Life Tracker Application

Software Requirements Specification

Draft 1

[September 23, 2019]

Team Members: Martin Fritz, Huy Le, Jonathan Wallis, Jacob Roquemore

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide a general description, including design constrains, the functional requirements of the document, the multiple interfaces of this application, performance requirements, and user characteristics of this project.

1.2 Scope

This application will be an android application that assists the user with recurring reminders in their lives. It will be based in Android and run on android phones. There will not be cross-compatibility for this application at this moment. Users will be able to access their data remote from the cloud from any application and their data will be stored on a server. Users will be able to create, edit and delete "reminders" that are either daily habit builders or long-term reminders.

1.3 Definitions, Acronyms, and Abbreviations

Users - Customer or primary user of our application.

App - Mobile application

Cloud - Online server

HTTP/s – The HTTP or HTTPS protocol; An application layer networking protocol

Google Maps – An application from Google that will display geographical map information, as well as navigation information [3]

Google Calendar – An application from Google that can store time-based events online and offline [4]

Android Studio – Official IDE for Android Development [6]

API – Application Programming Interface

TCP - Transmission Control Protocol; A transport layer networking protocol

1.4 References

[1] IEEE, *IEEE Recommended Practice for Software Requirements Specifications*. 1998.

[2]Draw.io. (2019). *Flowchart Maker & Online Diagram Software*. [online] Available at: https://www.draw.io/ [Accessed 24 Sep. 2019].

[3]"About – Google Maps", *Google.com*, 2019. [Online]. Available: https://www.google.com/maps/about/#!/. [Accessed: 24- Sep- 2019].

[4]"Google Calendar: Free Calendar App for Personal Use", *Google.com*, 2019. [Online]. Available: https://www.google.com/calendar/about/. [Accessed: 24- Sep- 2019].

[5]"Flowchart Maker & Online Diagram Software", *Draw.io*, 2019. [Online]. Available: https://www.draw.io/. [Accessed: 24- Sep- 2019].

[6]"Meet Android Studio | Android Developers", *Android Developers*, 2019. [Online]. Available: https://developer.android.com/studio/intro. [Accessed: 24- Sep- 2019].

1.5 Overview

In the next chapter the document dives into the overall description of the project itself. It looks at the interfaces, product functions, user characteristics, constrains, and assumptions and dependencies for the app. Chapter three analyzes the specific requirements of the app, including the external interfaces, functional requirements, performance requirements, design constrains and other requirement constraints for the app.

2 Overall Description

2.1 Product Perspective

Our LifeTracker product is a new, independent piece of software created by a small team. The LifeTracker product relies on a variety of preexisting pieces of software, systems, and hardware. For our mobile app, it will require a modern 64-bit Android phone with internet connectivity and internal storage. Internet connectivity is needed for users to retrieve and store task-based data, as well as demographic data, on an external web server and database. Internet connectivity is also required for calling third-party apps such as Google Calendar or Google Maps. Additionally, GPS connectivity is needed for Google Map interactivity. Lastly, access to an Android phone's internal storage is needed for storing user data locally on an individual's phone for offline access. A broad diagram of the connectivity between our piece of software and sub-components, as well as external systems, can be seen in Figure 2.1 below:

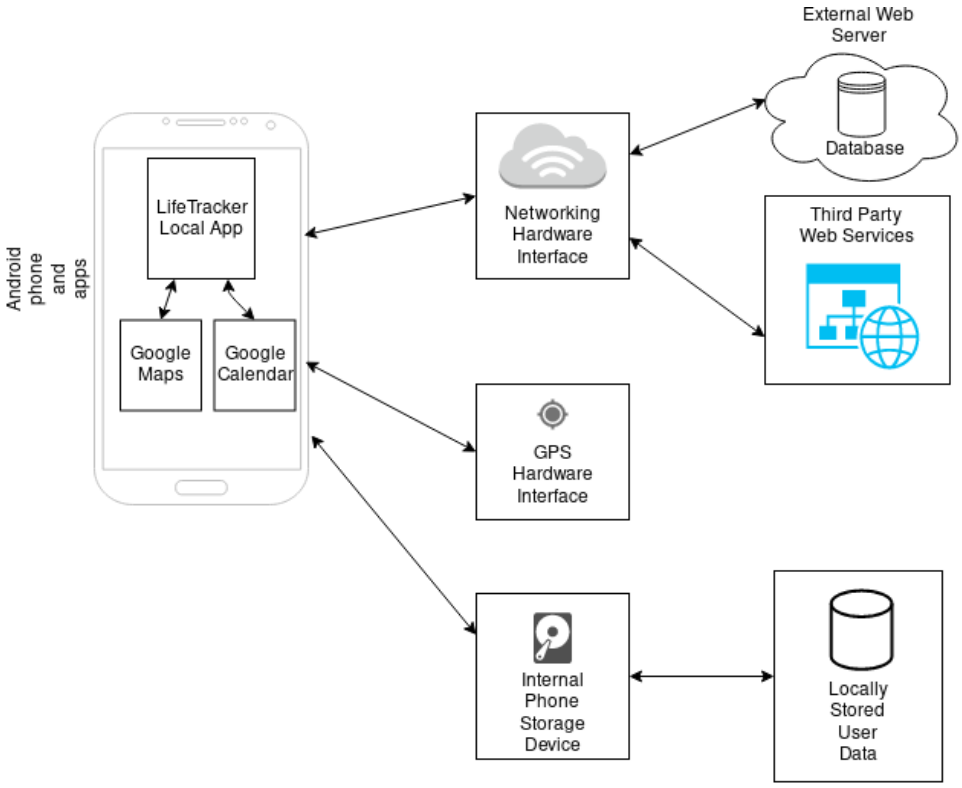

Figure 2-1 Broad diagram of system and component connectivity

Source: (Draw.io, 2019)

2.1.1 System Interfaces

The following is a list of systems that our application will rely on and interact with:

- External Web Server
  - Will provide user data in JavaScript Object Notation (JSON) format
  - Also, will store user profile data and registration data in a MySQL database as a part of the LAMP (Linux, Apache, MySQL, PHP) stack
  - Has the capability to provide web server interaction through PHP or other web server interactive languages
  - Can function as a general HTTPS server through Apache
- External Third-Party Services through APIs or URLs
  - Makes calls to both Google Maps API and Google Calendars API
  - Can make calls to other services either through simple URL requests in Android Custom Tab, which is essentially a browser visually embedded in the application, or through third party service API requests

2.1.2 User Interfaces

The LifeTracker software has primarily been designed around a portrait orientation mode, although it will have the ability to scale with landscape mode as well. There are five primary screen layouts the user will see depending on context: registration and login, tasks for the day, weekly progress and tasks, missed tasks and their frequency, and the user's profile.

Upon opening the app for the first time after installation/reinstallation, or if a user has deleted the application's cache and data, the user will be shown a login screen, as can be seen in Figure 2-2 below. The user then has the option of logging into a preexisting account using either an email/username as well as their password if they have one. Otherwise, if the user does not have a preexisting account or needs a new account, then they can select the Sign-Up button if they are a new user. If this button is selected, then the user will see the registration screen, as seen in Figure 2-3 below. This screen will allow a user to enter an email, password, and username to create a new account, with the option to enter additional details later.

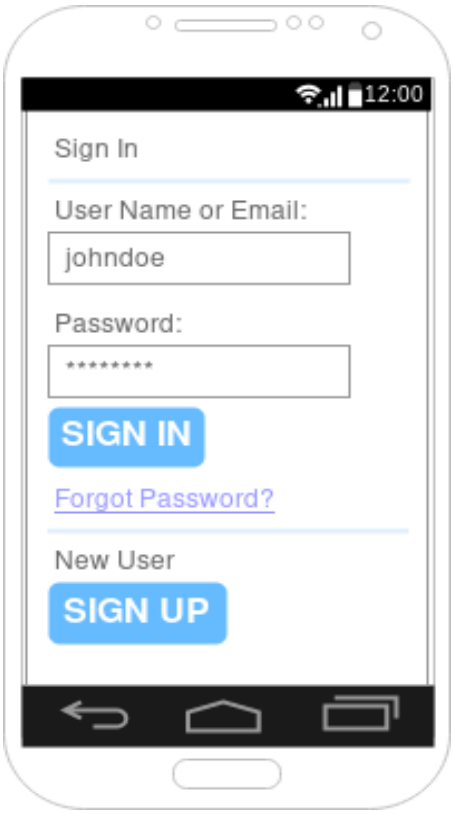

Figure 2-2 Login screen

Source: (Draw.io, 2019)

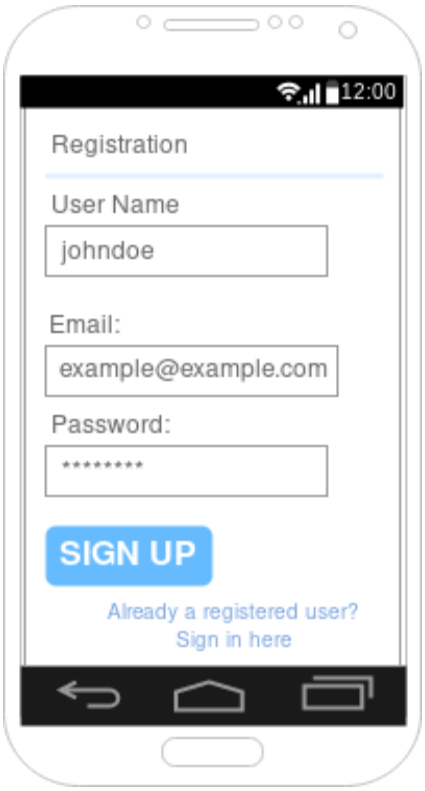

Figure 2-3 Registration screen

Source: (Draw.io, 2019)

Once a user has logged in, then their user authentication information should remain on their specific device until either they delete the apps data/cache, or they delete the app itself. Upon a user's repeated opening of the app, they will start at the daily task screen, as can be seen below in Figure 2-4. This screen will allow a user to view the tasks they have to accomplish today, as well as a progress bar showing their progress for the day. The user can indicate they have finished a task by checking it off the list using the check mark buttons to the left of the task. Additionally a user can tap on some tasks to pull up information such as location of an event on Google Maps, or places to buy an item that is needed that day through third-party APIs such as Amazon.

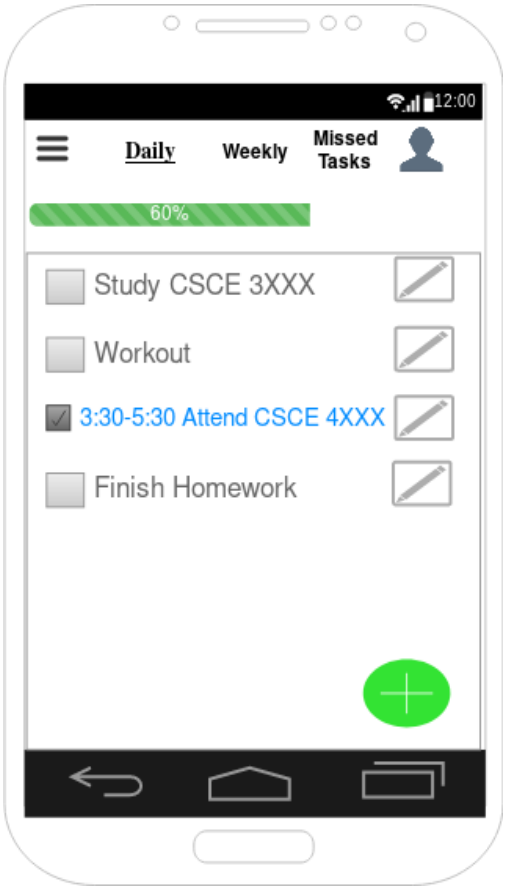

Figure 2-4 Daily tasks screen

Source: (Draw.io, 2019)

Another screen which the user can interact with is the weekly tasks overview, as can be seen in Figure 2-5 below. The screen can be accessed by simply selecting it's text, "Weekly", at the top center of the screen. This screen can show task completion progress of the days in the current week, as well as tasks in previous weeks. By default, the progress of the current week will be shown to the user. If the user wants to look at task completion summaries from previous weeks (or further weeks, depending on the currently selected week), then they can do this by selecting the down arrow in the top left of the screen next to the currently displayed week. By clicking on Tasks, it will show them the specific tasks they marked as completed for that week. An example of a previous week can be seen in Figure 2-6 below.

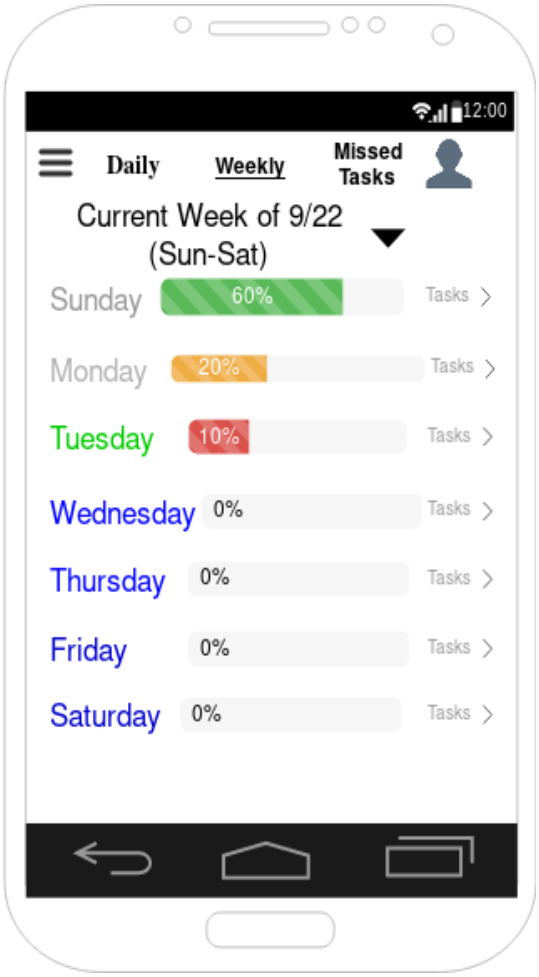

Figure 2-5 Weekly progress page (current)

Source: (Draw.io, 2019)

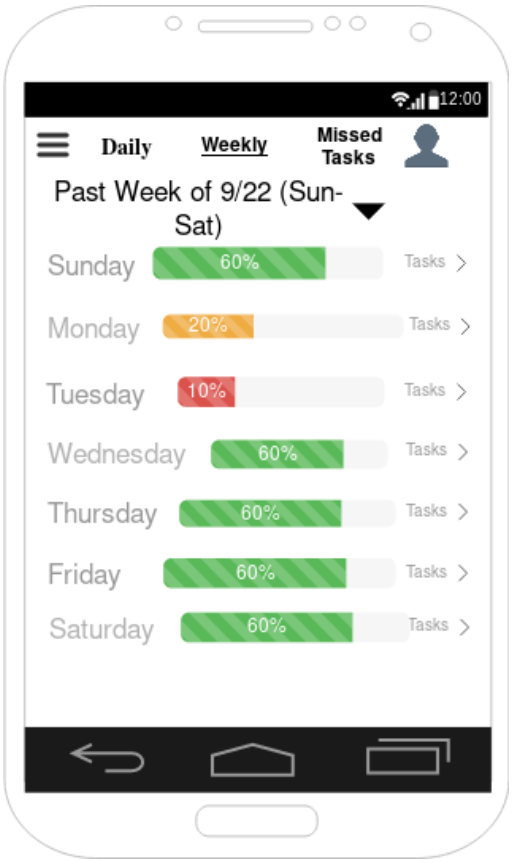

Figure 2-6 Weekly progress page (past)

Source: (Draw.io, 2019)

Another task focused screen is the missed task section, as can be seen in Figure 2-7 below. This page allows a user to view overall statistics as to their most frequently missed tasks as well as the frequency of each one. There is an option to scroll through the tasks, as indicated by the scroll bar on the right side of the screen. When a user taps details, it will show them more details and analytical data about that specific task.

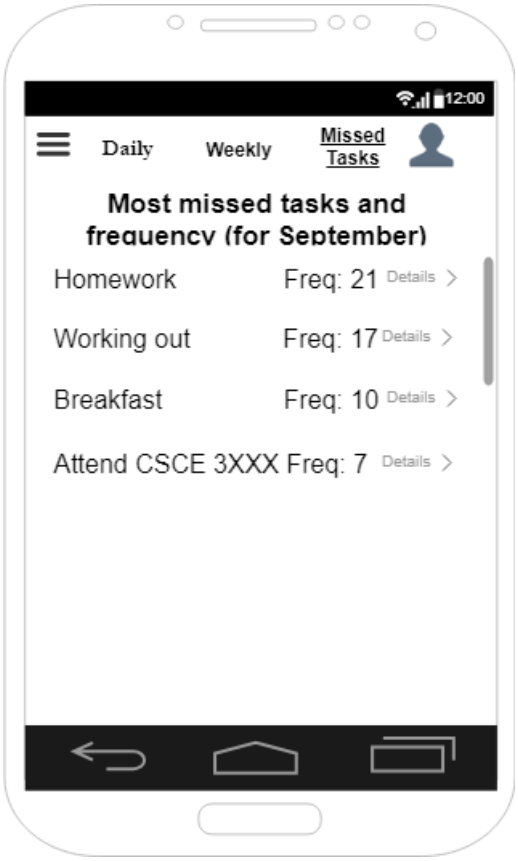

Figure 2-7 Missed Tasks Page

Source: (Draw.io, 2019)

The last task specific pages are the pages brought up when a user adds a task (by pressing the plus icon in the Daily screen as can be seen in Figure 2-4 above) and edits a task (by pressing the pencil icon next to a task in the daily tasks screen as seen in Figure 2-4 above.) These pages are similar in appearance, but one page allows a user to enter new information related to a task such as its descriptive title and it's type, while the other allows a user to edit pre-existing tasks. The task addition page can be seen in Figure 2-8 below, while the task editing page can be seen in Figure 2-9 below.

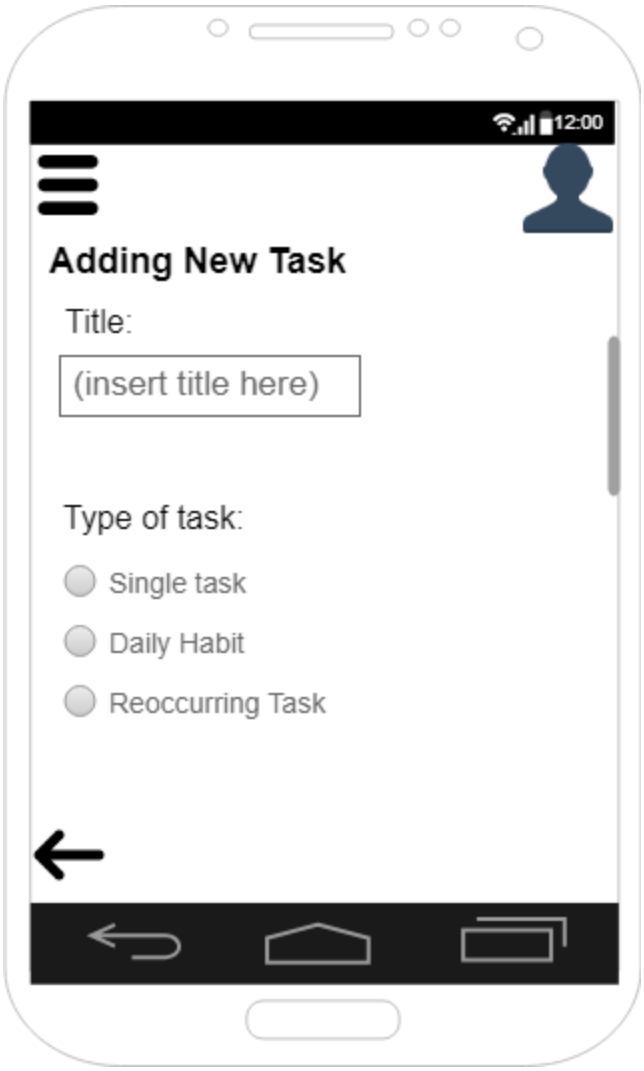

Figure 2-8 Task addition page

Source: (Draw.io, 2019)

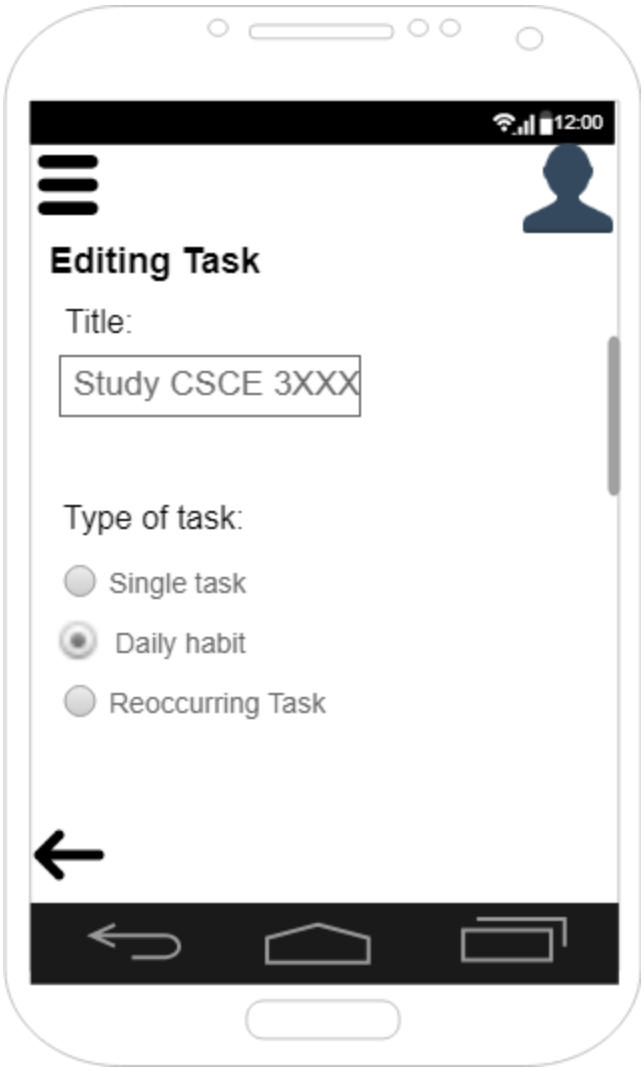

Figure 2-9 Task editing page

Source: (Draw.io, 2019)

The last overall important application page is the user profile page which can be seen in Figure 2-10 below. This page will allow a user to instantaneously view their profile picture, as well as edit it or add a new one by pressing the pencil edit icon to the bottom right of the picture. This page also allows a user to view their profile information, which can be scrolled through as indicated by the scrollbar to the right of the screen. Lastly, the user can select the Edit Account Information option at the bottom of the screen to arrive at a screen in which the user can edit their profile data such as their name, username, and other demographic information.

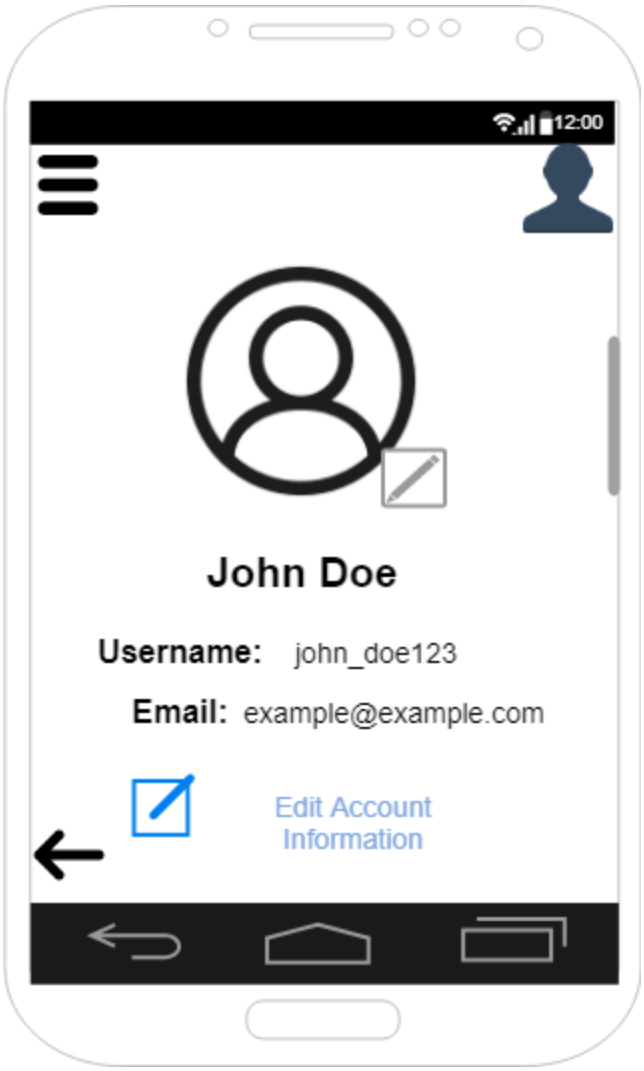

Figure 2-10 Profile Information Screen

Source: (Draw.io, 2019)

In addition to these screens, there is a drop down menu which can be accessed in each screen by tapping on a the box in the upper right hand corner of the screen with the user's profile picture, as can be seen in Figure 2-11 below. This will allow a user to either get to their user profile screen, as seen in Figure 2-10 above, access system settings, or allow a user to log out of their current account.

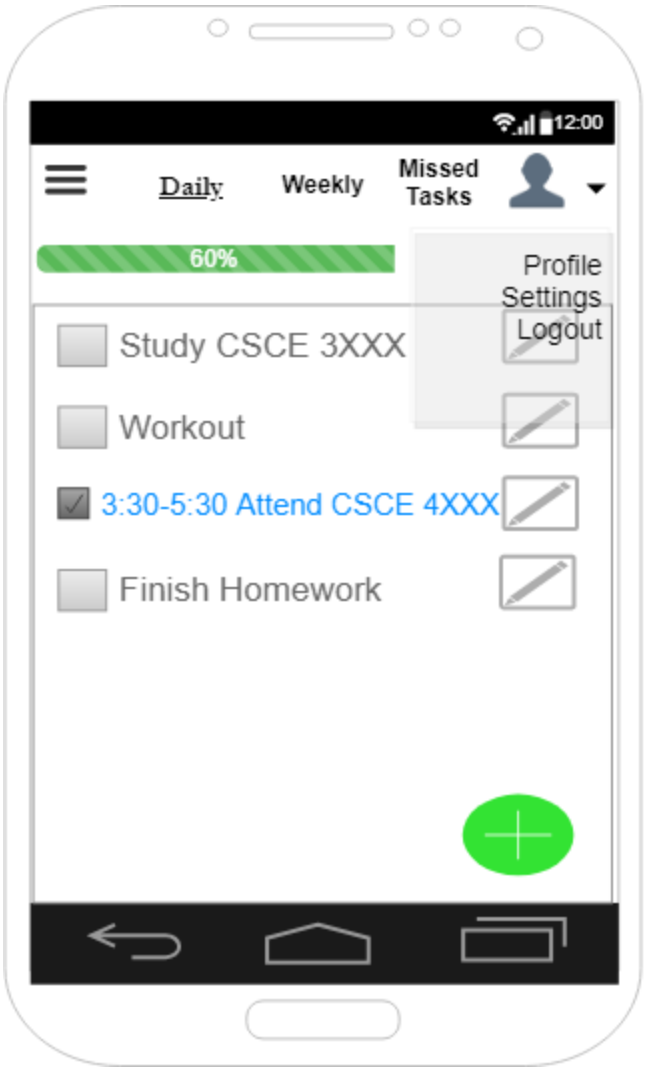

Figure 2-11 Profile dropdown menu

Source: (Draw.io, 2019)

2.1.3 Hardware Interfaces

The primary target of development for this application are Android based phones with ARM Architecture. There are multiple integral pieces of hardware to a majority of Android phones that our application will interface with. A list of these hardware components in which the application interacts can be seen below:

- Android hardware components
  - Internal storage
    - For storing user data through Room (a layer of abstraction over an SQLite database)

- Networking interface hardware (primarily cellular or Wi-Fi based)
  - Used for connecting to the application's external web server as well as other third-party services
- GPS navigation hardware
  - Used for geolocation-based tasks and events, such as a concert or meeting. This will also rely on the Google Maps API.

2.1.4 Software Interfaces

The specific operating system platform this application will be built for is Android v4.0.3 and above, which can be found at: *https://source.android.com/* (replace this with proper IEEE citation if needed.) Additionally, other software products that are required for specific features are Google Maps which can be downloaded on android phones at:

https://play.google.com/store/apps/detailsid=com.google.android.apps.maps&hl=en_US ,

and Google Calendar which can be download on android phones at: https://play.google.com/store/apps/details?id=com.google.android.calendar&hl=en_US . Additionally, our web server will consist of the Linux, Apache, MySQL, and PHP stack, or LAMP stack, which can be seen through the following outline:

- Linux
  - Kernel version: Linux Kernel v5.0
  - Distro version: Ubuntu 18.04.3 (LTS) x64
  - Source: *http://releases.ubuntu.com/18.04/*
- Apache
  - Version: apache2 (from Ubuntu package repository)
  - Source: *http://httpd.apache.org/*
- MySQL
  - Version: mysql-server (from Ubuntu package repository)
  - Source: *https://www.mysql.com/*
- PHP
  - Version: php (v7.2) (from Ubuntu package repository)
  - Source: *https://www.php.net/*

2.1.5 Communications Interfaces

Any online functionality such as synchronization of user data across devices, or access to Google Maps and Calendars features, will require an active internet connection through either a cellular based communication or through Wi-Fi based communication. Additionally, the HTTP/s protocol (which will run on TCP Port 80 for any unsecure connections to third party services that do not support HTTPS and TCP Port 443 for SSL based HTTPS that involves secure user data retrieval and sending as well as access to secure third-party services) will be a primary source of communication.

2.2 Product Functions

The broad goal of this app is to allow a user to form strong daily habits to improve their lifestyle, as well as provide reminders for events that have reoccurrences, such as doctors appointments, oil changes, food stockage, etc. The use will be able to see their progress towards forming strong daily habits both through viewing a percentage-based progress bar on a daily base, as well as percentage-based progress bars for days of the week. Also, a user can see what habit-based tasks are the most missed and see additional details on the task to determine a way to improve their commitment to these tasks. Lastly, the app will have integration with multiple API services such as Amazon Shopping or Google Calendar/Maps to improve their efficiency in completing tasks, as well as gain a stronger understanding of the tasks they need to accomplish.

2.3 User Characteristics

The primary audience this application is being designed for is a general audience, or more specifically, a general audience that uses smartphones daily. This means, for instance, the app could be used to improve a college students study habits, as well as help them form a strong lecture schedule that is enhanced with locational data.

2.4 Constraints

The overall constraints with this app will be any potential flaws that come with the Android operating system itself. Since this app will be designed for a wide audience who uses Android smart phones, their may be differing performance results between devices. Another limit could be the application’s reliance on an externally hosted online server for user data. Although some data can be stored locally on the user’s phone, if the server is ever down or having configuration issues, then this could result in a user’s inability to synchronize accurate data amongst their devices. Another constraint on our system is the reliance on third party servers for API integration. For example, if the servers that hosted the online functionality of Google Calendar were down or not functioning properly, then the ability to export events to Google Calendar could be unavailable. Lastly, since we have decided to use Android Studio and the Android SDK

for development purposes, we may not have the time to integrate any potential cross-platform support with other mobile devices such as iOS.

2.5 Assumptions and dependencies

The primary assumption is that a user is using an Android device that runs on ARM architecture with an operating system version number 4.0.3. If a user does not meet these requirements, then the application may either not be able to be installed, or it may result in incomplete functionality. The app will also depend on a user having a consistent and strong internet connection for the login and registration process, as well as adequate synchronization of data and access to third-party APIs. Otherwise, they may only have access to tasks that are stored on their device for the current month. Finally, for accessing Google Maps for a task's associated location, and Google Calendar for the purpose of exporting time and date details for a task to their calendar service, it is assumed that a user has these apps installed. Otherwise, they may not be able to access these services.

3 Specific Requirements

3.1 External Interface

3.1.1 User interfaces

3.1.1.1 Login screen

- **Purpose:** To allow a preexisting user to login either after logging out of an account, reinstalling the application, removing previous application data, or installing the application on a new device. Also, will allow a user to register for a new account.
- **Source of input:** Initial launch of application or launch after reinstallation/deletion of data. It can accept keyboard input from a user via the textboxes as seen in Figure 2-2.
- **Possible outputs:** Registration screen, daily tasks screen, or a failure message. Can also send information in the textboxes to an external server via an HTTP POST request
- **Relationships to other inputs/outputs:** The ability to login via the login screen is vital for a user to be able to use other features of the application. This screen can also lead to the registration screen by pressing the, "Sign up", button seen in Figure 2-2
- **Screen formats/organization:** Primarily designed for a portrait orientation, with optional scaling for landscape
- **Window formats/organization:** Will be organized in a manner like an HTML form, with boxes for text input as well as a button to submit the data in the form
- **Data formats:** Simple dialog boxes that accept string-based data, and GUI buttons configured in xml
- **Command formats:** Any data sent out will be sent through an HTTP POST request over a secure HTTPS/SSL connection for safety purposes
- **End messages:** If successful, then a user will be taken to the daily tasks screen. If the task fails, then it will depend on the situation. If the HTTP response message from the server takes longer than 2 seconds then the user will receive a message indicating a timeout such as, "Could not login to your account, the connection timed out! Please try again". If a user attempts to enter a password and it does not evaluate to the same hashed password currently stored in the database, then the system will pause for that user 7 seconds, and then send back an error message indicating an unsuccessful login attempt, such as, "Login failed due to incorrect username or password. Please try again."

3.1.1.2 Registration screen

- **Purpose:** To allow a new user, or existing user, to create a new account using a new email address. Allows them to specify a username, email address, and password for a new account.
- **Source of input:** Registration button from the Login screen. It can accept input from a user's keyboard via the textboxes for the Username, Email, and Password textboxes as seen in Figure 2-3
- **Possible outputs:** Daily tasks screen, login screen, or a failure message. It will also output any successfully entered user data to the mySQL database on the external server.
- **Relationships to other inputs/outputs:** Can lead back to the login screen via the "Already a registered user? Sign in here", hyperlink seen in Figure 2-3
- **Screen formats/organization:** Primarily designed for a portrait orientation, with optional scaling for landscape
- **Window formats/organization:** Will be organized in a manner like an HTML form, with dialog boxes for text input as well as a button to submit the data in the form
- **Data formats:** Simple dialog boxes that accept string-based data
- **Command formats:** Any data sent out will be sent through an HTTP POST request over a secure HTTPS/SSL connection for safety purposes
- **End messages:** If successful, the user will be taken to the Daily Tasks screen. If the task fails, then it will depend on the situation. If the HTTP response message from the server takes longer than 2 seconds then the user will receive a message indicating a timeout such as, "Could not login to your account, the connection timed out! Please try again". If the user inputs an email or username that is already stored in the database, then the user will receive error messages accordingly this such as, "This username has already been taken!", or, "This email address already has an account registered on it!".

3.1.1.3 Daily tasks

- **Purpose:** To allow a user to look at the progress of completed tasks for the current day via a percentage progress bar at the top of the screen, edit pre-existing tasks via the pencil icon, and add any new tasks via the circular plus icon at the bottom of the screen. (Put figure number here). Additionally, if an internet connection is available, then the user's current state of task completion on their phone will be synchronized with the state of task completion in the database.
- **Source of input:** Opening the app after a successful user authentication or by selecting the, "Daily Tasks" menu option.
- **Possible outputs:** The task addition screen via the circular cross icon at the bottom right of the screen, a task editing screen for a specific task via the pencil icon to the right of each listed task, and the weekly tasks, and missed tasks screen via the menu options at the top of the screen.

- **Relationships to other inputs/outputs:** Will give task completion information and task description data to the weekly progress sections and missed tasks sections, and it will receive task data from previous dates based on if a task is a reoccurring task or not
- **Screen formats/organization:** Primarily designed for a portrait orientation, with optional scaling for landscape
- **Window formats/organization:** Organized in a checklist format, with editing options via the pencil icons next to each checklist item.
- **Data formats:** The task names and descriptions will be presented in a simple string format, the progress bar will be represented in a GUI format using an Android xml attribute, and the plus icon will also be a GUI element.
- **Command formats:** Any data sent out will be sent through an HTTP POST request over a secure HTTPS/SSL connection for safety purposes, and any data received will be received through an HTTP GET request over a secure HTTPS/SSL connection
- **End messages:** Depending on the situation, the daily tasks screen will either represent an up to date and synchronized result of the description of the screen above in Purpose and Formats, or it will present what is currently stored on the screen and then display an error message if the server does not send an HTTP response in 2 seconds through a dialog box at the bottom of the screen indicating that there was a synchronization error, such as the following, “The device could not connect to the server! Could not sync the tasks!”.

3.1.1.4 Weekly tasks

- **Purpose:** To allow a user to view the overall progress of their task completion both throughout the current week and throughout previous weeks. It will also allow a user to view specific tasks that were completed for a week via the Tasks option to the right of each weekday.
- **Source of input:** From either the daily task or missed tasks screen by selecting the, “Weekly tasks”, menu option at the top of the screen.
- **Possible outputs**: The daily tasks and missed tasks screens via the menu options at the top of the screen, as seen in Figures 2-4, 2-5 through 6, and 2-7. It may also lead to a screen formatted like the daily tasks screen seen in Figure 2-4 that shows an overview of the tasks completed for that weekday by selecting the, “Tasks”, option indicated by the rightward facing arrow to the right of each weekday.
- **Relationships to other inputs/outputs:** Gathers task completion information for the current week and day, as well as task completion information from previous weeks.
- **Screen formats/organization:** Primarily designed for a portrait orientation, with optional scaling for landscape
- **Window formats/organization:** Organized in a vertical menu list format, with a GUI progress bar next to each day of the week gathered from the progress bar Android xml attribute

- **Data formats:** The weekdays and any associated labels will be presented in a simple string format, and the progress bar next to each weekday will be represented in a GUI format using an Android xml attribute
- **Command formats:** Any data sent out will be sent through an HTTP POST request over a secure HTTPS/SSL connection for safety purposes, and any data received will be received through an HTTP GET request over a secure HTTPS/SSL connection
- **End messages:** Depending on the situation, the weekly tasks screen will either represent an up to date and synchronized result of the description of the screen above in Purpose and Formats, or it will present what is currently stored on the individual's phone on the screen and then display an error message if the server does not send an HTTP response in 2 seconds through a dialog box at the bottom of the screen indicating that there was a synchronization error, such as the following, "The device could not connect to the server! Could not sync the weekly tasks!".

3.1.1.5 Missed tasks

- **Purpose:** To allow a user to view the tasks they have missed and the overall frequency of the tasks that they have missed. Also, selecting the right arrow next to the frequency numerical output that represents the average frequency of a user's missed tasks will lead to useful statistical information for a task. This statistical information is TBD.
- **Source of input:** From either the daily task or missed tasks screen by selecting the, "Weekly tasks", menu option at the top of the screen. It will receive its information from the user's task accomplishment information stored currently in the user's internal data storage.
- **Possible outputs:** The daily tasks and missed tasks screens via the menu options at the top of the screen. It may also lead to a screen formatted in a list format for a task via the rightward facing arrow to the right of each task.
- **Relationships to other inputs/outputs:** Gathers task completion information for the current week and day, as well as task completion information from previous weeks.
- **Screen formats/organization:** Primarily designed for a portrait orientation, with optional scaling for landscape
- **Window formats/organization:** Organized in a vertical menu list format that sorts a user's tasks by average frequency of incompletion, with the average frequency information located to the right of each task
- **Data formats:** The tasks and their frequencies will be represented in simple string formats and numerical formats respectively
- **Command formats:** Any data sent out will be sent through an HTTP POST request over a secure HTTPS/SSL connection for safety purposes, and any data received will be received through an HTTP GET request over a secure HTTPS/SSL connection
- **End messages:** Depending on the situation, the weekly tasks screen will either represent an up to date and synchronized result of the description of the screen above in Purpose

and Formats, or it will present what is currently stored on the individual's phone on the screen and then display an error message if the server does not send an HTTP response in 2 seconds through a dialog box at the bottom of the screen indicating that there was a synchronization error, such as the following, "The device could not connect to the server! Could not sync the weekly tasks!".

3.1.1.6 Task addition

- **Purpose:** The purpose of this user interface screen is to allow a user to add a task to their account. This task can either be a one-time task specific to that day, or it can be a habit based task that will reoccur at a frequent amount, a reoccurring task that will remind a user when an important task is coming up at an infrequent interval such as 6 months, 2 weeks, etc., or an important event.
- **Source of input:** The task screen will appear to a user via the circular cross button at the bottom right of the user's screen under Daily Tasks. The task will have input options in the form of input text boxes for information such as the name of the task, as well as radio buttons indicating that a user's task is either a regularly occurring habit based task, a one-time event, or an infrequent task that will occur at a wide interval.
- **Possible outputs:** The app can output the task to the external web server, in which the web server can then store the task in a database which will be used for regular synchronization. Also, the primary three screens can be accessed again by simply pressing the back arrow located in the bottom left of the screen.
- **Relationships to other inputs/outputs:** The information provided via this screen will be used in all of the resulting data shown in the three primary screens: Daily Tasks, Weekly Tasks, Missed Tasks.
- **Screen formats/organization:** Primarily designed for a portrait orientation, with optional scaling for landscape
- **Window formats/organization:** Organized in a vertical orientation with a mixture of user input text boxes, as well as radio buttons for choosing options
- **Data formats**: All the data a user inputs will either be through a simple string format, or will be a predetermined data type based on the option chosen through the radio buttons
- **Command formats:** Any data sent out will be sent through an HTTP POST request over a secure HTTPS/SSL connection for safety purposes, and any data received will be received through an HTTP GET request over a secure HTTPS/SSL connection
- **End messages:** The information shown on a user's screen will depend on the context of the situation. Once a user tries to submit a task at the end of the screen, if the HTTP response indicating a successfully sent message fails to be received in 2 seconds, then the program will instead add the task to a user's local storage, go back to the daily tasks screen, and state in a dialog box at the bottom of the screen, "Could not add task to server. Task will be synchronized with the server later.". If a user submits a task successfully, in that the form submission takes less than 2 seconds, then the user will be

taken back to the daily tasks screen and a dialog box at the bottom of the screen will state, “Task successfully added!”.

3.1.1.7 Task editing

- **Purpose:** Allows the user to edit any preexisting tasks, including details such as the tasks name, the task type, and other details.
- **Source of input:** This screen will receive informational data from a specific task after the pencil icon, representing the ability to edit, is selected next to a task and fill out the text boxes and radio selection buttons accordingly.
- **Possible outputs:** In this screen a user will be able to change the values in any already filled out text information in the text boxes, as well as change the type of task that it is by selecting a different radio button than the one currently selected. The user will be able to submit these changes, and these changes will be given to the external server database if successfully submitted, and also stored in a user’s local database on their device. Also, a user will be able to go back to the previous screen they were located in before opening the task editor by pressing the back arrow icon located in the bottom left of the screen.
- **Relationships to other inputs/outputs:** This screen will rely on information from tasks that occupy the daily tasks page and will affect both tasks for the current day as well as tasks in the future.
- **Screen formats/organization:** Primarily designed for a portrait phone orientation, with optional scaling for landscape
- **Window formats/organization:** Organized in a vertical orientation with a mixture of user input text boxes, as well as radio buttons for choosing options
- **Data formats:** All the data a user inputs will either be through a simple string format, or will be a predetermined data type based on the option chosen through the radio buttons
- **Command formats:** Any data sent out will be sent through an HTTP POST request over a secure HTTPS/SSL connection for safety purposes, and any data received will be received through an HTTP GET request over a secure HTTPS/SSL connection
- **End messages:** Once a user tries to submit an edited task at the end of the screen, if there was no change in the data that was previously filled out, then the app will not submit anything and will not display any dialog. After submission, if the HTTP response indicating a successfully sent message fails to be received in 2 seconds, then the program will instead add the task to a user’s local storage, go back to the daily tasks screen, and state in a dialog box at the bottom of the screen, “Could not add task to server. Task will be synchronized with the server later.”. If a user submits a task successfully, in that the form submission takes less than 2 seconds, then the user will be taken back to the daily tasks screen and a dialog box at the bottom of the screen will state, “Task successfully added!”.

3.1.1.8 Profile screen

- **Purpose:** Allows a user to view information related to their profile including their Full Name if provided, their username, and their profile picture. Also, from this screen a user can easily change their profile picture, and a user can access the ability to edit their profile information.
- **Source of input:** This screen can be accessed by a user selecting the, "Profile", option setting from the drop-down menu seen in (fig num here).
- **Possible outputs:** This application will not output any specific data to other interfaces, unless a user selects the Edit Account Information label. The details for the Edit Account screen are TBD.
- **Relationships to other inputs/outputs:** The dropdown menu that is used to access this screen can be accessed by a user selecting their profile picture. Also, the primary details behind a user's profile, which is their username and email, will determine where their data is stored within the external server's database.
- **Screen formats/organization:** Primarily designed for a portrait phone orientation, with optional scaling for landscape
- **Window formats/organization:** Organized in a vertical orientation with a user's profile picture, Full Name (if provided), User Name, and email all in a centered position going along the middle of a user's screen.
- **Data formats:** The data formats that are displayed on this screen include a mixture simple text strings, and a user's profile picture in either a PNG or JPG format
- **Command formats:** Any data sent out will be sent through an HTTP POST request over a secure HTTPS/SSL connection for safety purposes, and any data received will be received through an HTTP GET request over a secure HTTPS/SSL connection
- **End messages:** If the user receives a successful HTTP response for their information, then the screen described in the formats and purpose above will be shown and no dialog boxes will display at the bottom. Otherwise, if a user does not receive a successful HTTP response within 2 seconds, then the information that would normally be displayed, such as a user name, will be entirely blank, and a user will receive an error dialog at the bottom of the screen that indicates this, such as, "Could not access the server for profile information!".

3.1.1.9 Drop down menu

- **Purpose:** To allow a user to access the Profile Screen, any Application Settings local to their device, the details of which are TBD, and the option to logout.
- **Source of input:** This menu overlay can be accessed by selecting a user's profile picture in the top right of a user's screen when it is visibly displayed in a screen.
- **Possible outputs:** This can allow a user to either be led to the Profile Screen, Application Settings, or a logout screen.

- **Relationships to other inputs/outputs:** This drop down menu does not have any specific relations to other inputs/outputs besides those listed in the two sections above, except for the Logout option which will result in an output that takes a user back to the Login screen they would see upon the initial opening of the app, or on other conditions detailed in Section 3.1.1.1.
- **Screen formats/organization:** This is a simple menu overlay that will take up approximately 1/8 of the top right of screen or less no matter the orientation.
- **Window formats/organization:**
- **Data formats:** All menu labels will be shown in plain string text.
- **Command formats:** Whenever a label is selected, it will simply follow a route to the page that the label is associated with.
- **End messages:** If a user selects the logout button, then the application will return a user to the Login screen along with a dialog box at the bottom of the screen that indicates a successful logout such as, "Successfully Logged Out!"

3.1.2 Hardware interfaces

3.1.2.1 GPS Hardware

- **Purpose:** Used for integration with Google Maps API to help a user get real time directions to the specified location for a task or event.
- **Source of input:** The GPS hardware unit will act as the source of input for the Google Maps API.
- **Units of measure:** The units of measures for latitude and longitude are in degrees.
- **Possible outputs:** The GPS hardware will output latitude and longitude values that are calculated by the Google Maps API for navigational data.
- **Relationships to other inputs/outputs:** The GPS hardware interface receives interaction from the application software whenever a user pulls up the location of an event or task by accessing details about a task that is displayed in the Daily Tasks menu.

3.1.2.2 Wi-Fi and Cellular Hardware

- **Purpose:** Used for establishing a connection to any external or third-party servers.
- **Source of input:** The Wi-Fi antennae and cellular antennae will act as forms of input to the Wi-Fi and cellular components on an Android device's motherboard. The components will receive, at the transport level of networking, TCP data.
- **Units of measure:** Network throughput will be measured in bps (bits per second) ranging from Kbps to Mbps
- **Possible outputs:** The networking hardware will output and HTTPS or other TCP related data through the transport network layer, and propagate this information bit by bit through the physical layer
- **Relationships to other inputs/outputs:** The Wi-Fi and cellular hardware are vital to the overall functionality and synchronization abilities of the app

3.1.2.3 Internal data storage device

- **Purpose:** Used for storing user data locally on the device. This is so data is accessible within a matter of milliseconds rather than having to wait potentially multiple seconds for the network to gather information each time, as well as keep data stored in the event of connection failures.
- **Source of input:** Any user data that has been gathered from successful networking requests will be stored into a local SQLite database on the user's device
- **Units of measure:** Storage throughput can be measured in a range of KB/s (Kilobytes per second) and MB/s (Megabytes per second).
- **Possible outputs:** Any user data that is not stored on the external server's database can be retrieved from a user's local SQLite database and stored onto the server's database.
- **Relationships to other inputs/outputs:** This is primarily related to the overall scheme of user data both offline, or locally stored, and online, or stored on a network.

3.1.3 Software interfaces

3.1.3.1 External Web Server (LAMP stack)

- **Purpose:** Used to store multiple user's data in one central source and allow multiple users to access all their task related data.
- **Source of input:** User network requests which include login requests, registration requests, and task data requests.
- **Units of measure:** Network throughput will be measured in bps (bits per second) ranging from Kbps to Mbps. Also, available server storage will be measured in range of MB (megabytes) to GB (gigabytes).
- **Possible outputs:** Can return user data in a JSON format and can also return HTTP response messages indicating successful or unsuccessful connection attempts.
- **Relationships to other inputs/outputs:** This is primarily related to the overall scheme of user data both offline, or locally stored, and online, or stored on a network.

3.1.3.2 Third Party API services

- **Purpose:** Used to process requests related to functions a task can carry with, such as locational data, the ability to automatically purchase
- **Source of input:** User network requests for
- **Units of measure:** Network throughput will be measured in bps (bits per second) ranging from Kbps to Mbps.

- **Possible outputs:** Can return user data in a JSON format or in a format specified by an APIs documentation.
- **Relationships to other inputs/outputs:** The data that is gathered from the response from an API service can be integrated to the details of a user's task.

3.1.4 Communication interfaces

3.1.4.1 TCP Ports 80 and 443

- **Purpose:** Used for network communication for both the external first-party server as well as any third-party APIs.
- **Source of input:** User network requests
- **Units of measure:** Network throughput will be measured in bps (bits per second) ranging from Kbps to Mbps.
- **Possible outputs:** Can return HTTP/s response information and data through these ports
- **Relationships to other inputs/outputs:** The data that is gathered from these ports are vital to the internet communication abilities of the whole application.

3.2 Functional requirements

3.2.1 User Mode

- The software will require users to set up a profile.
- The software will require users to validate their identities using a unique profile and password.
- The software will allow users to create multiple tasks that develop a daily routine.
- The software will allow users to check a radio box to signify completion of daily task.
- The software will allow users to create custom reoccurring tasks that follow user-defined reoccurrences.
- The software will allow users to schedule appointments.
- The software will generate reminders of upcoming appointments.
- The software will allow users to indicate if a daily routine task is time sensitive.
- If the daily routine task is time sensitive, the software will generate a reminder to the user.
- The software will track completion percentage of daily tasks on a daily basis.
- The software will display weekly averages of completion for the prior week to the user for review.
- The software will highlight most missed tasks and track frequency of misses.
- The software will display completion bars for daily tasks that resets daily and fills with completed tasks.
- The software will connect to external database for storing tasks and calendar information.
- The software will allow for the scheduling of tasks that may occur at irregular intervals.
- The software will allow for users to create custom notes within each data object.

- The software will have the ability to connect to APIs such as Google Maps allowing for the display of appointment locations.

3.3 Performance requirements

The requirements in this section provide a detailed specification of the user interaction with the mobile application

Quality requirement: 6

ID: QR1

TITLE: Search task or event.

DESCRIPTION: The search feature should be prominent and easy to find for the user. In order for a user to find the search feature easily.

ID: QR2

TITLE: The response time of a search.

DESCRIPTION: The response time of a search will begin when a user clicks on the search button, then the request going to server, then the response will be received from the server, and finally the response will be processed by the mobile application.

METER: Measurements obtained from 1000 searches during testing (iOS 9, Android 5.0).

MUST: No more than 2 seconds during 100% of the searches during testing.

ID: QR3

TITLE: The time to open the application.

DESCRIPTION: The front-page load time must be no more than 2 seconds for users starting from the application is opened.

ID: QR4

TITLE: The waiting for inactive user.

DESCRIPTION: In the event the user inactive for longer than 1 minute, the system shall display an informational message to let the user know and log out of the user's account.
MUST: Must waiting for the user inactive for 1 minute.

ID: QR5

TITLE: Check for duplicate task.

DESCRIPTION: If the user input a duplicate task, the system must let the user know to ask for user's decision.

ID: QR6

TITLE: Require for user password.

DESCRIPTION: The user password must contain at least one upper letter or one special letter, and the password length must be longer than 8 characters.

3.4 Design constraints

The app will be designed to operate as an Android mobile application that will run on an Android phone. The app must be able to run on Android. We will use android studio to build and test the app. We all utilize android studio Framework for deployment. We will utilize virtual devices and real devices for user testing and development. The user must be able to retrieve their data from multiple different devices. It must be able to retrieve information on the internet from the cloud. On the cloud we will store the reminders and user profiles. The application must provide live tracking of time remaining on app. There must be sorting implemented to sort reminders for users to access. The daily reminders must reset daily. Tracking data for completion of daily tasks must be sent to the cloud when the application is opened. Any changes made to user data or the reminders should be reflected on the user's account and be uploaded to the cloud immediately. The deletion of a reminder should not remove the data from the tracking data. Must be able to send push notifications to users regarding tasks. Admins can access statistics and reminders of users. Admins cannot access info of users. The app should be able to handle the design constrains of other standard android applications. They should be able to run in the background, handle multiple systems running at the same time and interact with the rest of the device while the app is running. The app should be able to communicate with other applications but does not offer support to other apps. Should send info to calendar applications. The app should be able to handle resource constrains from phones and be able to handle being killed by the operating system. The app's components should not contain app data or state and should not depend on one another. The data model needs to hold the user ID and user object to drive the UI.

All Data passed from the cloud to the device should pass through the repository modules to handle data operations. Sources of data for the app will not be held in the app's entry points. Rather, they will coordinate with other components to retrieve the subset of data that is relevant to that entry point. The UI of the app will be derived from the components that are responsible for handling data. This will protect the app from lifecycle or the concerns from the device.

3.5 Other requirements

Any additional requirements are TBD.