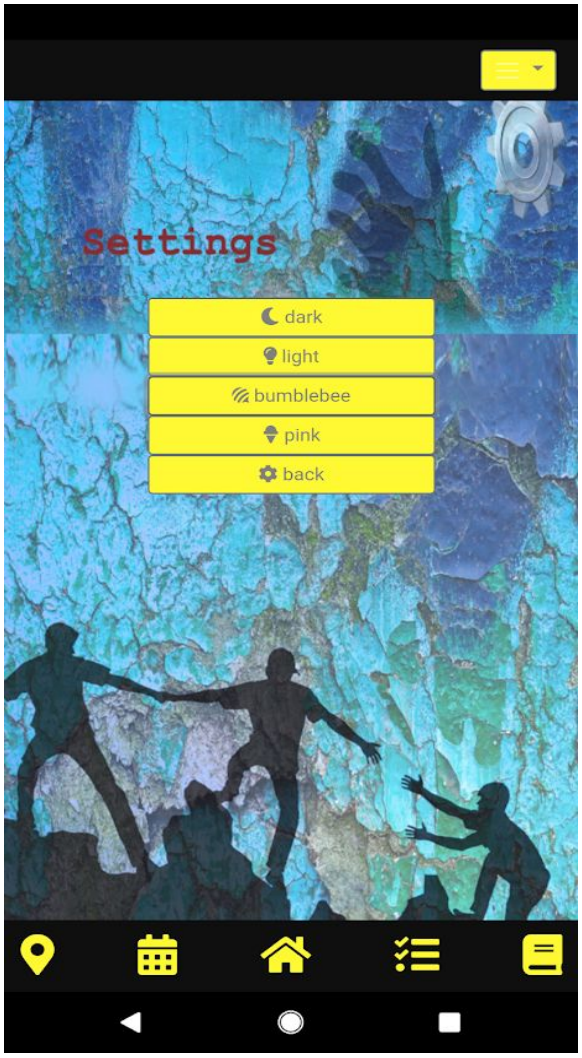
i. Dark



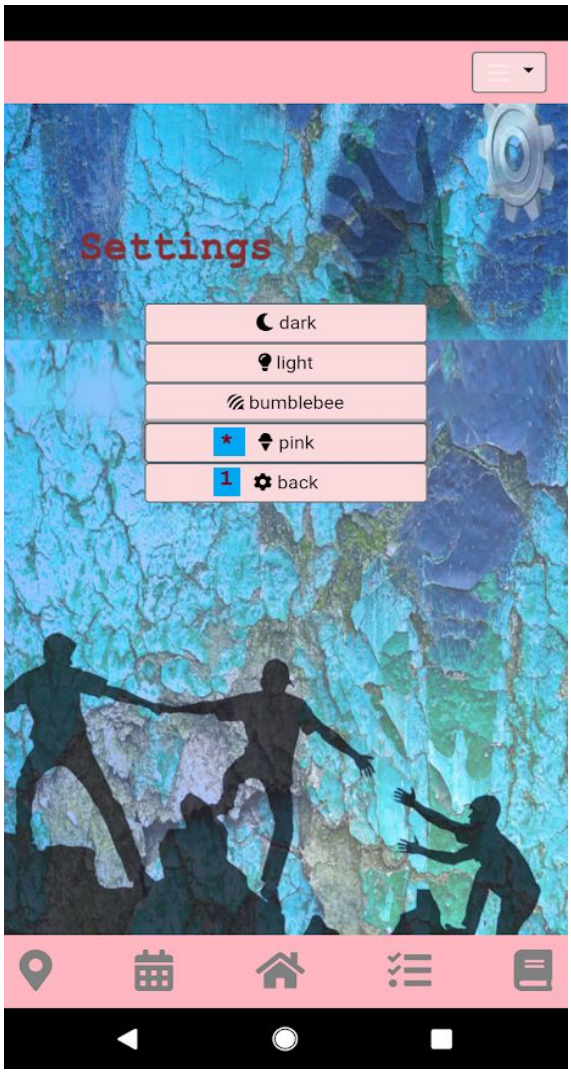
ii. Light



iii. Bumblebee



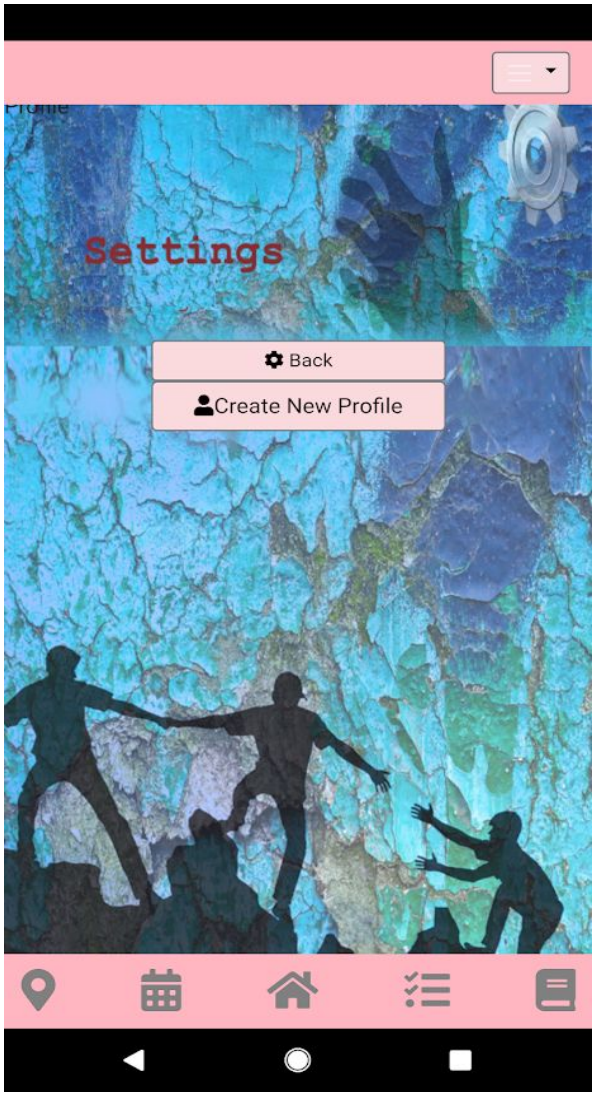
iv. Pink



* Optional pink theme - A ribbon can be in place of the ice cream cone, honoring breast cancer awareness. Also, a blue ribbon - theme to honor recovery and addiction awareness seems like an idea for the future!

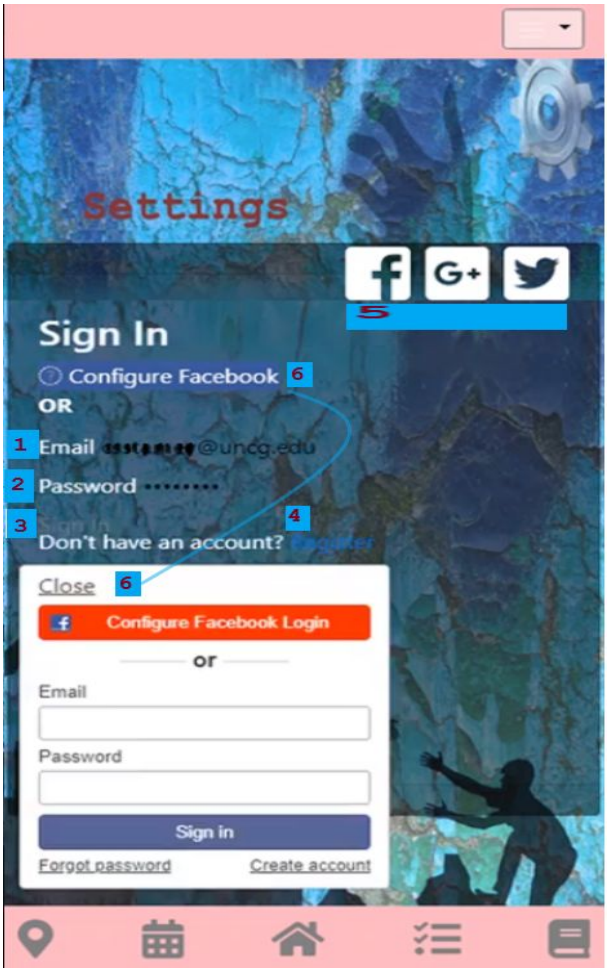
1. Most settings (gear icon) pages include a back button to return to the main settings page. Exception being the profile page; discussed later.

② Profile



This is not a requirement to use the app. This optional feature gives users the ability to create a profile.

a. Create New Profile / Login

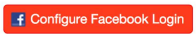


1. **Email Placeholder**
2. **Password Placeholder**
3. **Click Sign In Button**
4. **Or If no profile, Register new**
5. **Social Icons**
6. **Configure Facebook Login**

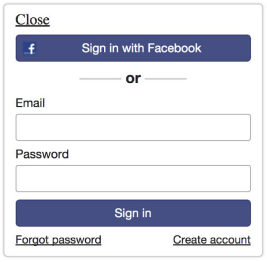
Login page is made available once a profile is created or a user chooses to login from a social network platform.
(5&6) will need app set up; below for Facebook, *please refer to link* here

accounts-ui with one login service, when correctly set up

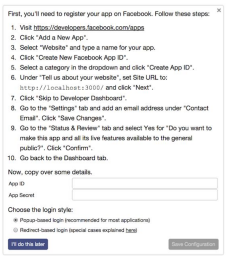
button to set up login service



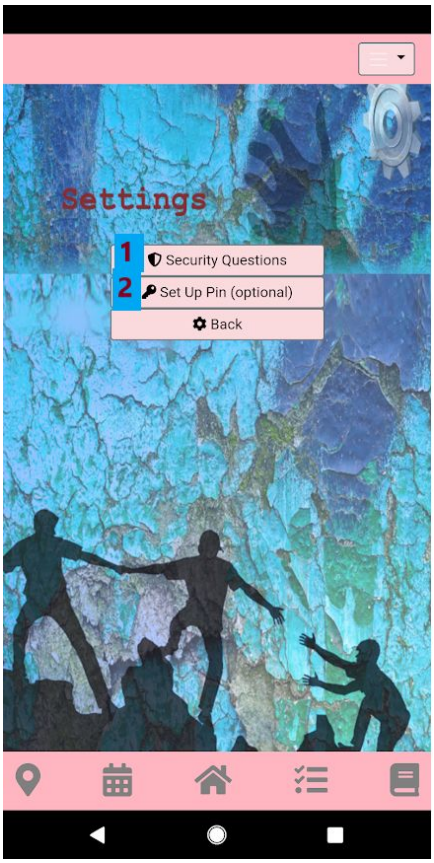
accounts-ui dropdown with multiple login services



popup with directions to set up login service



❸ Password

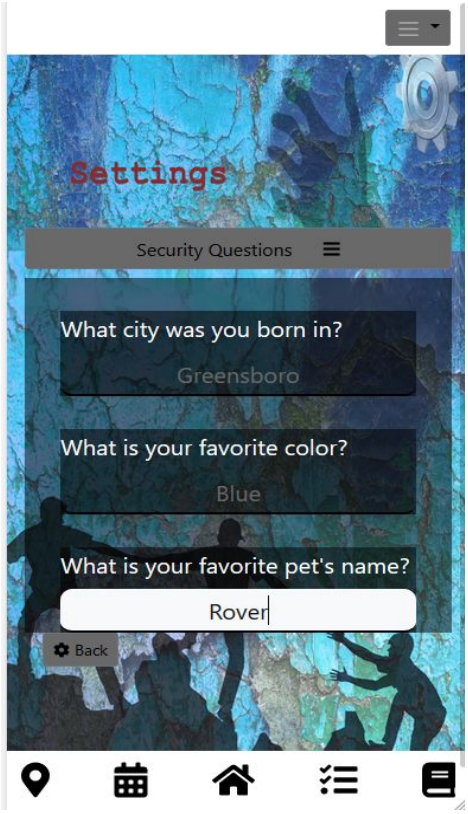


1. **Security Questions**
2. **Set Up Pin**

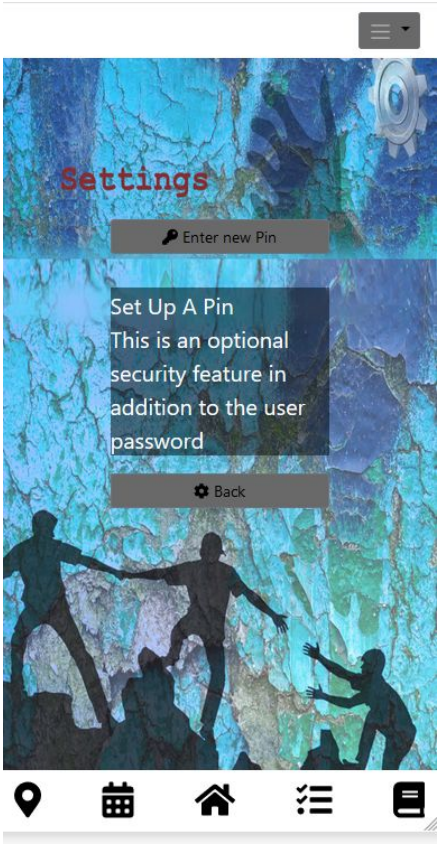
User can add security questions. To further protect account, a pin(2) can be set up for a 2-step verification. These are optional and not required for user accounts. (below)

While not fully-functional, these are used to display the extent of user accounts.

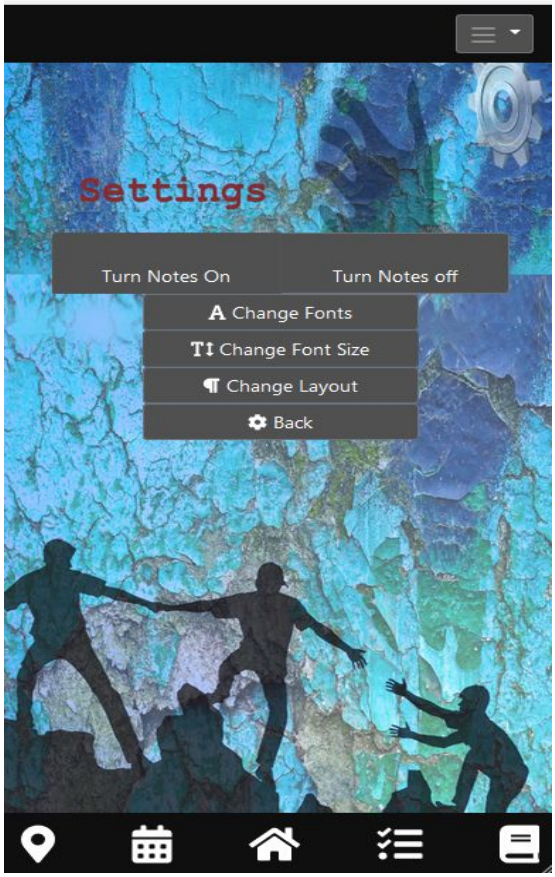
1. **Security Questions**



2. Pin



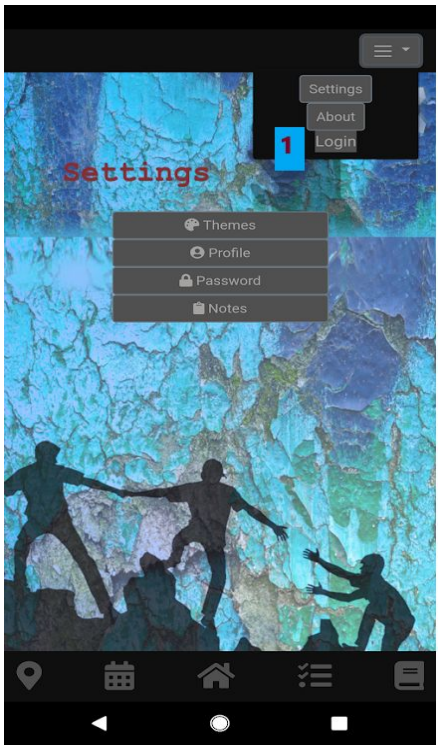
❹ Notes

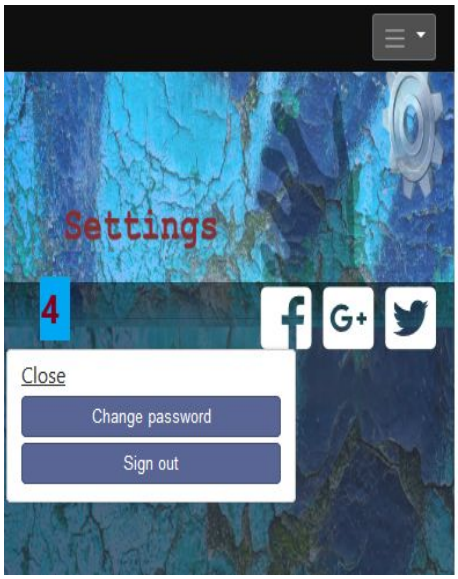


1. Turn notes on or off
2. Change Font
3. Change Font Size
4. Change Layout

Optional layout module for possible future Journal/ Log feature. If the journal/ log feature is implemented, this module would give users the ability to customize their journal/log.

⑤ Login

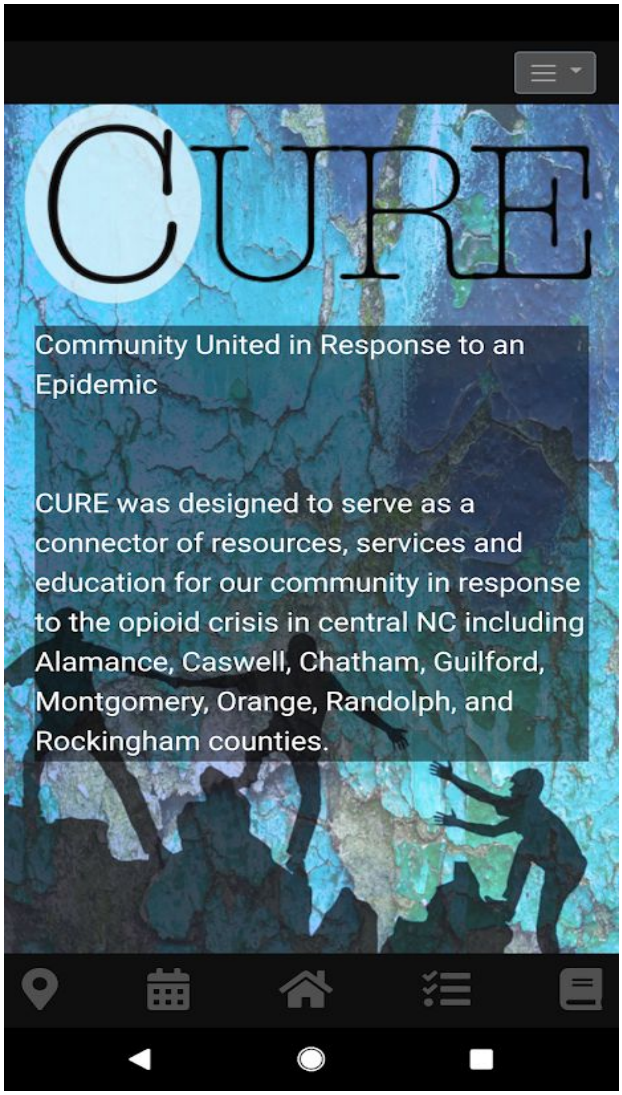




Once user creates profile and precedes to login(1), the login button is replaced in the dropdown menu with a logout button(2), which is accessible on any screen. The users' email is now visible(3) on that login page with its own dropdown(4). This gives the user options to change password or sign out.

2. About

Gives a brief synopsis of what the CURE app is about.



4. Hot Line

Hot Line allows users a quick and easy way to call emergency services for opioid addiction.

There are some basic functions available to the user, as follows:

1. Emergency Contacts

The three top buttons each call an emergency line in North Carolina. Simply touch one of the buttons to call them.

2. Add Contact

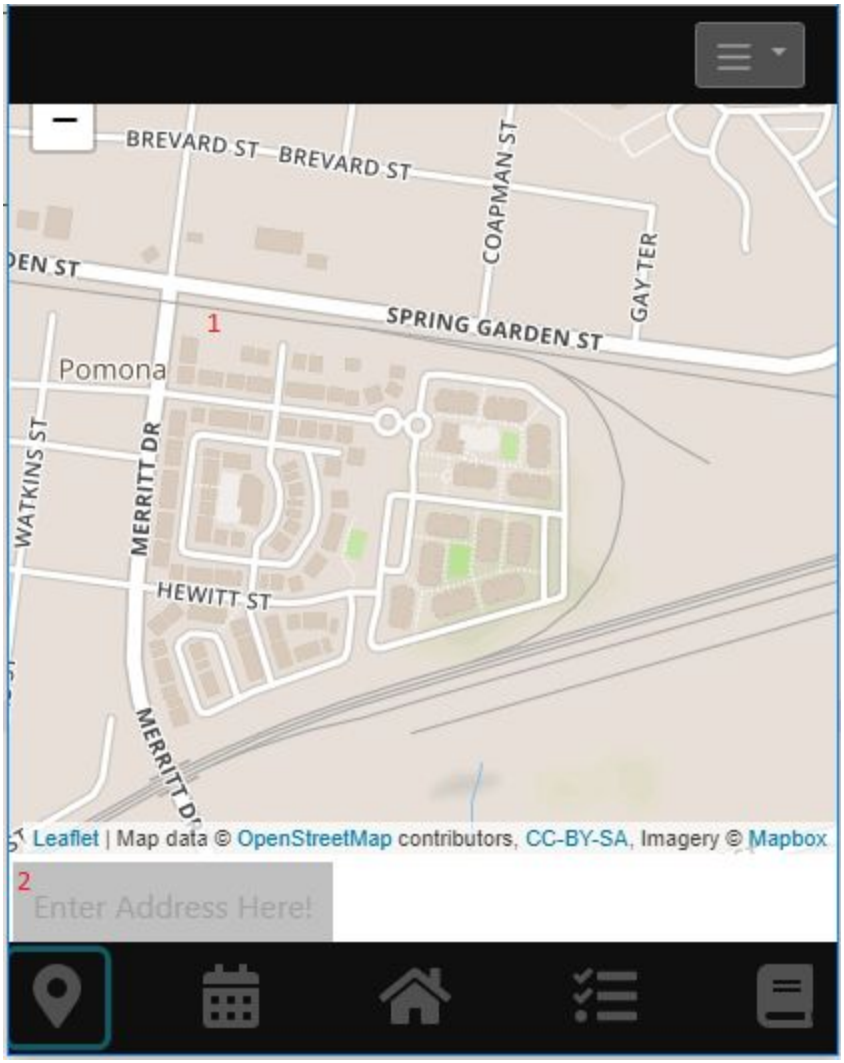
Touching the “+” button allows users to add a new emergency contact.

3. Delete Contact

Touching the “-” button allows users to remove an existing user-made emergency contact. The default numbers cannot be removed.

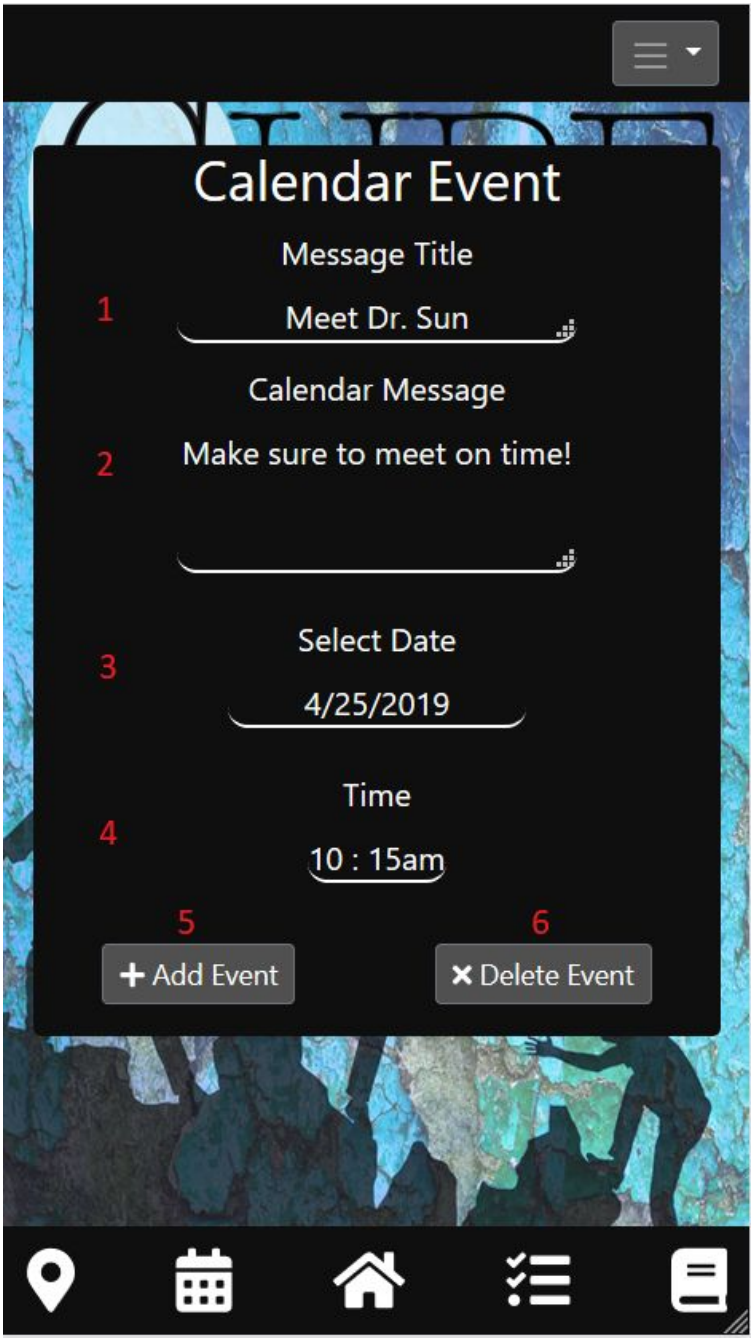
5. Map Subsystem - Alpha

1. Map Interface
    a. Click Events
    Users can click on the map and choose to register a location.
    b. Traversal
    Users can drag their fingers/mouse across the map and look at different locations.
2. Address Event Bar
    a. Users can click on this button to enter an address. Upon entering, the map will jump to that general location.



6. Calendar Events

The calendar menu allows users to add events to their native calendar so that notifications will be updated to the phone without needing to run the application in the background. Calendar Events are updated in real time and require a specific time be set to send to the calendar.



The following functions are:

1. Message Title

Enter a title for you message.

2. Calendar Message

An extra note that will be appended to the calendar event.

3. Select Date

Selecting the date text box opens a mini calendar which allows the user to choose a date. Doing so auto-populates the date into the text field.

4. Time

Choose the time for which this event will remind you. Touching the text field will open a pop-up which allows the user to choose a time.

5. Add Event

Once all appropriate fields are filled, this adds the event to the native calendar.

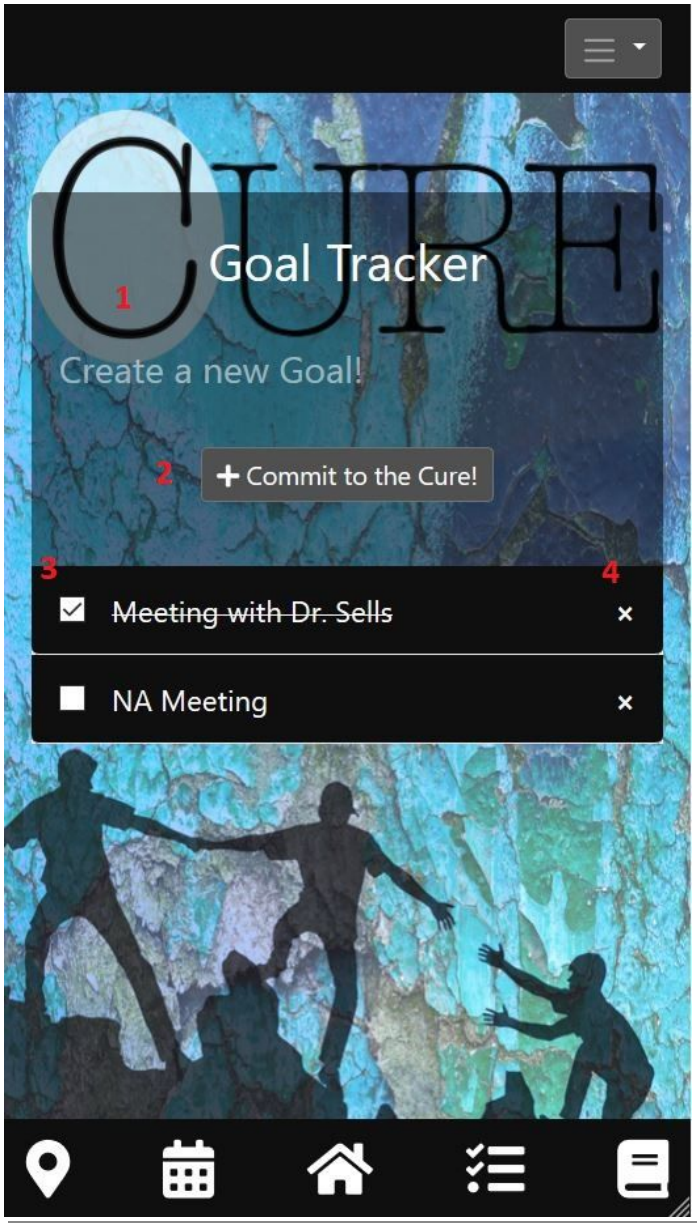
6. Delete Event

Once clicked, allows the user to delete an existing event stored by the application.

Any event that is stored will also be updated (with the title of the calendar event) to the user’s goals in the goal tracker page.

8. Goal Tracker

The goal tracker allows users to create a list of goals that they wish to complete. This is to provide a sense of privacy to the user allowing them to have their recovery goals all contained within the CURE application.The goals can be updated as they are completed as well as removed.The goal tracker also can have goals pre populated from the calendar application.



The following functions are:

1. Create a new goal
Enter the text of you new goal

2. Commit to the Cure!
Add the new goal to the list

3. Update Box
Update a completed goal giving the text a strike through

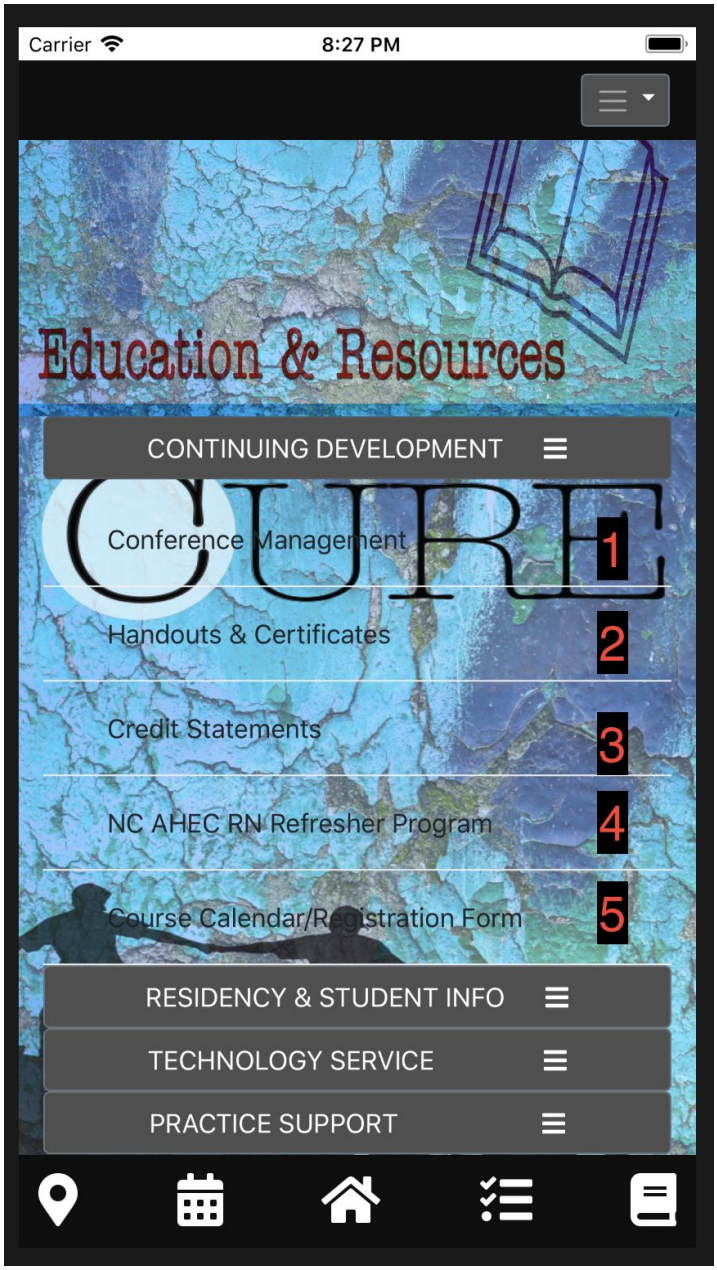
4. X button.
Delete the particular goal.

3.6. Education & Resources

The education allows the users to get access to resources and tools that help them to stay on track and learn about their community. It provides information about health care centers, online course, and certificate.

The following function are:

1. Continuing Development
2. Residency & Student Info
3. Technology Service
4. Practice Support

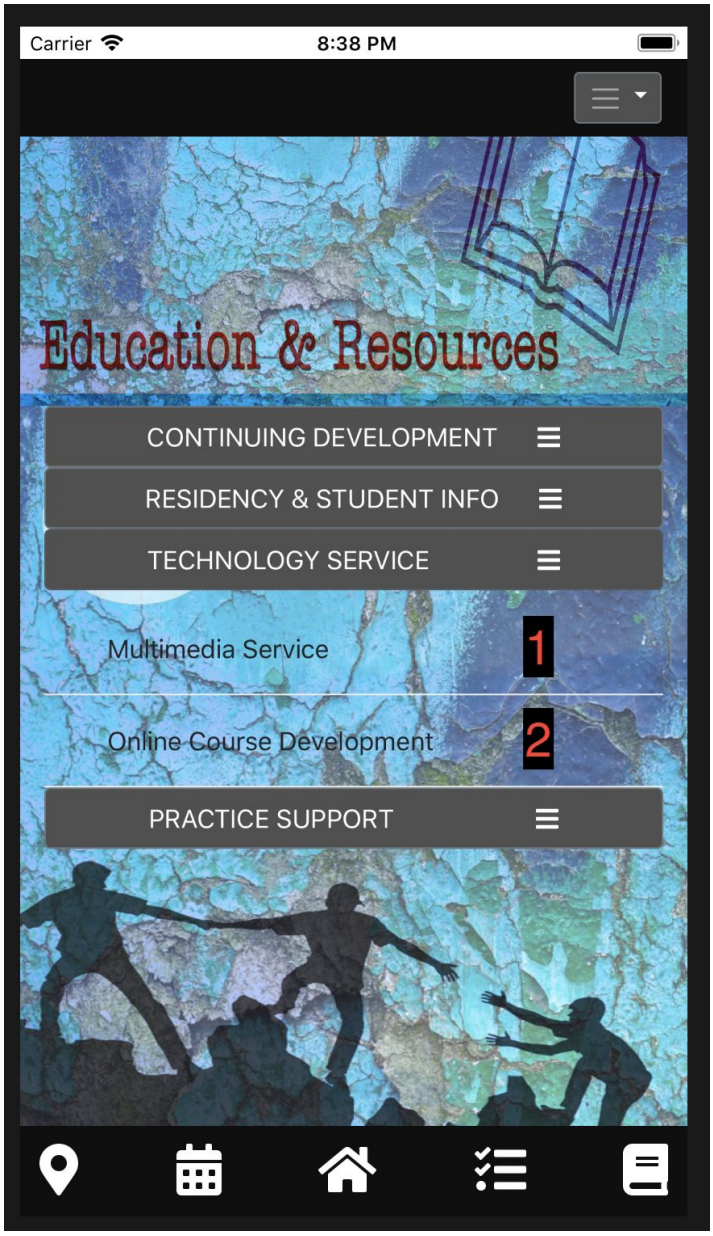


3.6.1 Education & Resources

1. Conference Management
2. Handouts & Certificates
3. Credit Statement
4. NC AHEC RN Refresher Program
5. Course Calendar/Registration Form

3.6.2 Residency & Student Info

1. Residency Training
2. Clinical Students & Preceptors
3. Student Housing /ORPCE
4. Pharmacy Students
5. The Reach Program



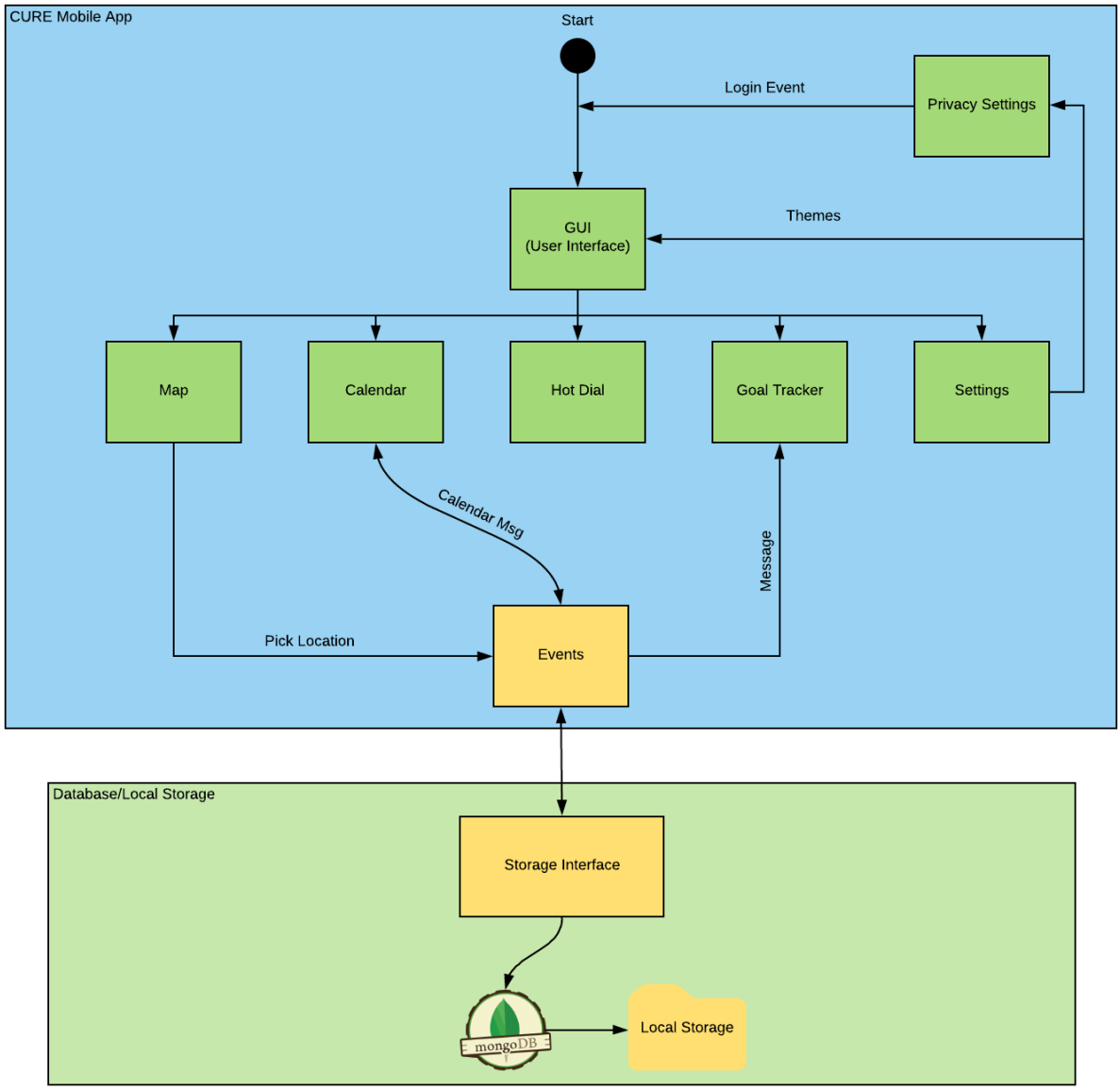
3.6.3 Technology Service

1. Multimedia Service
2. Online Course Development

3.6.4 Practice Support

1. Contact Information & Resources
2. Library Service
3. Cone Health Medical Library

A. APPENDIX - Menu Flow



Source Code

See source code at our github:
https://github.com/stevieclean/UNCG-CSE-Senior_Proj

Team Members

Chauncey Davidson - https://github.com/cldavids2015
Brian Goughnor - https://github.com/bgoughn
Joshua Brown - https://github.com/JoshuaBr7
Charles Stamey - https://github.com/stevieclean
Nakava Kibunzi - https://github.com/Gnakava