Software Design Descriptions

Consolidated Health App

Cody Price
Rafal Ryczek
Aaron Jordan
Quinn Poyneer

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this SDD is to create a completed design description on how the consolidated health app, HealthOut will be built. This document will describe the three main subsystems, the seventeen modules used by those subsystems, how each subsystem interacts with one another, how each module inside each subsystem interacts with one another, the database and how its tables are used by each subsystem and their modules, and the services each subsystem and their modules provides to the user. These descriptions will be accompanied with documentation including case models, system architecture, object collaboration diagrams, and sequence diagrams. The chosen software process and it's steps in order to create the software will also be discussed. [1]

The expected audience of this document is Dr. Yeong-Tae Song (the client of this project), the group 5 development team (the developers of this project), and the developers that will handle the maintenance of this project. Other possible expected audience of this document would be other students taking COSC 412-001 at Towson University. [3]

1.2 Scope

HealthOut is a mobile application that allows the user to consolidate their log data from multiple 3rd party health apps (provided the 3rd party health apps have an available API that we have been given access to). The user can then set health-related goals, view their progress, and displays each goal graphically.

There are three subsystems to HealthOut, being the User Account subsystem, the Compatible Apps subsystem, and the Health Goals subsystem. The User Account subsystem handles all matters related to the user's account for HealthOut. The Compatible Apps subsystem handles all matters related to the 3rd party apps that are compatible for consolidation with HealthOut. The Health Goals subsystem handles all matters related to the user's health goals and displaying those goals. [2]

1.3 Definitions

HealthOut	Name of the consolidated health app project
User	Someone that interacts with HealthOut
API	Code that allows two software programs to communicate with each other
Health app	An app that collects the users data about

	the users health and stores it as log data.
log data	Data that will be stored in a database. For HealthOut we will be pulling log data that refers to facts and statistics, pertaining to health (i.e. steps walked, miles walked, calories burned, etc.), collected together for reference or analysis.
3rd party developer	Software developers who do not belong to the Group 5 software development team
3rd party app	Programs developed by a 3rd party developer
Android Studio	The official integrated development environment for Google's Android operating system.
Graphic User Interface (GUI)	The graphical user interface is a form of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.
Database	Program used for storing, finding, and changing information
External Database	A database stored on the server of a 3rd party company that HealthOut will have for only it’s users through that companies API.
Subsystem	A collection of modules representing one part of the software. Used to separate and organize the system architecture. (Ex: Account management, Commerce)
Module	A component of the software with unique functionality. (Ex: Display product, Login to system)
Data entity	An object in the system used to store

	information. (Ex: Customer, Seller)
Concurrent Process	Piece of software executing simultaneously with other processes. (Ex: Internet browser tabs)
Class	An extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions).
Object	Refers to a particular instance of a class.
Function	A named section of a program that performs a specific task.
Variable	A data value that can change, depending on conditions or on information passed to the program.
Parameter	Variables used to pass information between functions or procedures.
Data type	An attribute of data which tells the compiler or interpreter how the programmer intends to use the data.
String	A data type made up of a sequence of characters.
Boolean	A data type that is either true or false.
Integer	A data type made up of whole numbers
Double	A data type made up of whole numbers, fractions, or decimals

2. References

[1] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications

[2] (Joan Teamleader, Paul Adams, Bobbie Baker, Charles Charlie), 2004, Web Publishing System

[3] Masters Studio Project, 2003, Web Accessible Alumni Database

[4] BluePay, SwipeSimple Mobile Application User Guide

[5] Atlanta Regional Commission – MSAA, 2017, System Design Document

[6] Penn State Harrisburg Fall, 2006, Detailed Design Document Format

3. Decomposition description

3.1 Database decomposition

HealthOut uses a single external database through the Microsoft Azure API. This database will hold all data for all users of our app. Each user's data will be spread across several related tables, each corresponding to a different category of data. Currently there are six known tables and many more possible tables. The known tables are the **Users**, **3rdPartyApp**, **3rdPartyAppLogin**, **goal_types**, **TimePeriod**, and the **Goal tables**. The many more possible tables would be a table for every compatible 3rd party health app to store every user's log data for that particular app. For simplicity, we will collectively refer to these possible tables as the **CompatibleAppLogs tables.**

3.1.1 User Table

This table stores each HealthOut user's account information. This includes a unique id number for each user's account, their email address, and their password.

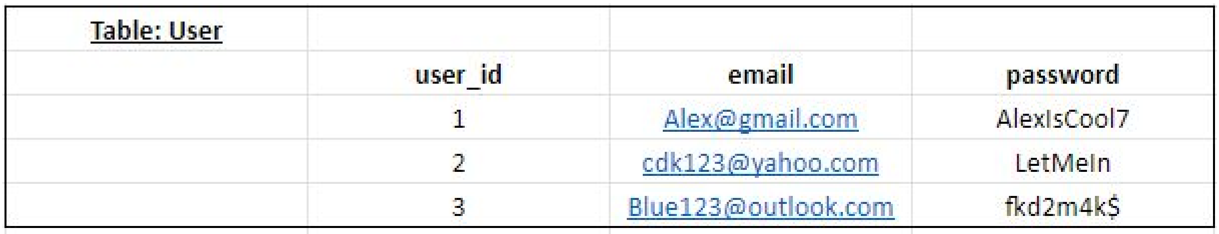


Table: User			
	user_id	email	password
	1	Alex@gmail.com	AlexIsCool7
	2	cdk123@yahoo.com	LetMeIn
	3	Blue123@outlook.com	fkd2m4k$

3.1.2 App table

This table stores each 3rd party health app that is compatible with HealthOut. This includes a unique id number for each app and their name.

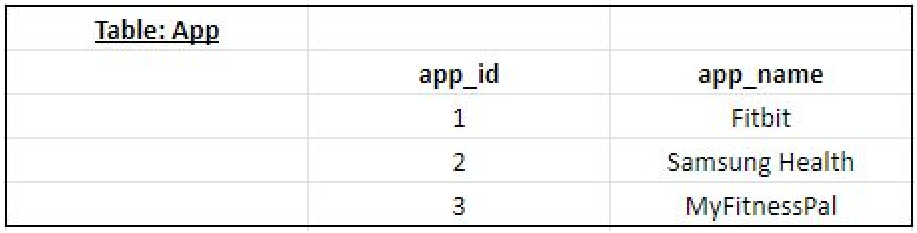


Table: App		
	app_id	app_name
	1	Fitbit
	2	Samsung Health
	3	MyFitnessPal

3.1.2 API table

This table stores each HealthOut user's login information for all 3rd party health apps that they register to HealthOut. This includes the user id from the **User Table**, the app id from the **App table**, whether or not the user has a particular app registered, the user's username for the app, the user's email address for the app, and the user's password for the app.

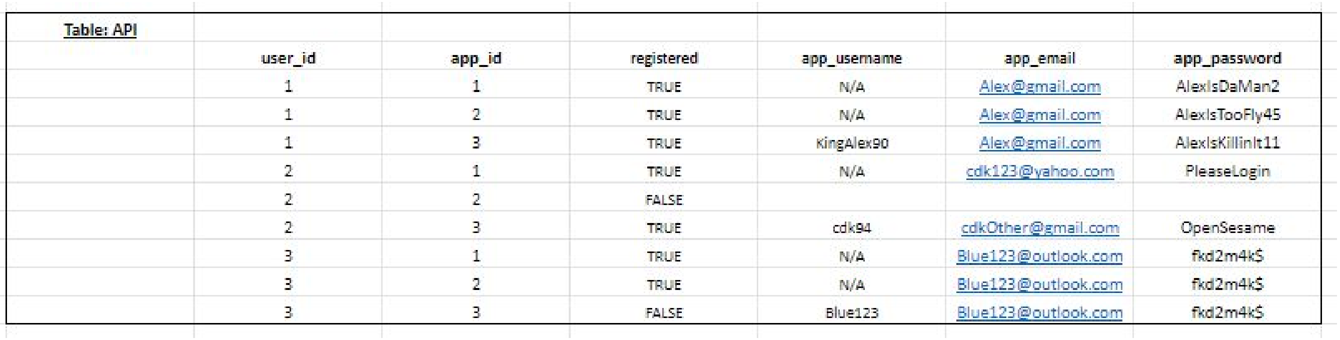


Table: API						
	user_id	app_id	registered	app_username	app_email	app_password
	1	1	TRUE	N/A	Alex@gmail.com	AlexIsDaMan2
	1	2	TRUE	N/A	Alex@gmail.com	AlexIsTooFly45
	1	3	TRUE	KingAlex90	Alex@gmail.com	AlexIsKillinIt11
	2	1	TRUE	N/A	cdk123@yahoo.com	PleaseLogin
	2	2	FALSE			
	2	3	TRUE	cdk94	cdkOther@gmail.com	OpenSesame
	3	1	TRUE	N/A	Blue123@outlook.com	fkd2m4k$
	3	2	TRUE	N/A	Blue123@outlook.com	fkd2m4k$
	3	3	FALSE	Blue123	Blue123@outlook.com	fkd2m4k$

3.1.2 Type table

This table stores all compatible health metrics that HealthOut will track and allow the user to make goals for (Steps Walked, Calories Burned, etc). This includes a unique id number for each goal type and the name of the goal type.

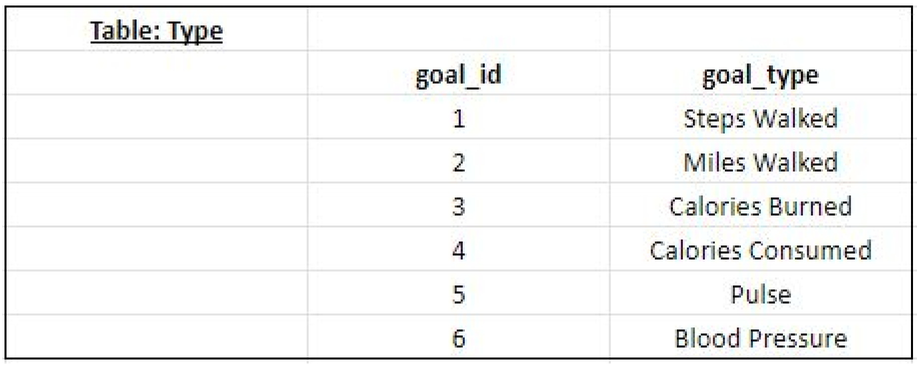


Table: Type		
	goal_id	goal_type
	1	Steps Walked
	2	Miles Walked
	3	Calories Burned
	4	Calories Consumed
	5	Pulse
	6	Blood Pressure

3.1.2 Period table

This table stores all compatible time periods for which a goal can be set for (daily, weekly, monthly, yearly). This includes a unique id number for each time period and the name of the time period.

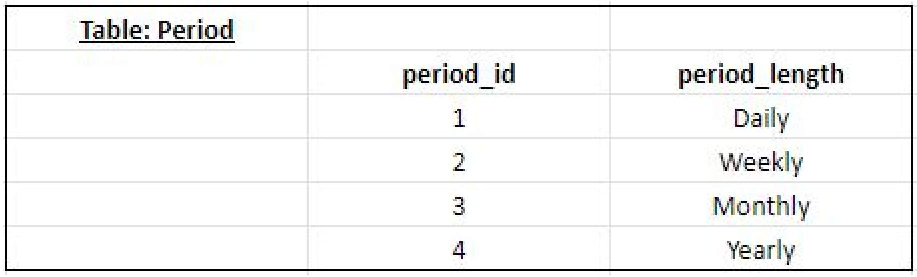


Table: Period		
	period_id	period_length
	1	Daily
	2	Weekly
	3	Monthly
	4	Yearly

3.1.2 Goal table

This table stores each HealtOut user's goals. This includes the user id from the **User Table**, the app id from the **App table**, the goal id from the **Type table**, the period id from the **Period table**, and the target value for the goal.

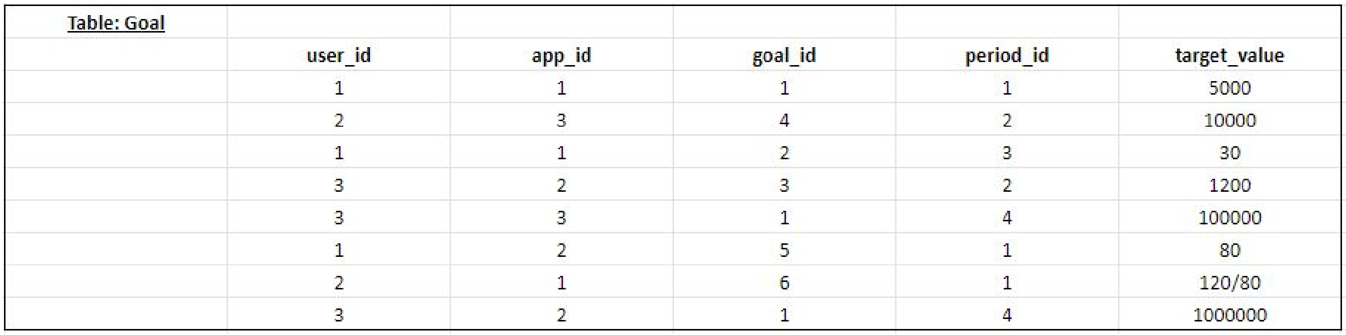


Table: Goal	user_id	app_id	goal_id	period_id	target_value
	1	1	1	1	5000
	2	3	4	2	10000
	1	1	2	3	30
	3	2	3	2	1200
	3	3	1	4	100000
	1	2	5	1	80
	2	1	6	1	120/80
	3	2	1	4	1000000

3.1.2 CompatibleAppLogs Tables

Each one of these tables corresponds to a single compatible 3rd party health app with HealthOut. Each one of these tables stores each HealthOut user's log data for their particular app. Each of these tables includes includes the user id from the **User Table**, all goal types from the **Type table**, and an epoch timestamp for the time at which the data was stored.

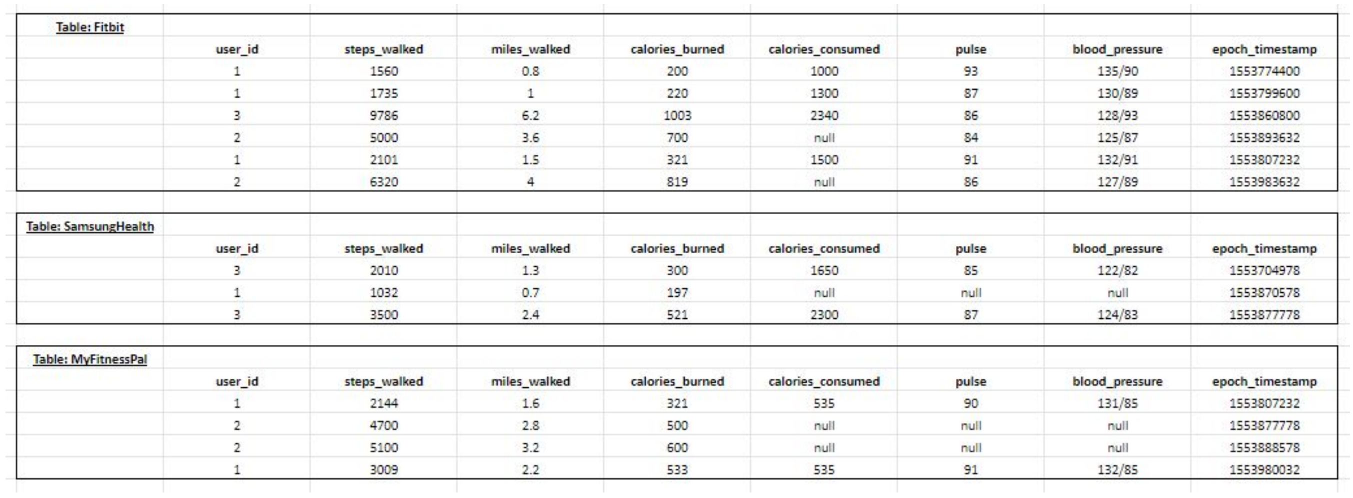


Table: Fitbit	user_id	steps_walked	miles_walked	calories_burned	calories_consumed	pulse	blood_pressure	epoch_timestamp
	1	1560	0.8	200	1000	93	135/90	1553774400
	1	1735	1	220	1300	87	130/89	1553799600
	3	9786	6.2	1003	2340	86	128/93	1553860800
	2	5000	3.6	700	null	84	125/87	1553893632
	1	2101	1.5	321	1500	91	132/91	1553807232
	2	6320	4	819	null	86	127/89	1553983632

Table: SamsungHealth	user_id	steps_walked	miles_walked	calories_burned	calories_consumed	pulse	blood_pressure	epoch_timestamp
	3	2010	1.3	300	1650	85	122/82	1553704978
	1	1032	0.7	197	null	null	null	1553870578
	3	3500	2.4	521	2300	87	124/83	1553877778

Table: MyFitnessPal	user_id	steps_walked	miles_walked	calories_burned	calories_consumed	pulse	blood_pressure	epoch_timestamp
	1	2144	1.6	321	535	90	131/85	1553807232
	2	4700	2.8	500	null	null	null	1553877778
	2	5100	3.2	600	null	null	null	1553888578
	1	3009	2.2	533	535	91	132/85	1553980032

3.2 Subsystem decomposition

Detailed descriptions of the subsystems used by HealthOut, outlining how the project is divided and what each section is in charge of.

3.2.1 User Account subsystem

The User Account subsystem handles all matters related to the user's account for HealthOut. This subsystem is the coupling of the following eight modules: Register Account, Activate Email, Delete Account, Recover Password, Login, Logout, Change Email, and Change Password.

This subsystem has two actors, the primary actor being that user that triggers all the modules used by this subsystem, and the secondary actor being the Microsoft Azure API that hosts our apps database.

This subsystem uses a single table in our Microsoft Azure database, being the **User table**. All modules used by this subsystem, except the Logout Module, will interact with the **User table**. The Logout module does not interact with any databases.

The User Account subsystem is an independent subsystem, meaning that it does not rely on the other two subsystems to work. The only association between the User Account subsystem and the other two is that the other two subsystems can only run once the user first registers an account and/or logs in through the User Account subsystem. [1]

3.2.2 Compatible Apps subsystem

The Compatible Apps subsystem handles all matters related to the 3rd party apps that are compatible for consolidation with HealthOut. This subsystem is the coupling of the follow four modules: Display 3rd Party Apps, Register App, Unregister App, and Update Logs.

This subsystem has three actors, the primary actor being that user that triggers all the modules used by this subsystem, and the two secondary actors being the Microsoft Azure API that hosts our database and the APIs of all compatible 3rd party health apps.

This subsystem uses 3 of the known tables, as well as the still to be determined **CompatibleAppLogs tables,** in our Microsoft Azure database. The three known tables being the **Users**, **3rdPartyApp**, and the **API tables**. The Display 3rd Party Apps module does not interact with any tables in the HealthOut database. The Register App, Unregister App, and Update Logs modules will interact with the **Users**, **3rdPartyApp**, and the **API tables**. The Register App and Update Logs modules will also interact with all **CompatibleAppLogs tables**.

The Compatible Apps subsystem is a dependent subsystem, as it relies on the User Account subsystem for the user to have registered an account and have logged in before it can work. The Compatible Apps subsystem is also associated with the Health Goals subsystem, as the Health Goals subsystems relies on Compatible Apps subsystem to store and update the log data in the **Apps Logs database**. [1]

3.2.3 Health Goals subsystem

The Health Goals subsystem handles all matters related to the user's health goals and displaying those goals. This subsystem is the coupling of the follow six modules: Display Goals to Edit, Set Goal, Change Goal, Remove Goal, Display Goals to Main, and Display Progress Graph.

This subsystem has two actors, the primary actor being that user that triggers all the modules used by this subsystem, and the secondary actor being the Microsoft Azure API that hosts our apps database.

This subsystem uses all six known tables, as well as the still to be determined **CompatibleAppLogs tables,** in our Microsoft Azure database. The six known tables being the **Users**, **3rdPartyApp**, **3rdPartyAppLogin**, **goal_types**, **TimePeriod**, and the **Goal tables**. The Display Goals to Edit and Display Goals to Main modules will not interact with any tables in the HealthOut database. The Display Progress Graph will interact with all tables used in the HealthOut database. The Display Progress Graph The Set Goals, and Change Goals modules will use all known tables but not the **CompatibleAppLogs tables**. The Remove Goals module will only interact with the **Users** and **Goal tables.**

The Health Goals subsystem is a dependent subsystem, as it relies on both the User Account subsystem and the Compatible Apps subsystem to work. The Health Goals subsystem relies on the User Account subsystem for the user to have registered an account and have logged in first. The Health Goals subsystem relies on the Compatible Apps subsystem for the user to have registered at least one 3rd party health app to HealthOut and for their log data to be stored and updated in the **App Logs database**. [1]

3.3 Module decomposition

Detailed descriptions of the modules used by HealthOut, outlining all the components and functionalities of the app.

3.3.1 Register Account module

This module is responsible for allowing the user (primary actor) to register an account to HealthOut. This module is codependent with the Active Email module. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses a single table in the database, being the **User table.** This module has permission to alter the data that is in the **User table**.

This module uses a single GUI, being the Register Account page, which will ask the user to enter their desired email address and password. If both are valid inputs, as in the email address follows the 'prefix@domain' format and is not already registered to an account, and the password is between 5 - 50 characters, then this module will pass that email to the Activate Email module. When the Activate Email module returns true, then will try to gain access to HealthOut's database through the Microsoft Azure API. If successful, then this module will create an account for the user in the **User table**. Otherwise, the user will be prompted to try again later. [1]

3.3.2 Activate Email module

This module is responsible for allowing the user to activate an email address for HealthOut. This module is codependent with either the Register Account module or the Change Email module. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses a single table in the database, being the **User table.** This module has permission to alter the data that is in the **User table**.

This module uses no GUI, but is instead activated by either the Register Account module or the Change Email module. When called by either one of theses two modules, they will pass this module an email address. This module will then send an activation email to that email address that contains an activation link. When the user opens that email and clicks that link, a signal will be sent to this module which will return true to the module which called it. [1]

3.3.3 Delete Account module

This module is responsible for allowing the user (primary actor) to delete their account off HealthOut. This module is dependent upon the Register Account module as described in section 4.1.2 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses a single table in the database, being the **User table.** This module has permission to alter the data that is in the **User table**.

This module uses a single GUI, being the Account Options sidebar, which will give the user the option to delete their account. If a user decides they'd like to delete their account, then this module will try to gain access to HealthOut's database through the Microsoft Azure API. If successful, then this module will take their account information out of the **User table**. Otherwise, the user will be prompted to try again later. [1]

3.3.4 Recover Password module

This module is responsible for allowing the user (primary actor) to recover their password for their HealthOut account. This module is dependent upon the Register Account module as described in section 4.1.3 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses a single table in the database, being the **User table.** This module has permission to alter the data that is in the **User table**.

This module uses a single GUI, being the Login page. When the user clicks the text that says "Forgot Password?", a prompt will appear asking the user to enter their email address. After doing so, if the email address is valid, as in it follows the 'prefix@domain' format, then this module will try to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then the user will be prompted to try again later. If successful, then this module will search the **User table** to make sure the email is registered to an account. If it is, then a recovery email will be sent to that email address. This email will include the password to that account. [1]

3.3.5 Login module

This module is responsible for allowing the user (primary actor) to login to HealthOut. This module is dependent upon the Register Account module as described in section

4.1.4 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses a single table in the database, being the **User table.** This module has permission to alter the data that is in the **User table**.

This module uses a single GUI, being the Login page, which will ask the user enter the email and password to an account. This module will then try to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then the user will be prompted to try again later. If successful, then this module will search the **User table** to see if the email and password the user entered correctly matches an account. If so, then the user will be logged into that account. [1]

3.3.6 Logout module

This module is responsible for allowing the user (primary actor) to logout of their HealthOut account. This module is dependent upon the Register Account and Login modules as described in section 4.1.5 of this SDD. This Module does not use the HealthOut databases.

This module uses a single GUI, being the Account Options sidebar, which will give the user the option to logout of their account. If a user decides they'd like to logout of their account, then they will be logged out and taken to the login page. [1]

3.3.7 Change Email module

This module is responsible for allowing the user (primary actor) to change the email address to their HealthOut account. This module is dependent upon the Register Account, Login, and Activate Email modules as described in section 4.1.6 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses a single table in the database, being the **User table.** This module has permission to alter the data that is in the **User table**.

This module uses two GUI, being the Account Options sidebar and the Edit Account page. First this module will use the Account Options sidebar to give the user the option to edit their account. If a user decides they'd like to edit their account, then they will be taken to the Edit Account page. Here they will need to enter their newly desired email and their current password. If the email they entered follows the 'prefix@domain' format, then this module will try to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then the user will be prompted to try again later. If successful, then this module will search the **User table** to make sure the new email is not already registered to an account and that the password they entered correctly matches their password. If so, then this module will pass that email to the Activate Email module. When the Activate Email module returns true, then this module will update the user's email in the **User table**. [1]

3.3.8 Change Password module

This module is responsible for allowing the user (primary actor) to change the password for their HealthOut account. This module is dependent upon the Register Account and Login modules as described in section 4.1.7 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses a single table in the database, being the **User table.** This module has permission to alter the data that is in the **User table**.

This module uses two GUI, being the Account Options sidebar and the Edit Account page. First this module will use the Account Options sidebar to give the user the option to edit their account. If a user decides they'd like to edit their account, then they will be taken to the Edit Account page. Here they will need to enter their current password and their newly desired password. If their newly desired password is valid (between 5 - 50 characters), then this module will try to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then the user will be prompted to try again later. If successful, then this module will check the **User table** to make sure the current password they entered correctly matches their current password. If so, then this module will update the user's password in the **User table**. [1]

3.3.9 Display 3rd Party Apps module

This module is responsible for displaying for the user (primary actor) the list of all compatible 3rd party health apps with HealthOut, as well as which apps the user has registered/unregistered. This module is dependent upon the Register Account and Login modules as described in section 4.1.8 of this SDD. This Module does not use the HealthOut databases.

This module uses a single GUI, being the Register Apps page. This module will loop through all 3rd Party App data entities to see which apps the user currently has registered/unregistered. This module will then display the list of apps, along with which are registered/unregistered on the Register Apps page. [1]

3.3.10 Register App module

This module is responsible for allowing the user (primary actor) to register 3rd party health apps to HealthOut. This module is dependent upon the Register Account, Login, and the Display 3rd Party Apps modules as described in section 4.1.9 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses three known tables in the database, and the still to be determined **CompatibleAppLogs tables.** The known tables being the the **Users**, **3rdPartyApp**, and the **API tables.** This module can alter the data in the **API table** and the **CompatibleAppLogs tables**, but only has permission to pull data from the **Users** and **App tables**.

This module uses a single GUI, being the Register Apps page. When the user clicks the switch button to next to an unregistered 3rd party health app that they wish to register, this module will try to gain access to HealthOut's database through the Microsoft Azure

API. If unsuccessful, then the user will be prompted to try again later. If successful, then this module will pull what the app's id is from the **App table**, and what the user's id is from the **User table,** before using them to search through the **API table** to see if the user already has login information (username, email, password) for the selected app. If it does, then it will be sent to the selected app's API for verification. If the API gives no response due to being down or unavailable, then the user will be prompted to try again later. If the login information is still correct, then the selected app will be updated as registered in the **API table**. If it comes back as incorrect, then the login information for the selected app will be removed from the **API table**.

If there is no login information for the selected app stored in **API table**, or the stored login information came back as incorrect, then this module will prompt the user for their login information for the selected app. This module will then send that login information to the selected app's API. If the API gives no response due to being down or unavailable, then the user will be prompted to try again later. If it comes back as correct, then the selected app's API will also send the user's log data for the selected app. This module will store that log data in the app's corresponding **CompatibleAppsLogs table**. Then this module will store update the app as registered, as well as store the validated email address and password for the app, in the **API table**. [1]

3.3.11 Unregister App module

This module is responsible for allowing the user (primary actor) to unregister 3rd party health apps to HealthOut. This module is dependent upon the Register Account, Login, and the Display 3rd Party Apps modules as described in section 4.1.10 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses three tables in the database, being the the **Users**, **3rdPartyApp**, and the **API tables.** This module can alter the data in the **API table**, but only has permission to pull data from the **Users** and **App tables**.

This module uses a single GUI being the Register Apps page. When the user clicks the switch button to next to an registered 3rd party health app that they wish to unregister, this module will try to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then the user will be prompted to try again later. If successful, then this module will pull what the app's id is from the **App table**, and what the user's id is from the **User table,** before using them to locate and the app as unregistered for the user in the **API table**. [1]

3.3.12 Update Logs module

This module is responsible for updating the user's (primary actor) log data for all registered apps in the **App Logs database**. This module is dependent upon the Register Account, Login, and the Register Apps modules as described in section 4.1.11 of this SDD. This module has two secondary actors, being the Microsoft Azure API that hosts HealthOut's database, and the APIs of all compatible 3rd party health apps. This module uses three known tables in the database, and the still to be determined

CompatibleAppLogs tables. The known tables being the the **Users**, **3rdPartyApp**, and the **API tables.** This module can alter the data in the **API table** and the **CompatibleAppLogs tables**, but only has permission to pull data from the **Users** and **App tables**.

This module both uses the Main Menu page and also runs in the background routinely (at least once every hour). Whether activated because the user clicked the "Update" button on the Main Menu page, or it is time for a routine update, this module checks for updated log data from all registered 3rd party health apps through their APIs. This module does this by first trying to gain access to HealthOut's database through the Microsoft Azure API. If successful, then this module will pull what the user's id is from the **User table** and use to go through the **API table** chronologically and each time it finds a 3rd party health app that is registered to the user's id, it will try to update the user's log data for that app.

This module updates a user's log data for an app by first pulling the user's login information (username, email, password) for that app before sending it to that app's API. If it come back as incorrect, then this module will display an error prompt on the screen telling the user their login for that particular app is no longer valid. This module will then remove that login information from the **API table**, as well as update the app as unregistered. If it comes back as correct, then the chosen app's API will also send the user's log data for that app. This module will store that log data in the app's corresponding **CompatibleAppsLogs table**. [1]

3.3.13 Display Goals to Edit module

This module is responsible for displaying all the user's (primary actor) goals to the HealthOut Edit Goals page. This module is dependent upon the Register Account, Login, and the Register App modules as described in section 4.1.12 of this SDD. This Module does not use the HealthOut databases.

This module uses a single GUI being the Edit Goals page. First this module will loop through all 3rd Party App data entities to see if the user has at least one registered app. If the user does not, then a message is displayed on the Edit Goals page telling the user to register a 3rd party health app. If the user does have at least one registered app, then this module will loop through all Goal data entities and display the user's goal(s) on the Edit Goals page, with a "Add new" button directly below the last displayed goal. If the user does not have any set goals, then only the "Add new" button will appear. [1]

3.3.14 Set Goal module

This module is responsible for allowing the user (primary actor) to set new goals for themselves in HealthOut. This module is dependent upon the Register Account, Login, Register App, and the Display Goals to Edit modules as described in section 4.1.13 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses all six known tables in the database**.** This module has permission to alter the **Goal table**, but only has permission to pull data from the remaining five known tables.

Each goal will correspond to a particular registered 3rd party health app, a goal type (steps walked, miles walked, calories consumed, calories burned, blood pressure, and pulse), target for that goal type (i.e. 1000 steps, 3 miles, 500 calories, etc.), and time period for that goal type (day/week/month/year).

This modules uses two GUI, being the Edit Goals page and the Goals Details page. When the user clicks the "Add new" button on the Edit Goals page, they will be taken to the Goals Details page. Here the user can set which app is selected in the "Data pulled from" drop down list, which goal type is selected in the "Goal Type" drop down list, what number is in the "Target" text box, and what time period is selected in the "Period" drop down list. Only registered apps to the user appear in the "Data pulled from" drop down list.

This is achieved by the module first trying to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then then they will be returned to the Edit Goals page and a message will display telling the user to try again later. If successful, then pull what the user's id is from the **User table** and use this to search through the **API table** chronologically to see if a registered app appies to the user. If it does, then the app's id will be pulled and used to search through the **App table** where the name of the corresponding app will be pulled and sent to the "Data pulled from" drop down list. After all the user's registered 3rd party health app have been sent to the "Data pulled from" drop down list, this module will search through the **Type table** and send all goal types to the "Goal Types" drop down list. Then this module will search through the **Period table** and send all time periods to the "Period" drop down list. When the user clicks the "Apply" button, if all drop down lists and text boxes are filled, then this module will use the user's pull id to search the **UsersGoals** to see if the goal is a duplicate. If so, then a message will appear saying the goal is a duplicate, otherwise the goal will be added to the **Goal table**. [1]

3.3.15 Change Goal module

This module is responsible for allowing the user (primary actor) to change a goal in HealthOut. This module is dependent upon the Register Account, Login, Register App ,Set Goal, and the Display Goals to Edit modules as described in section 4.1.14 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses all six known tables in the database**.** This module has permission to alter the **Goal table**, but only has permission to pull data from the remaining five known tables.

Each goal will correspond to a particular registered 3rd party health app, a goal type (steps walked, miles walked, calories consumed, calories burned, blood pressure, and pulse), target for that goal type (i.e. 1000 steps, 3 miles, 500 calories, etc.), and time period for that goal type (day/week/month/year).

This modules uses two GUI, being the Edit Goals page and the Goals Details page. When the user clicks a goal on the Edit Goals page, they will be taken to the Goals Details page and the data for the goal (app pulled from, goal type, time period, and target value) will also be sent.. Here the user can set which app is selected in the "Data pulled from" drop down list, which goal type is selected in the "Goal Type" drop down list, what number is in the "Target" text box, and what time period is selected in the "Period" drop down list. Only registered apps to the user appear in the "Data pulled from" drop down list. Also all drop down lists and text boxes start holding the values sent to the page.

The module will then try to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then then they will be returned to the Edit Goals page and a message will display telling the user to try again later. If successful, then pull what the user's id is from the **User table** and use this to search through the **API table** chronologically to see if a registered app appies to the user. If it does, then the app's id will be pulled and used to search through the **App table** where the name of the corresponding app will be pulled and sent to the "Data pulled from" drop down list. After all the user's registered 3rd party health app have been sent to the "Data pulled from" drop down list, this module will search through the **Type table** and send all goal types to the "Goal Types" drop down list. Then this module will search through the **Period table** and send all time periods to the "Period" drop down list. When the user clicks the "Apply" button, if all drop down lists and text boxes are filled, then this module will use the user's pulled id to search the **UsersGoals** to see if the goal is a duplicate. If so, then a message will appear saying the goal is a duplicate, otherwise the goal will be added to the **Goal table**. [1]

3.3.16 Remove Goal module

This module is responsible for allowing the user (primary actor) to remove a goal from HealthOut. This module is dependent upon the Register Account, Login, Register App, Set Goal, and the Display Goals to Edit modules as described in section 4.1.15 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses the **Users** and **Goal tables**. This modules has permission to alter the **Goal table**, but only has permission to pull from the **User table.**

This modules uses a single GUI, being the Edit Goals page. When the user clicks the "Remove" button, then check boxes will appear next to all set goals. After the user checks the apps they would like to remove and clicks the "Remove" button again, this module will then try to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then then a message will display telling the user to try again later. If successful, then this module will pull what the user's id is from the **User table** and use this to search through the **Goal table** and remove those particular goals from the **Goals database**. [1]

3.3.17 Display Goals to Main module

This module is responsible for displaying all the user's (primary actor) goals to the HealthOut Main Menu page. This module is dependent upon the Register Account, Login, Register App, and the Set Goal modules as described in section 4.1.16 of this SDD. This Module does not use the HealthOut databases.

This module uses a single GUI being the Main Menu page. First this module will will loop through all 3rd Party App data entities to see if the user has at least one registered app. If the user does not, then a message is displayed on the Main Menu page telling the user to register a 3rd party health app. If the user does have at least one registered app, then this module will loop through all Goal data entities and display the user's goal(s) on the Main Menu page, [1]

3.3.18 Display Progress Graph module

This module is responsible for allowing the user (primary actor) to view a progress graph for a particular goal they have set in HealthOut. This module is dependent upon the Register Account, Login, Register App, Set Goal, and the Display Goals to Main modules as described in section 4.1.16 of this SDD. This module has a secondary actor, being the Microsoft Azure API that hosts HealthOut's database. This module uses all tables in the database**.** This module only has permission to pull data from all tables.

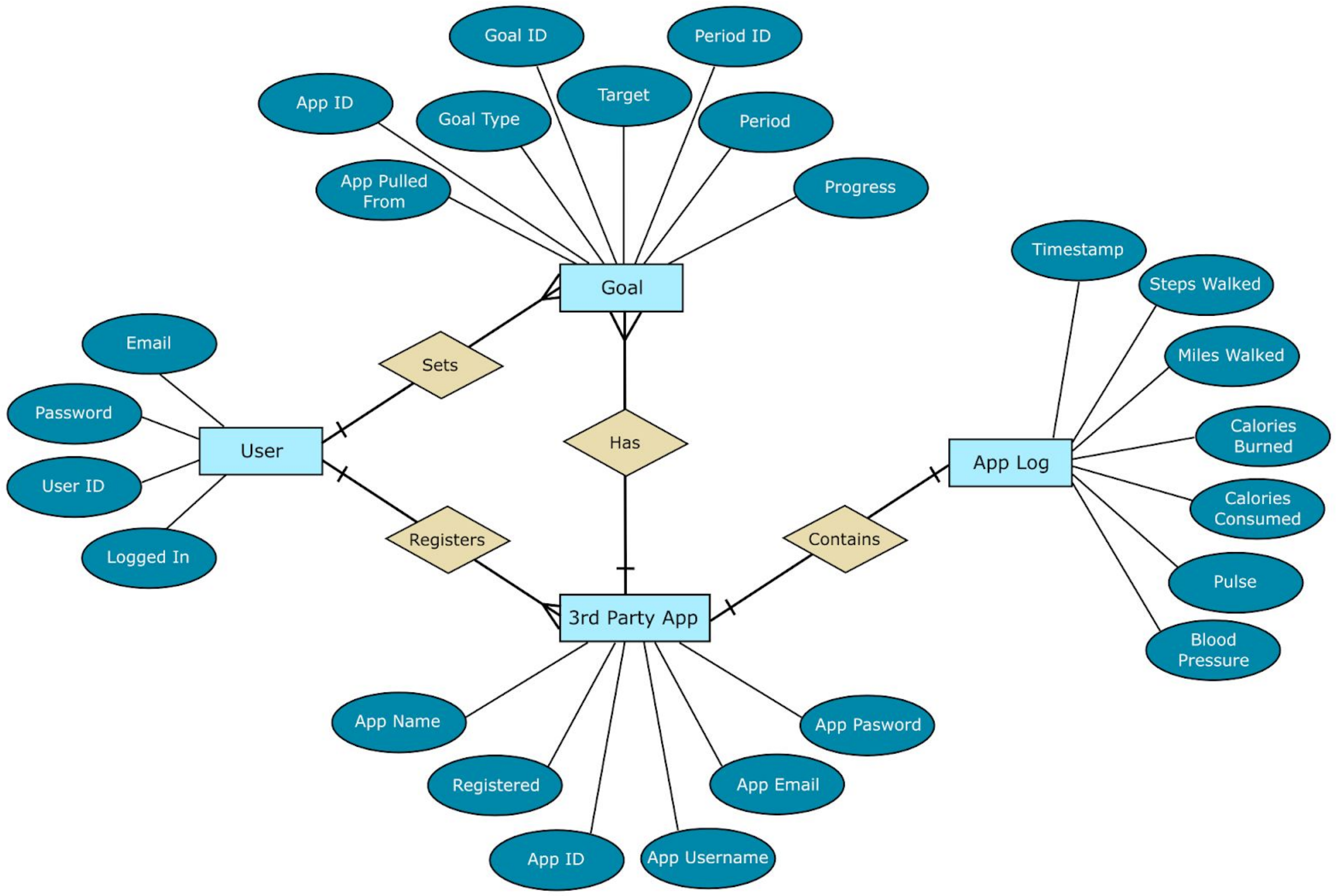
This module uses a two GUIs, being the Main Menu page and the Progress Graph page. When the user clicks on a goal on the Main Menu page, they will be taken to the Progress Graph page. This module will then try to gain access to HealthOut's database through the Microsoft Azure API. If unsuccessful, then then they will be take back to the Main Menu and a message will display telling the user to try again later. If successful, then this module will use the user's id to loop through the corresponding **CompatibleAppLogs table** and pull every instance of log data the user has stored there. This module will then use this log data to display a detailed graph of the user's progress towards achieving that goal. [1]

3.4 Concurrent process decomposition

Healthout has a single concurrent process, being the Update Logs module. All other modules will only run when triggered by the user and will only run while the user remains on the page where triggered. The Update Logs module however will run in the background, even while other modules are running in the foreground, checking to see if any apps the user has registered have updated their log data.

3.5 Data decomposition

Detailed descriptions of the data entities used by "HealthOut, outlining what types information will be stored and used by the mobile app.



3.5.1 User data entity

The User data entity holds the data related to a particular user's account. This data entity has four attributes, being the **logged_in, user_id, email**, and **password** attributes.

3.5.2 3rd Party App data entity

The 3rd Party App data entity holds the data related to a particular compatible 3rd party health app. This data entity has seven attributes, being the **app_id, app_name, registered**, **app_username**, **app_email**, and **app_password** attributes.

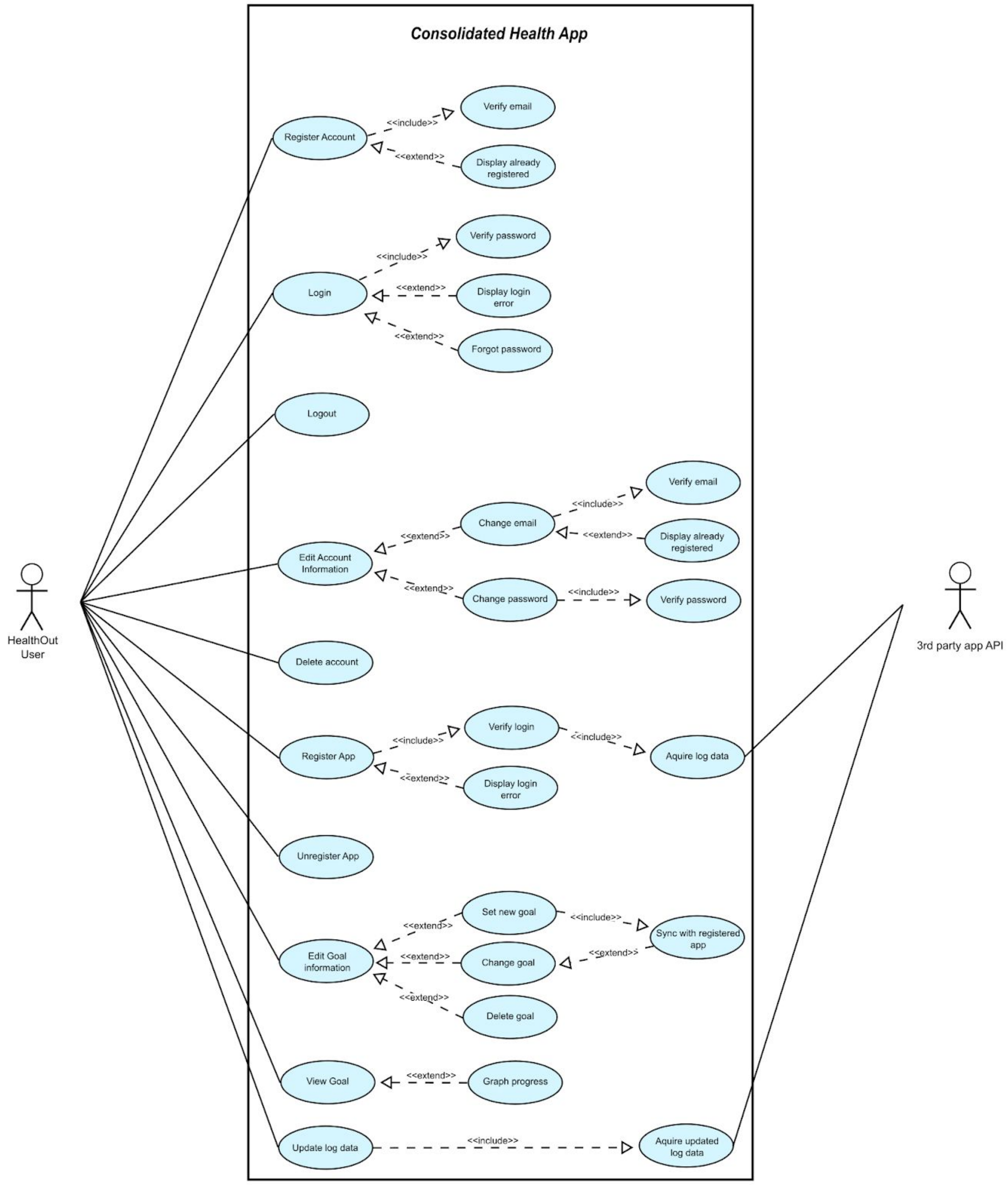
3.5.3 App Log data entity

The App Log data entity is a subclass of the 3rd Party App data entity. This data entity holds the log data related to the user's account for a particular registered 3rd party health app. This data entity has eight attributes, being the **steps_walked**, **miles_walked**, **calories_burned**, **calories_consumed**, **pulse**, **blood_pressure**, and **epoch_timestamp** attributes.

3.5.4 Goal data entity

The Goal data entity holds the data related to a particular goal. This data entity has nine attributes, being the **app_id, appPulledFrom**, **goal_id, goal_type**, **target_value, period_id, period_length**, and **progress** attributes.

3.6 Use Case Diagram


Consolidated Health App
Register Account
<<include>>
Verify email
<<extend>>
Display already registered
Login
<<include>>
Verify password
<<extend>>
Display login error
<<extend>>
Forgot password
Logout
Edit Account Information
<<extend>>
Change email
<<include>>
Verify email
<<extend>>
Display already registered
<<extend>>
Change password
<<include>>
Verify password
Delete account
Register App
<<include>>
Verify login
<<include>>
Aquire log data
<<extend>>
Display login error
Unregister App
Edit Goal information
<<extend>>
Set new goal
<<include>>
Sync with registered app
<<extend>>
Change goal
<<extend>>
<<extend>>
Delete goal
View Goal
<<extend>>
Graph progress
Update log data
<<include>>
Aquire updated log data
HealthOut User
3rd party app API

3.7 Use Cases

Use Case:	Register Account
Primary actor:	User
Goal in context:	To create an account for HealthOut
Precondition:	The user has already downloaded HealthOut
Trigger:	The user opens HealthOut, and clicks the “Register” button on the Login page.

Scenario:

1. User: maneuvers to Register Account page
2. User: enters their email
3. User: enters their desired password
4. User: enters their desired password again
5. User: clicks the “Sign Up” button
6. User: receives an email to activate their account
7. User: activates their account by following the link in activation email
8. App: accesses database through Microsoft Azure API
9. App: creates a new account for the user

Exceptions:

1. User is already registered to the app
2. Email is invalid
3. Password is invalid
4. Passwords do not match
5. API is down or unavailable

Priority:	Essential
When available:	First Increment
Frequency of use:	Most likely only once
Channel to actor:	Register Account page
Secondary actor:	Microsoft Azure API
Channels to secondary actor:	User Account subsystem

Open issues:

1. Should the database stores the wrong email or password
2. Should the user never receive an activation email

Use Case:	Recover Password
Primary actor:	User
Goal in context:	To recover a forgotten password to an account
Precondition:	The user has already registered an account for HealthOut
Trigger:	User clicks “Forgot Password?” text on Login page

Scenario:

1. User: enters their email into the Email prompt
2. User: clicks the “Continue” button
3. App: accesses database through Microsoft Azure API

4. App: finds password that corresponds to the account that the email belongs to
5. App: sends password to email
6. User: retrieves password

Exceptions:

1. User is not registered to HealthOut
2. Email is invalid
3. Microsoft Azure API is down or unavailable

Priority:	Essential
When available:	First Increment
Frequency of use:	Most likely either once, twice, or not at all
Channel to actor:	Login page
Secondary actor:	Microsoft Azure API
Channels to secondary actor:	User Account subsystem

Open issues:

1. Should HealthOut send the wrong password to the right email
2. Should HealthOut send the right password to the wrong email
3. Should HealthOut send the wrong password to the wrong email

---

Use Case:	Login
Primary actor:	User
Goal in context:	To login to HealthOut
Precondition:	The user has already registered an account for HealthOut
Trigger:	The user opens HealthOut while not logged into an account

Scenario:

1. User: maneuvers to Login page
2. User: enters their email
3. User: enters their password
4. User: clicks the “Login” button
5. App: accesses database through Microsoft Azure API
6. App: logs user into their HealthOut account

Exceptions:

1. User is not registered to HealthOut
2. Email is invalid
3. Incorrect password is entered
4. Microsoft Azure API is down or unavailable

Priority:	Essential
When available:	First Increment
Frequency of use:	Multiple times
Channel to actor:	Login page
Secondary actor:	Microsoft Azure API
Channels to secondary actor:	User Account subsystem

Open issues:

1. Should HealthOut not give access to a correct email and password
2. Should HealthOut give access to an incorrect email and password

---

Use Case:	Logout
Primary actor:	User
Goal in context:	To logout of HealthOut
Precondition:	The user has already logged in to HealthOut
Trigger:	The user clicks the hamburger(3 parallel lines) button on the main menu, then clicks the “Logout” button.

Scenario:

1. User: maneuvers to Main Menu page
2. User: clicks hamburger button
3. User: clicks “Logout” button
4. User: is prompted with “are you sure?”.
5. User: clicks the “Yes” button
6. App: logs user out of their HealthOut account

Exceptions:

1. none

Priority:	Essential
When available:	First Increment
Frequency of use:	Multiple times
Channel to actor:	Main Menu page
Secondary actor:	None
Channels to secondary actor:	None

Open issues:

1. Should HealthOut not log the user out

Use Case:	Change Email
Primary actor:	User
Goal in context:	To change the email to their HealthOut account
Precondition:	The user has already logged in to HealthOut
Trigger:	The user clicks the hamburger(3 parallel lines) button on the main menu, then clicks the “Edit Account” button.

Scenario:

1. User: maneuvers to Edit Account page
2. User: enters current password
3. User: enters desired new email
4. User: clicks the “Apply” button
5. App: accesses database through Microsoft Azure API
6. App: changes user’s email

Exceptions:

1. Incorrect current password
2. Invalid new email
3. Microsoft Azure API is down or unavailable

Priority:	Essential

When available:	First Increment
Frequency of use:	Most likely either once, twice, or not at all
Channel to actor:	Edit Account page
Secondary actor:	Microsoft Azure API
Channels to secondary actor:	User Account subsystem

Open issues:

1. Should HealthOut change their email despite an incorrect current password
2. Should HealthOut store an incorrect new email

Use Case:	Change Password
Primary actor:	User
Goal in context:	To change the password to their HealthOut account
Precondition:	The user has already logged in to HealthOut
Trigger:	The user clicks the hamburger(3 parallel lines) button on the main menu, then clicks the “Edit Account” button.

Scenario:

1. User: maneuvers to Edit Account page
2. User: enters current password
3. User: enters desired new password
4. User: enters desired new password again
5. User: clicks the “Apply” button
6. App: accesses database through Microsoft Azure API
7. App: changes user’s password

Exceptions:

1. Incorrect current password
2. Invalid new password
3. New passwords do not match
4. Microsoft Azure API is down or unavailable

Priority:	Essential
When available:	First Increment
Frequency of use:	Most likely either once, twice, or not at all
Channel to actor:	Edit Account page
Secondary actor:	Microsoft Azure API
Channels to secondary actor:	User Account subsystem

Open issues:

1. Should HealthOut change their password despite an incorrect current password
2. Should HealthOut change their password despite the new passwords not matching
3. Should HealthOut store an incorrect new password

Use Case:	Delete Account
Primary actor:	User
Goal in context:	To delete their HealthOut account

Precondition:	The user has already logged in to HealthOut
Trigger:	The user clicks the hamburger(3 parallel lines) button on the main menu, then clicks the “Delete Account” button.

Scenario:

1. User: maneuvers to Main Menu page
2. User: clicks hamburger button
3. User: clicks “Delete Account” button
4. User: is prompted with “are you sure?”.
5. User: clicks the “Yes” button
6. App: accesses database through Microsoft Azure API
7. App: deletes user’s account

Exceptions:

1. Microsoft Azure API is down or unavailable

Priority:	Essential
When available:	First Increment
Frequency of use:	Most likely either once or not at all
Channel to actor:	Main Menu page
Secondary actor:	Microsoft Azure API
Channels to secondary actor:	User Account subsystem

Open issues:

1. Should HealthOut not delete the account
2. Should HealthOut delete the wrong account

Use Case:	Register App
Primary actor:	User
Goal in context:	To register a 3rd party health app to HealthOut
Precondition:	The user must be logged in and have an account with the 3rd party health app they wish to register.
Trigger:	The user opens the Resiger App page and clicks a switch button next to an unregistered app

Scenario:

1. User: maneuvers to Register App page
2. User: clicks switch button next to desired app
3. User: enters login information for the desired app
4. User: clicks the “Continue” button
5. App: contacts the desired app’s API
6. 3rd party app’s API: sends app log data to HealthOut database

Exceptions:

1. User does not have an account with the desired app
2. 3rd party app’s API is down or unavailable

Priority:	Essential
When available:	Second Increment
Frequency of use:	Many times
Channel to actor:	Register app page
Secondary actor:	3rd party app API

Channels to secondary actor: Compatible Apps subsystem
Open issues:

1. Desired 3rd party health app will not give us access to their API
2. HealthOut sends wrong login info to the desired app's API
3. HealthOut sends users login info to the wrong app's API

Use Case:	Unregister App
Primary actor:	User
Goal in context:	To unregister a 3rd party health app from HealthOut
Precondition:	The user must be logged in and have at least one registered 3rd party health app with HealthOut
Trigger:	The user opens the Resiger App page and clicks a switch button next to an unregistered app

Scenario:

1. User: maneuvers to Register App page
2. User: clicks switch button next to desired app
3. App: changes app to unregistered

Exceptions:

1. The user tries to unregister an app that is not registered

Priority:	Essential
When available:	Second Increment
Frequency of use:	Many times
Channel to actor:	Register app page
Secondary actor:	None
Channels to secondary actor:	None

Open issues:

1. HealthOut does not unregister app
2. HealthOut unregisters the wrong app

Use Case:	Set Goal
Primary actor:	User
Goal in context:	To add a new goal to Healthout
Precondition:	The user must be logged in, and have at least one 3rd party health app registered to HealthOut.
Trigger:	The user opens the Edit Goals page and clicks the "Add new" button

Scenario:

1. User: maneuvers to the Specify Goal page
2. User: selects the 3rd party health app they wish to pull data from
3. User: selects a goal type for that app
4. User: chooses a target for that goal type
5. User: chooses a time period for the goal
6. User: clicks the "Apply" button
7. App: adds goal

Exceptions:
1. User does not have any 3rd party health apps registered
2. User does not select a registered health app
3. User does not select a goal type
4. User does not enter a target for the goal type
5. User enters an invalid target for the goal type
6. User does not select a time period for goal

Priority:	Essential
When available:	Second Increment
Frequency of use:	Multiple times
Channel to actor:	Specify Goal page
Secondary actor:	None
Channels to secondary actor:	None

Open issues:
1. HealthOut stores the wrong information for goal
2. Should there be a limit to how many goals can be created on one account

---

Use Case:	Change Goal
Primary actor:	User
Goal in context:	To change a goal on HealthOut
Precondition:	The user must be logged in, and already have a goal created
Trigger:	The user opens the Edit Goals page and clicks on a goal

Scenario:
1. User: maneuvers to the Specify Goal page
2. User: may change which app data is being pulled from for goal
3. Users: may change the goal type for goal
4. User: may change the target for goal
5. User: may change the time period for goal
6. User: clicks the “Apply” button
7. App: changes goal

Exceptions:
1. User changes target for goal to an invalid input

Priority:	Essential
When available:	Second Increment
Frequency of use:	Multiple times
Channel to actor:	Specify Goal page
Secondary actor:	None
Channels to secondary actor:	None

Open issues:
1. HealthOut stores the wrong information for goal

Use Case:	Remove Goal
Primary actor:	User

Goal in context:	To remove a goal on HealthOut
Precondition:	The user must be logged in, and already have a goal created
Trigger:	The user opens the Edit Goals page and clicks the “Remove” button

Scenario:

1. User: maneuvers to the Remove Goals page
2. User: may remove multiple goals at once by clicking more than one checkbox next to each goal
3. User clicks the “Remove” button
4. App: removes those goals

Exceptions:

1. User does not click any checkboxes

Priority:	Essential
When available:	Second Increment
Frequency of use:	Multiple times
Channel to actor:	Remove Goals page
Secondary actor:	None
Channels to secondary actor:	None

Open issues:

1. HealthOut does not remove the goals the user select for removal
2. HealthOut removes goals the user did not select for removal

---

Use Case:	View Goals
Primary actor:	User
Goal in context:	Allows the user to view their health goals
Precondition:	The user must be logged in and have set at least one goal
Trigger:	The user opens the Main Menu page

Scenario:

1. App: displays all goals

Exceptions:

1. User does not have any goals set

Priority:	Essential
When available:	Second Increment
Frequency of use:	Multiple times
Channel to actor:	Main Menu page
Secondary actor:	None
Channels to secondary actor:	None

Open issues:

1. HealthOut does not display all goals
2. HealthOut displays goals incorrectly
3. HealthOut displays duplicates of goals

Use Case:	View Progress Graph for Goal

Primary actor:	User
Goal in context:	Allows the user to view a progress graph for a health goal
Precondition:	The user must be logged in and have set at least one goal
Trigger:	The user opens the Main Menu page and clicks on a goal

Scenario:

1. App: displays a progress graph for that particular goal

Exceptions:

1. None

Priority:	Essential
When available:	Second Increment
Frequency of use:	Multiple times
Channel to actor:	Progress Graph page
Secondary actor:	None
Channels to secondary actor:	None

Open issues:

1. HealthOut does not display progress graph for any goal
2. HealthOut displays progress graph for wrong goal
3. HealthOut displays progress graph with wrong log data for goal

---

Use Case:	Update Log data
Primary actor:	User
Goal in context:	Allows the user to request an update for the log data to all their health goals
Precondition:	The user must be logged in and have set at least one goal
Trigger:	The user opens the Main Menu page and clicks the "Update" button

Scenario:

1. App: contacts the API of each app that the user has a goal set for
2. 3rd party app's API: sends app log data to HealthOut database
3. App: refreshes the goals on the main menu with any newly collected data

Exceptions:

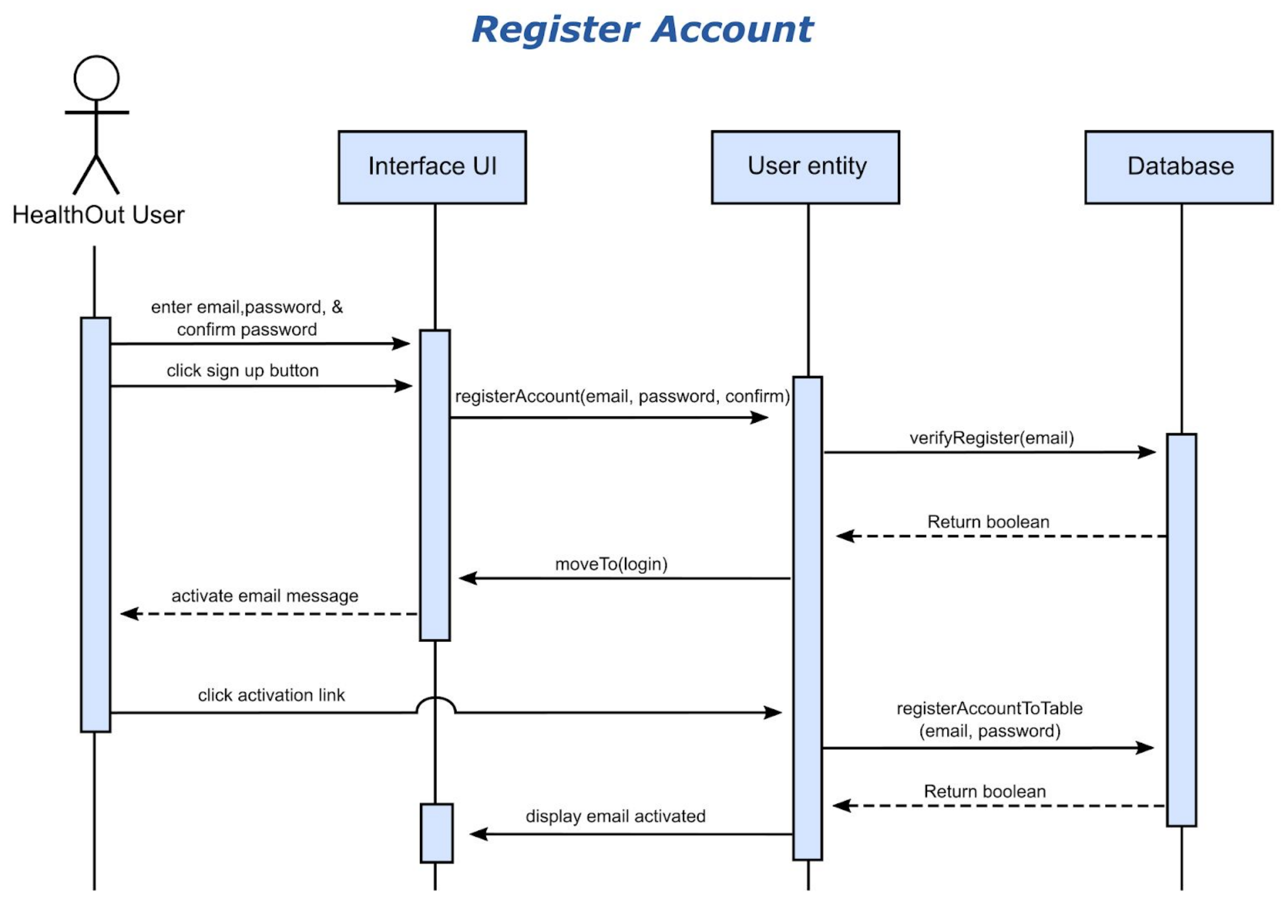
1. The user's login info has changed for a 3rd party health app that the user has a goal set for
2. 3rd party app's API is down or unavailable

Priority:	Essential
When available:	Second Increment
Frequency of use:	Multiple times
Channel to actor:	Main Menu page
Secondary actor:	Register app page
Secondary actor:	3rd party app API

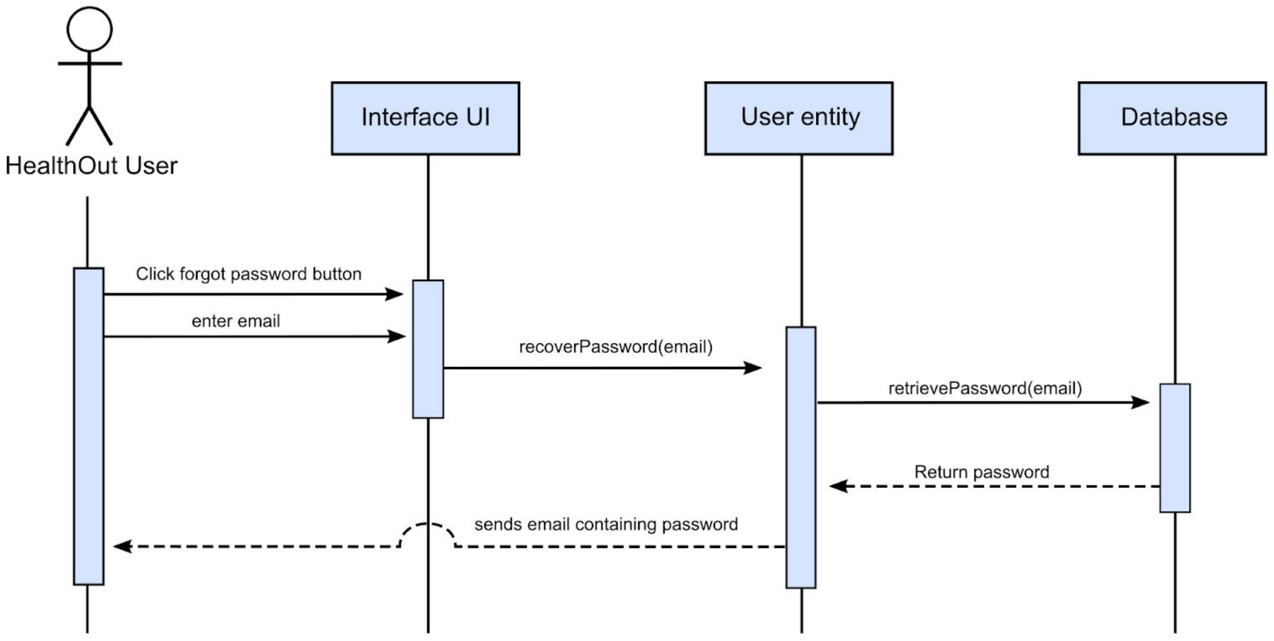
Open issues:

1. Desired 3rd party health app will not give us access to their API
2. HealthOut sends wrong login info to the desired app's API
3. HealthOut sends users login info to the wrong app's API

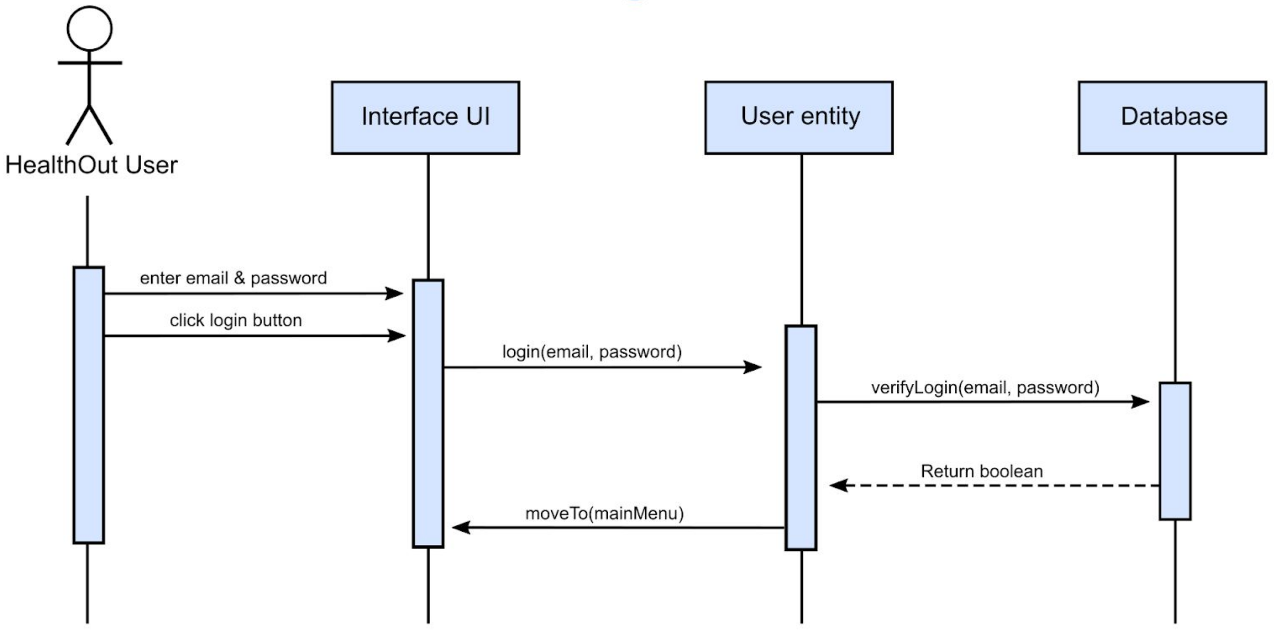
3.8 Sequence Diagrams



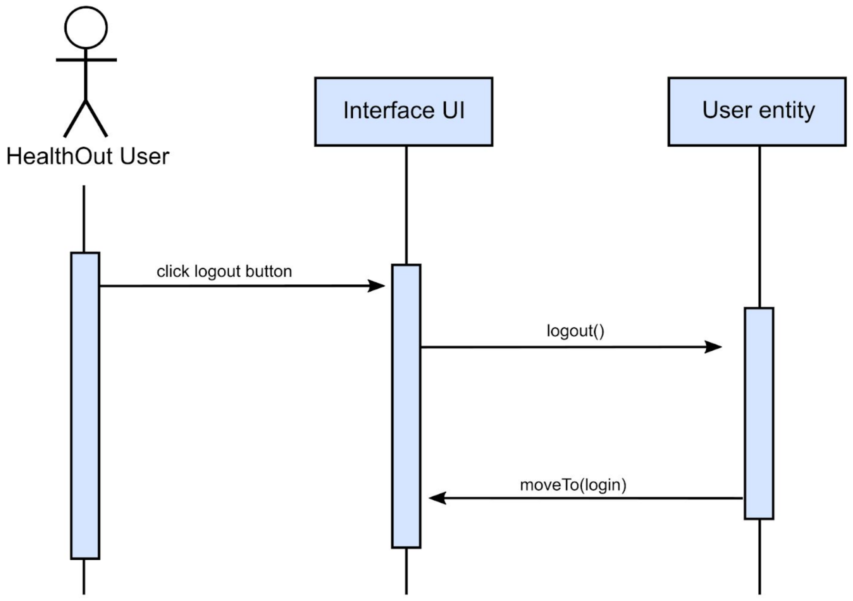
Recover Password



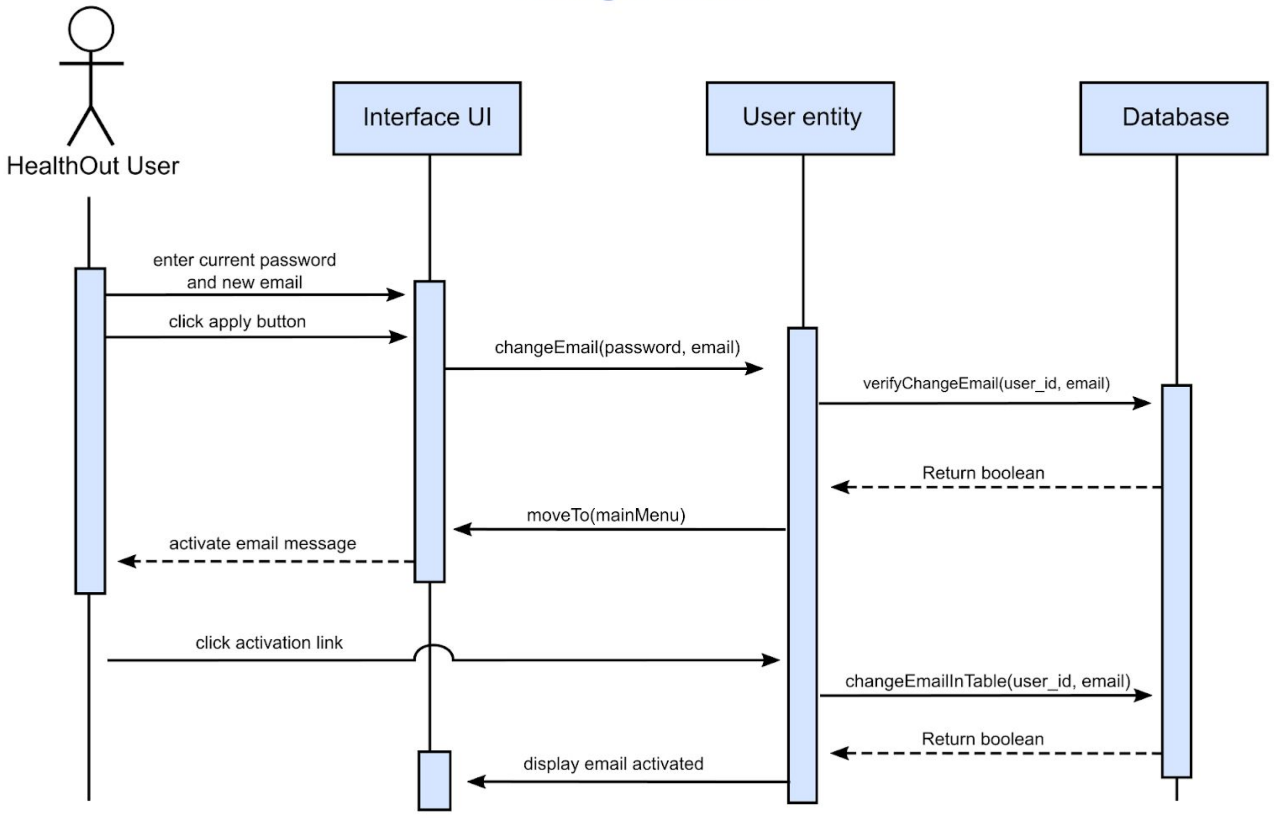
Login



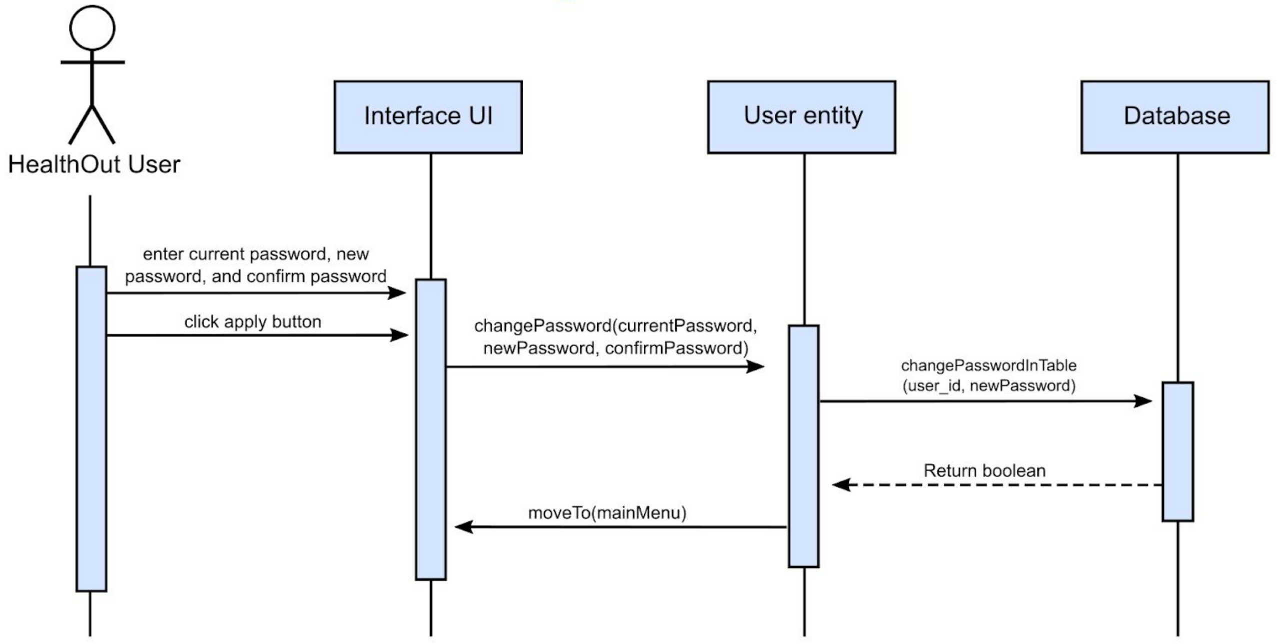
Logout



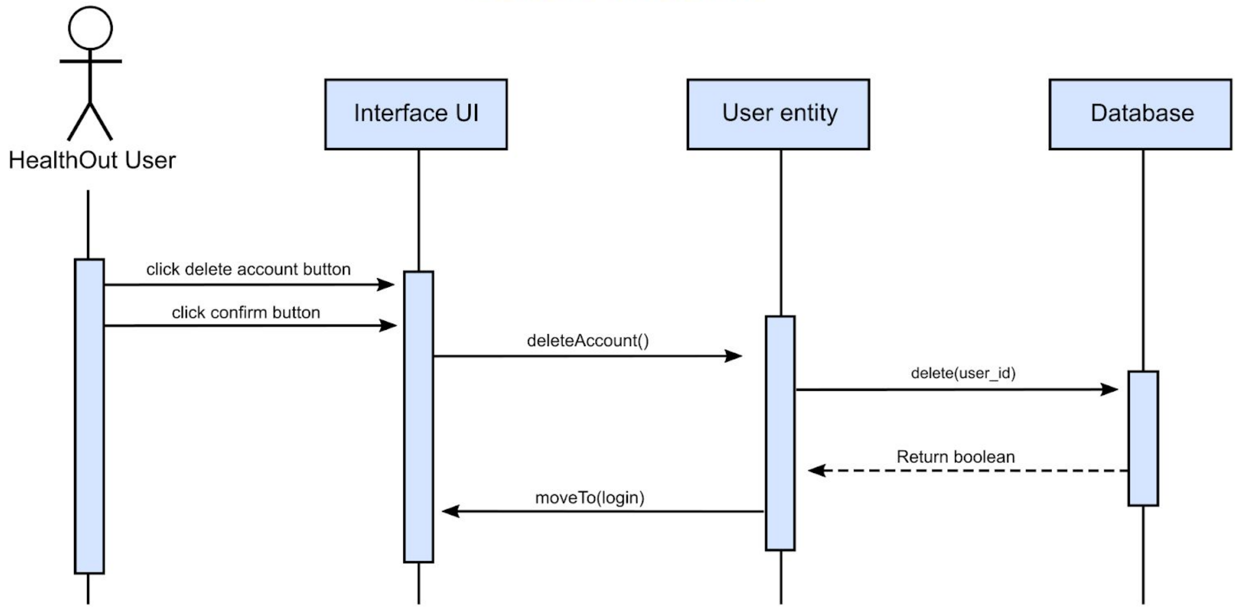
Change Email



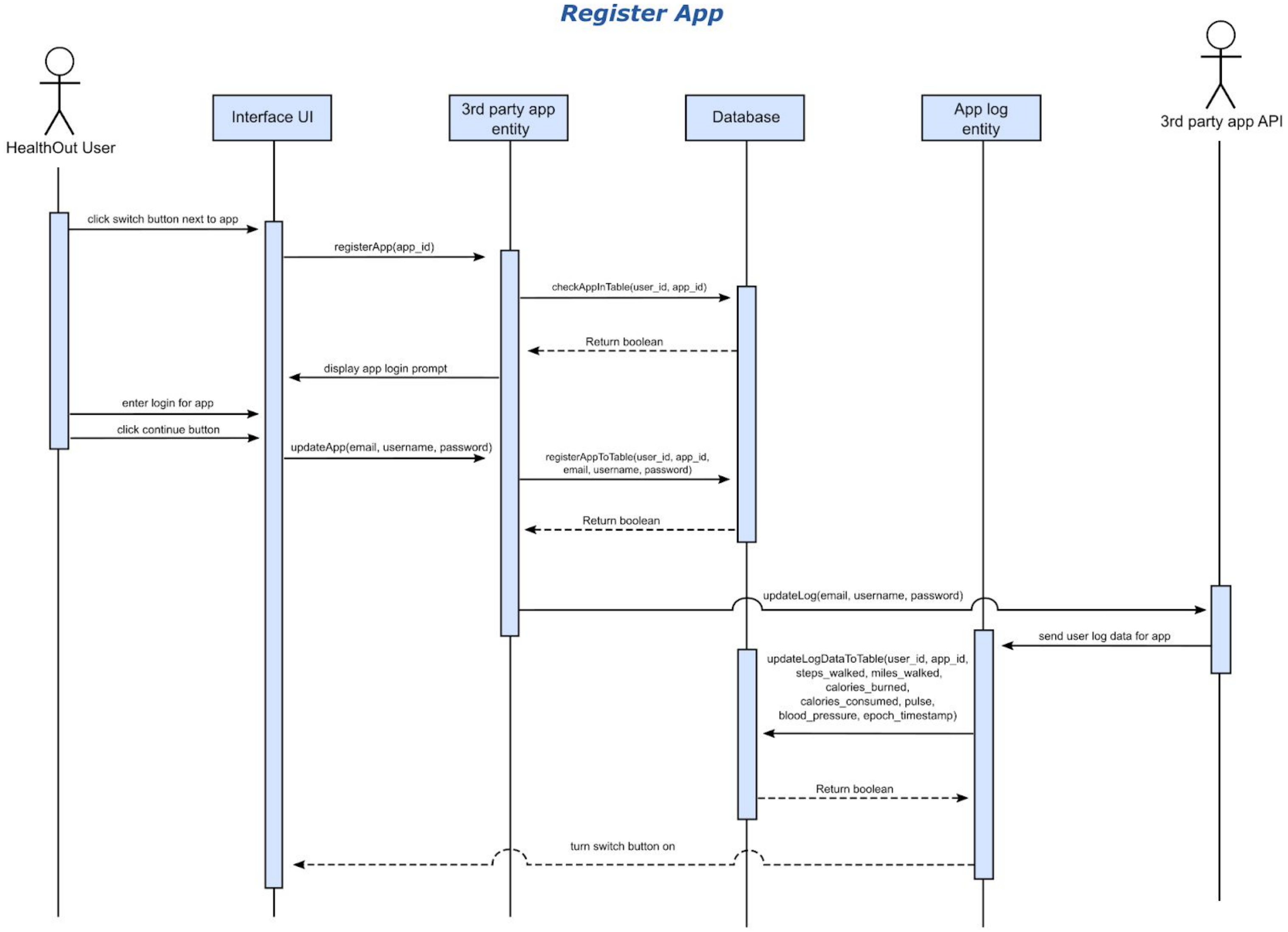
Change Password


HealthOut User
Interface UI
User entity
Database
enter current password, new password, and confirm password
click apply button
changePassword(currentPassword, newPassword, confirmPassword)
changePasswordInTable (user_id, newPassword)
Return boolean
moveTo(mainMenu)

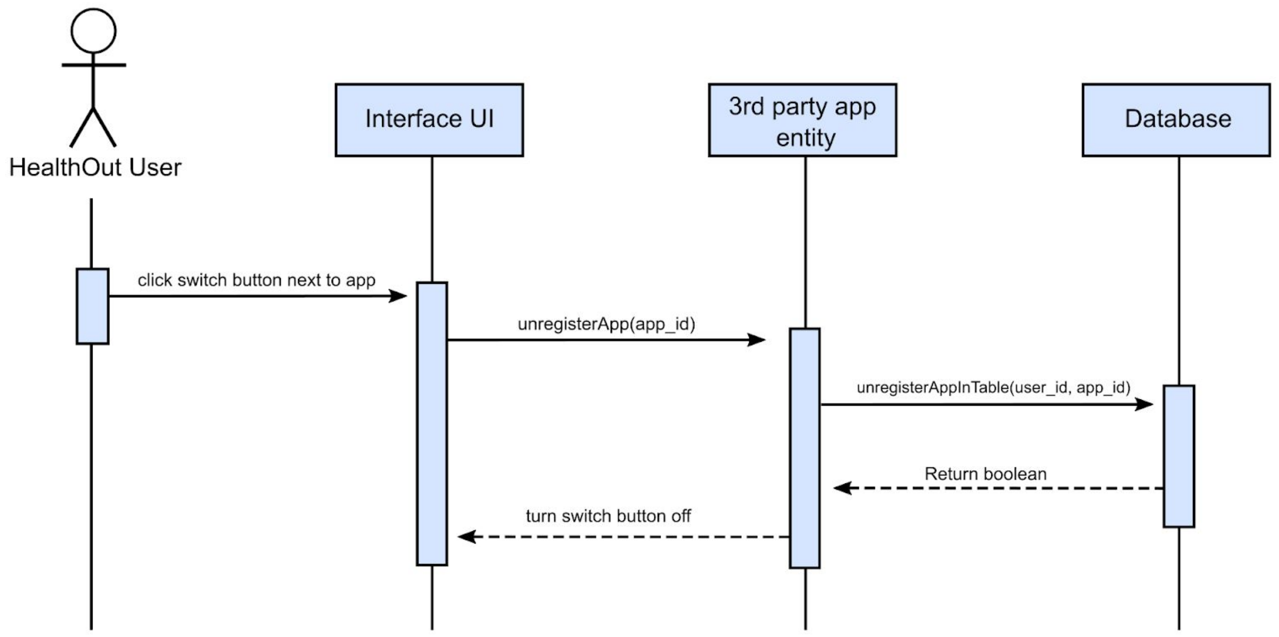
Delete Account



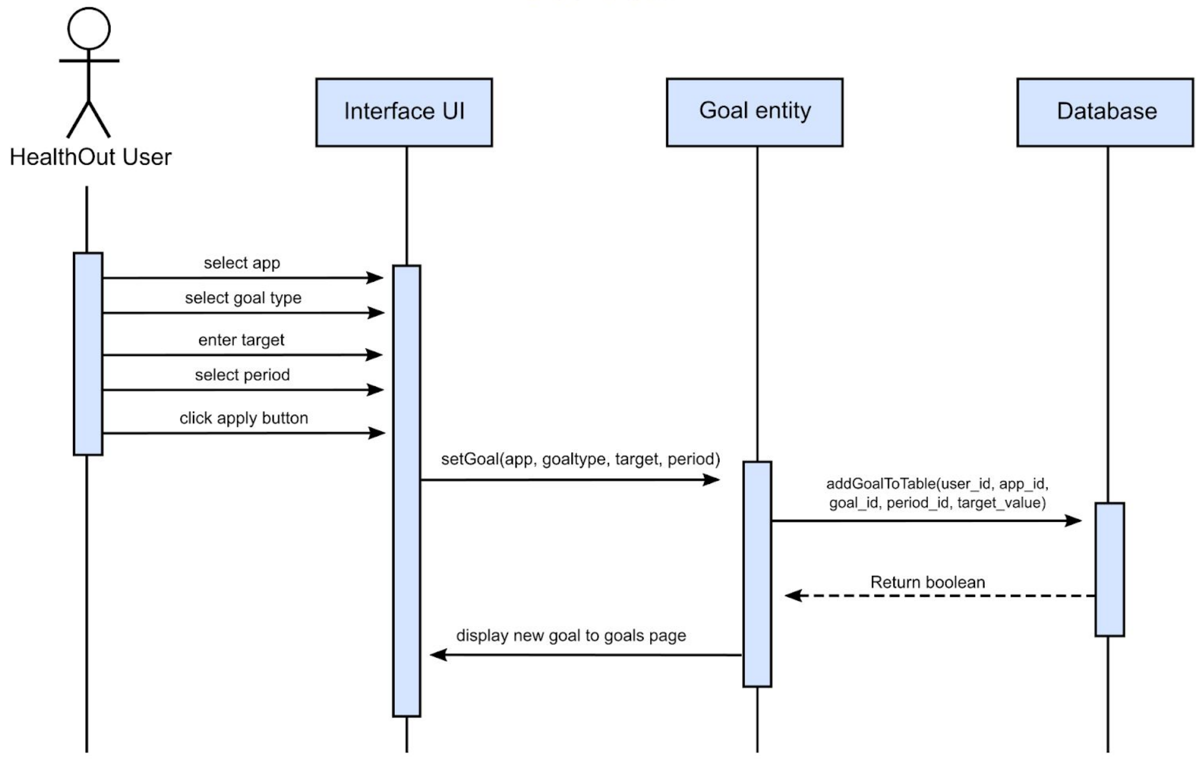
Register App


HealthOut User
Interface UI
3rd party app entity
Database
App log entity
3rd party app API
click switch button next to app
registerApp(app_id)
checkAppInTable(user_id, app_id)
Return boolean
display app login prompt
enter login for app
click continue button
updateApp(email, username, password)
registerAppToTable(user_id, app_id, email, username, password)
Return boolean
updateLog(email, username, password)
send user log data for app
updateLogDataToTable(user_id, app_id, steps_walked, miles_walked, calories_burned, calories_consumed, pulse, blood_pressure, epoch_timestamp)
Return boolean
turn switch button on

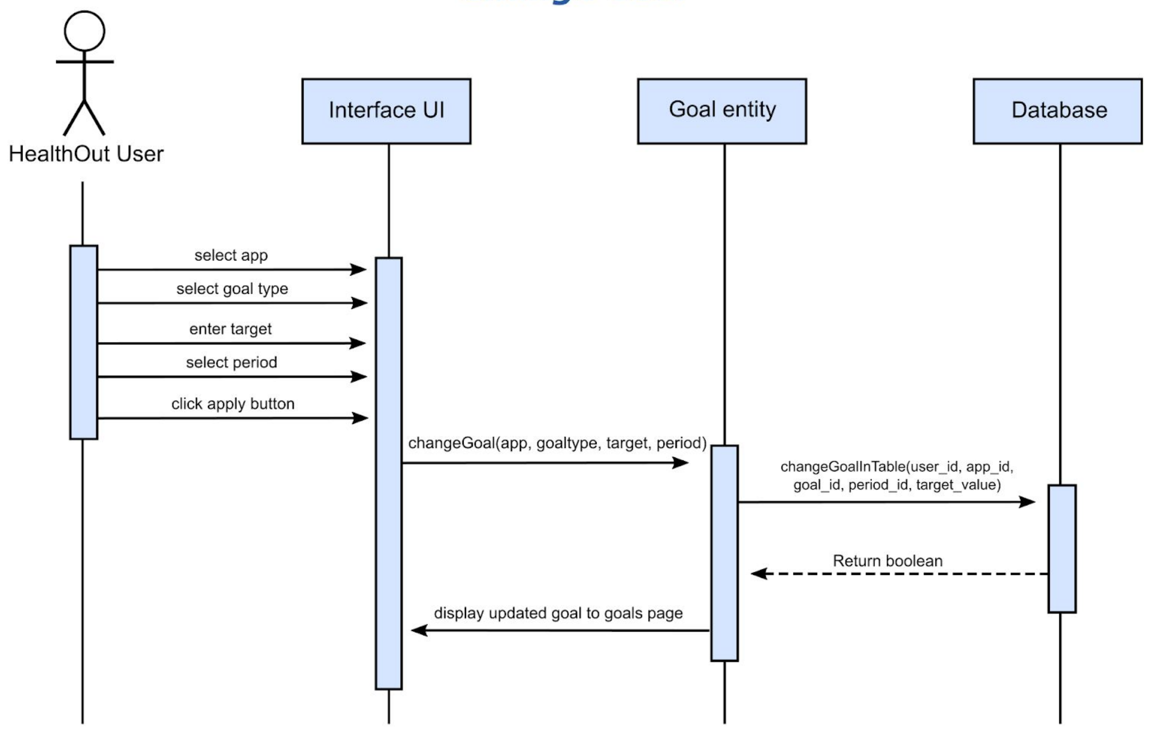
Unregister App



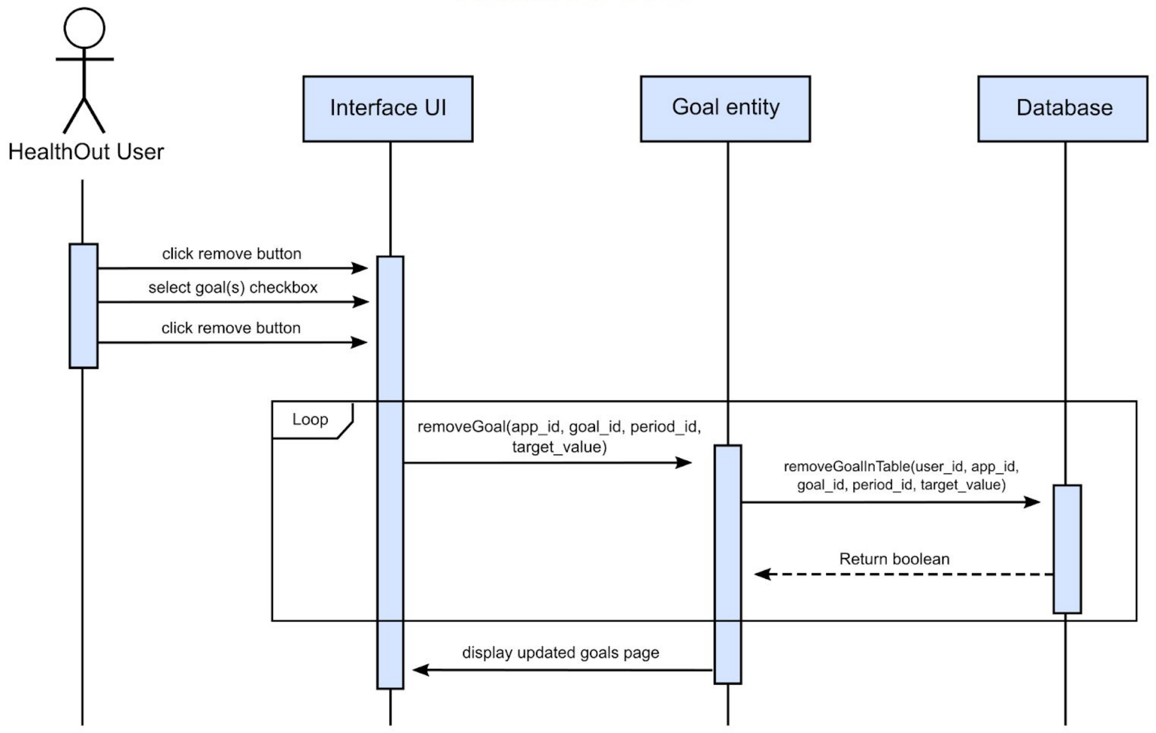
Set Goal


HealthOut User
Interface UI
Goal entity
Database
select app
select goal type
enter target
select period
click apply button
setGoal(app, goaltype, target, period)
addGoalToTable(user_id, app_id, goal_id, period_id, target_value)
Return boolean
display new goal to goals page

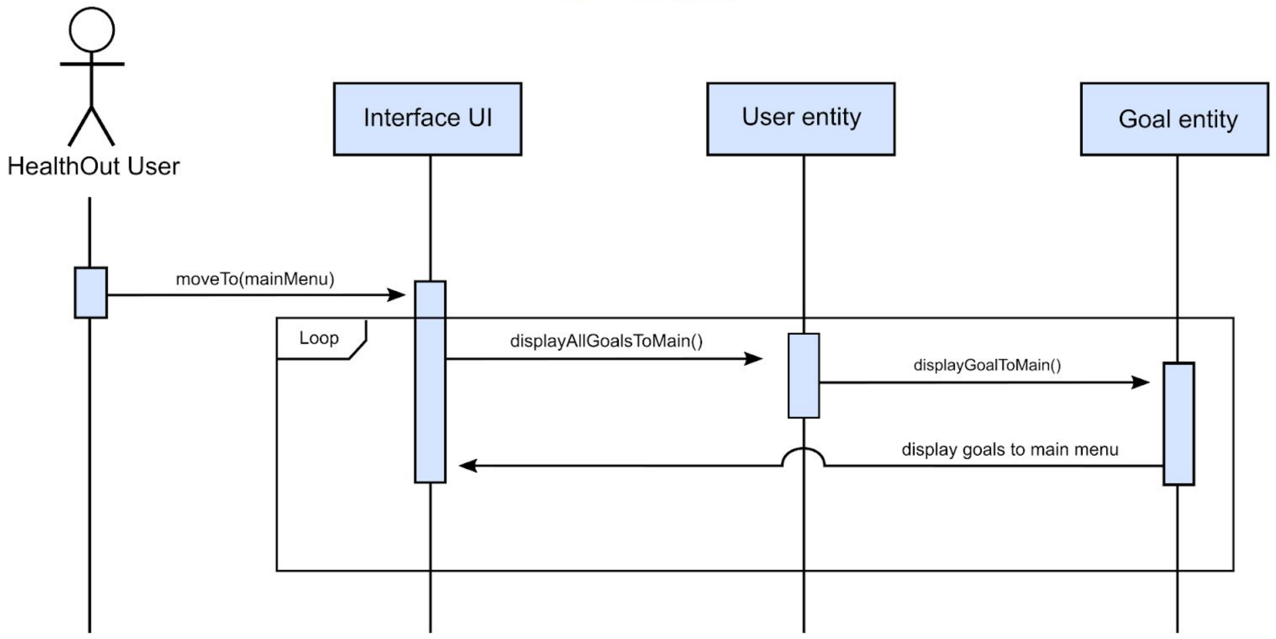
Change Goal


HealthOut User
Interface UI
Goal entity
Database
select app
select goal type
enter target
select period
click apply button
changeGoal(app, goaltype, target, period)
changeGoalInTable(user_id, app_id, goal_id, period_id, target_value)
Return boolean
display updated goal to goals page

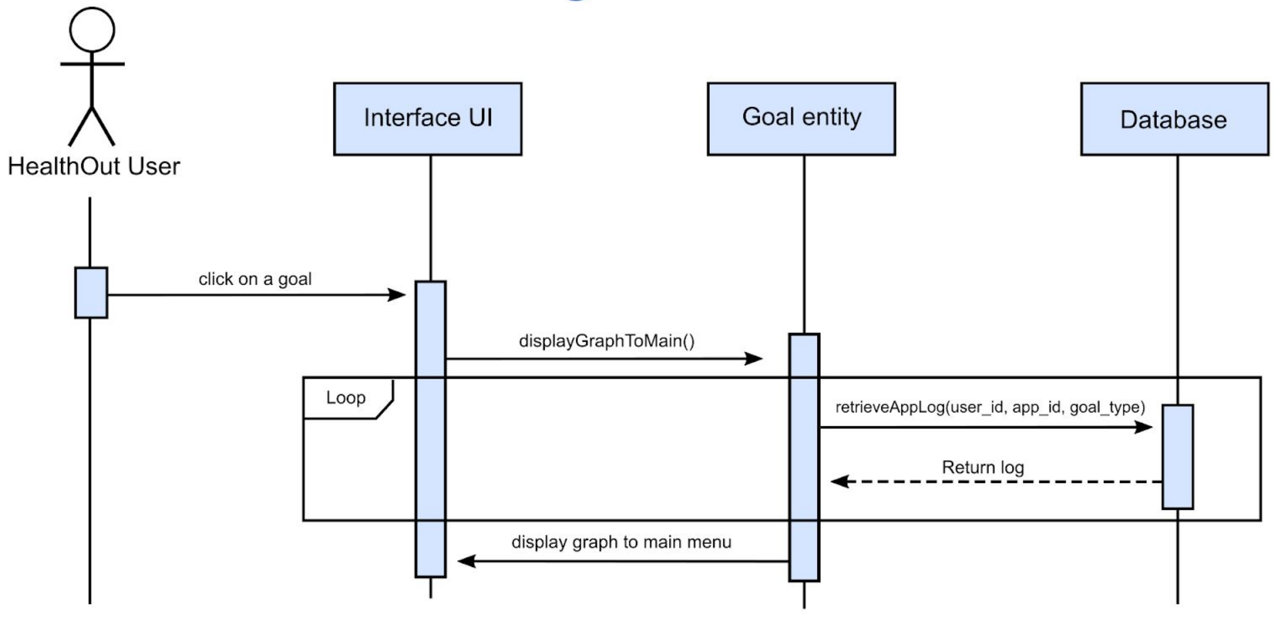
Remove Goal


HealthOut User
Interface UI
Goal entity
Database
click remove button
select goal(s) checkbox
click remove button
Loop
removeGoal(app_id, goal_id, period_id, target_value)
removeGoalInTable(user_id, app_id, goal_id, period_id, target_value)
Return boolean
display updated goals page

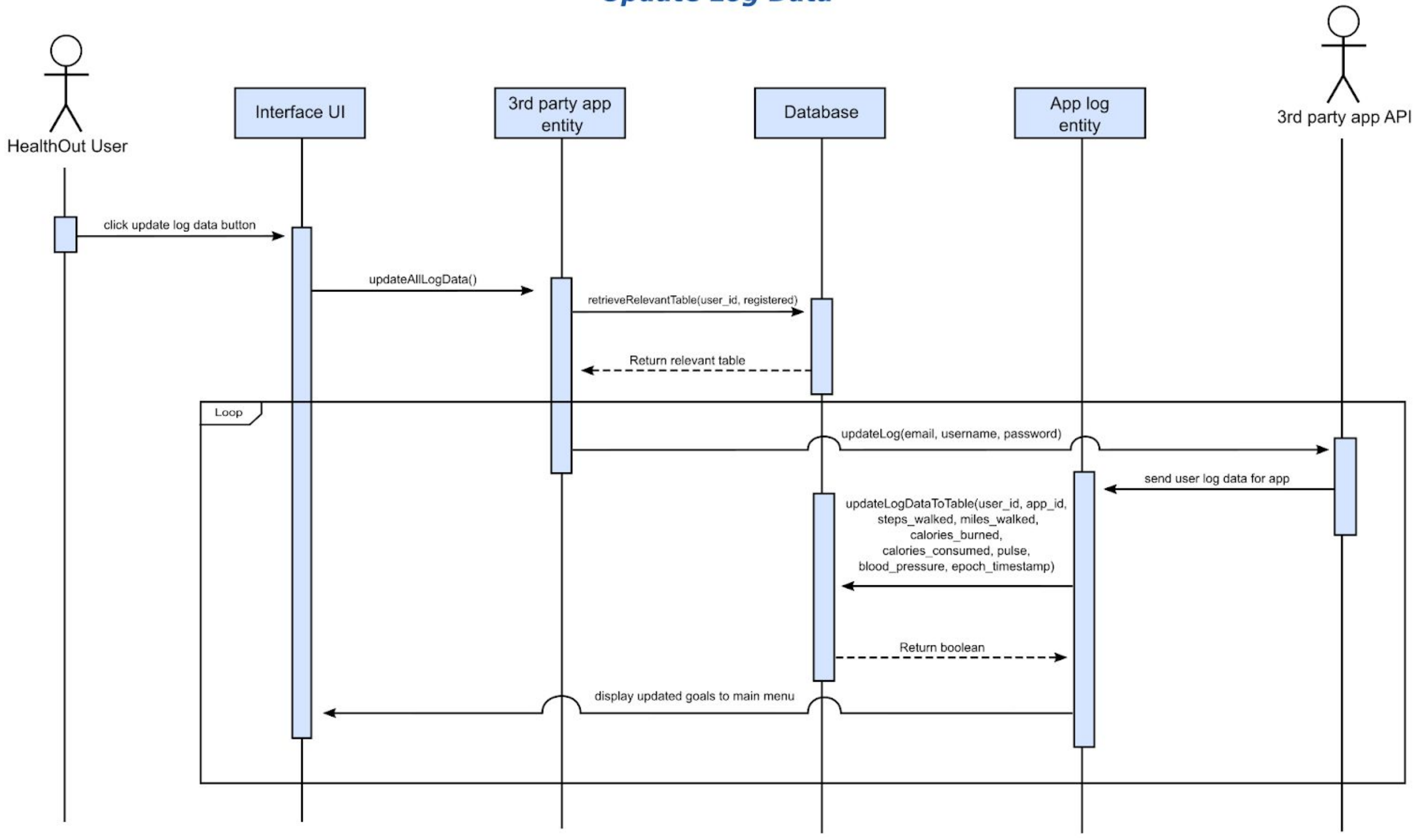
View Goals



View Progress for a Goal

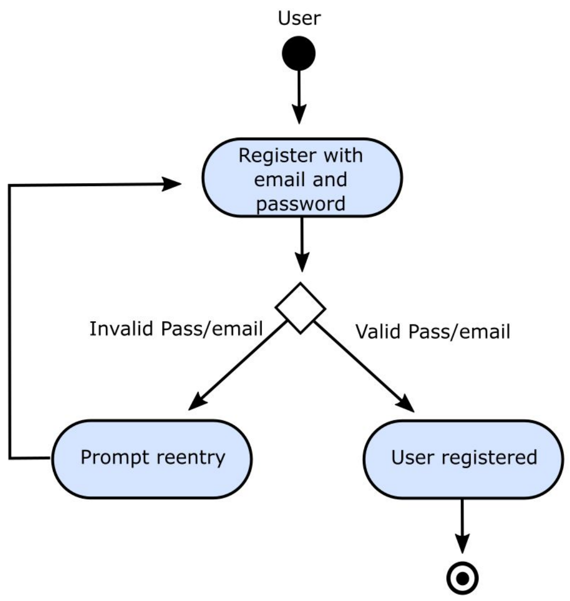


Update Log Data

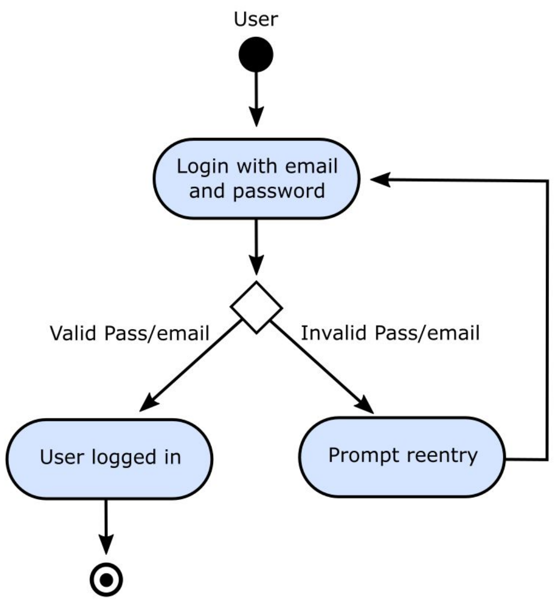

HealthOut User
Interface UI
3rd party app entity
Database
App log entity
3rd party app API
click update log data button
updateAllLogData()
retrieveRelevantTable(user_id, registered)
Return relevant table
Loop
updateLog(email, username, password)
send user log data for app
updateLogDataToTable(user_id, app_id, steps_walked, miles_walked, calories_burned, calories_consumed, pulse, blood_pressure, epoch_timestamp)
Return boolean
display updated goals to main menu

3.9 Activity Diagrams

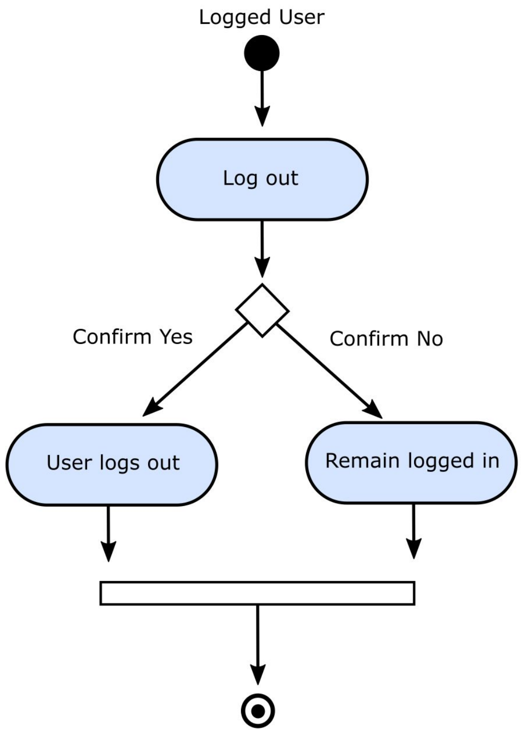
Register Account



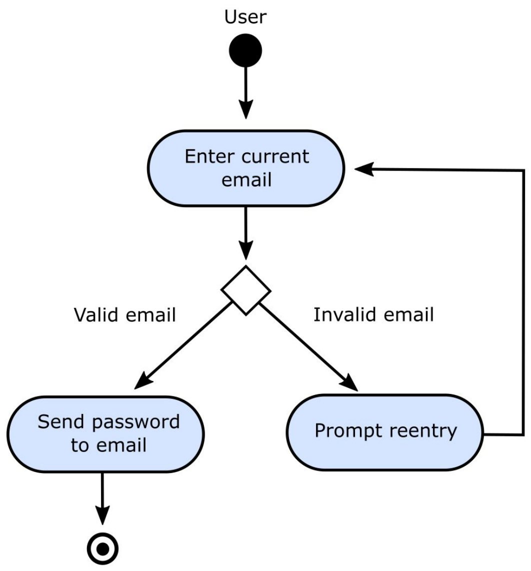
Login



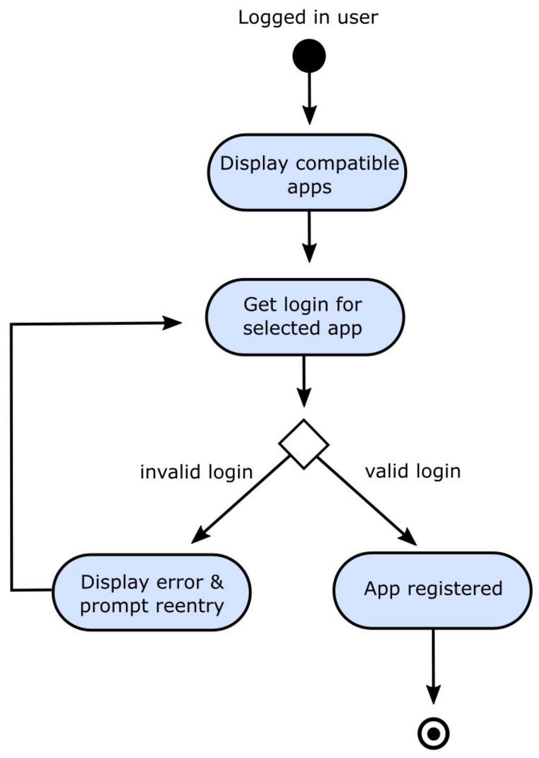
Log out



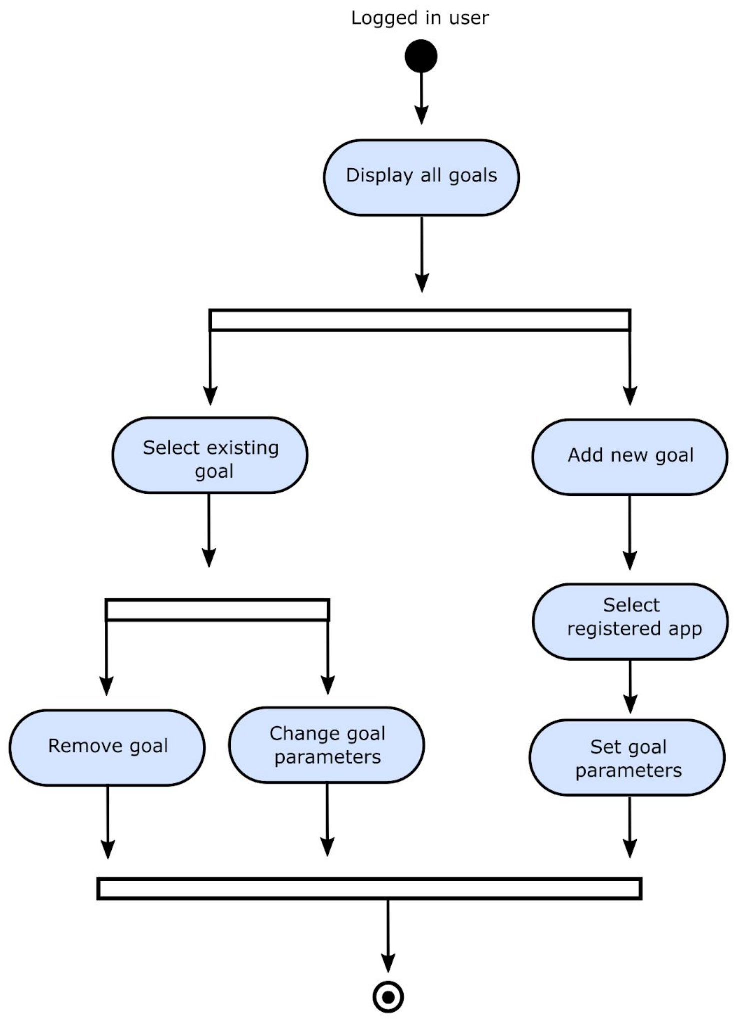
Recover password



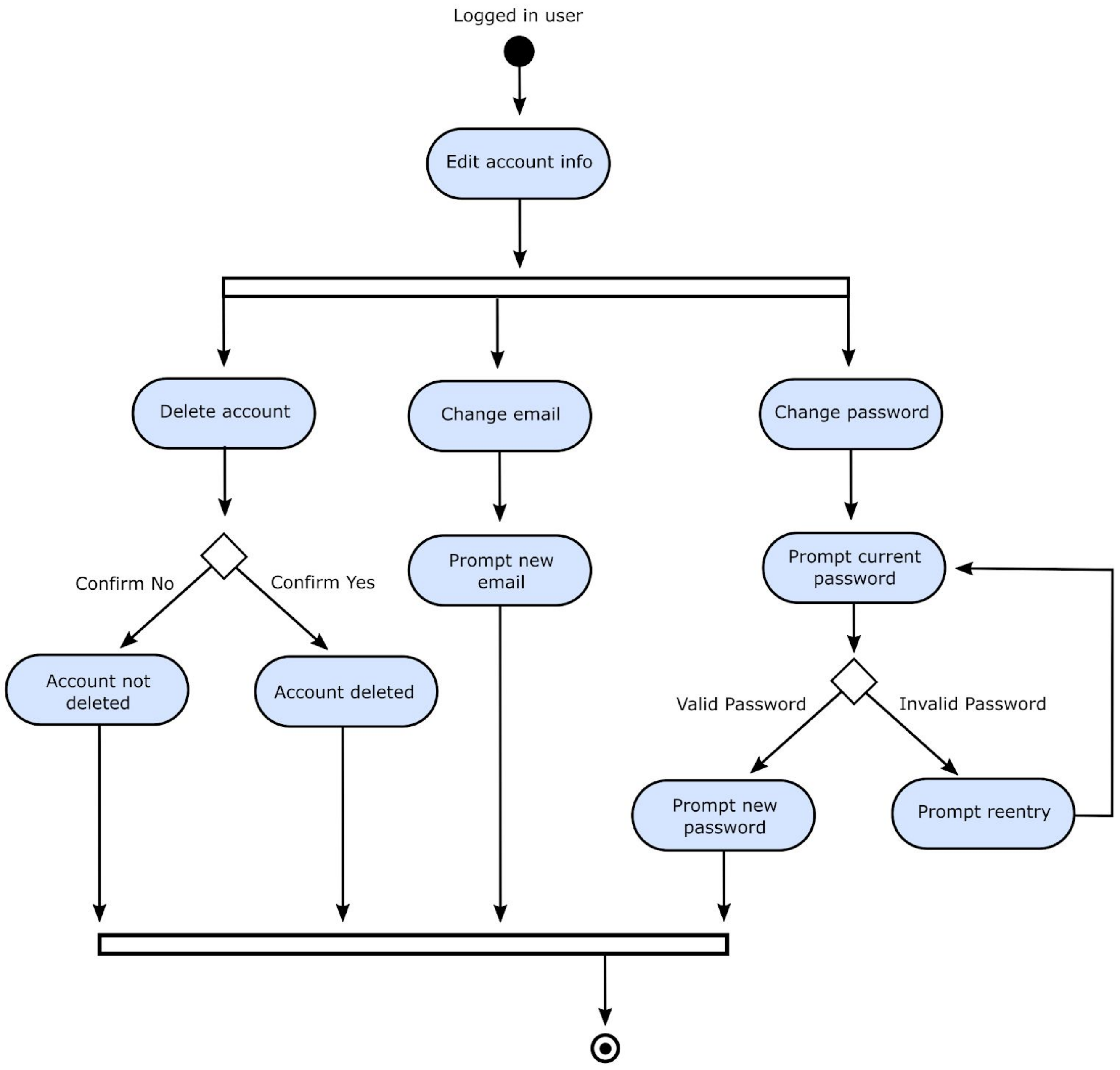
Register App



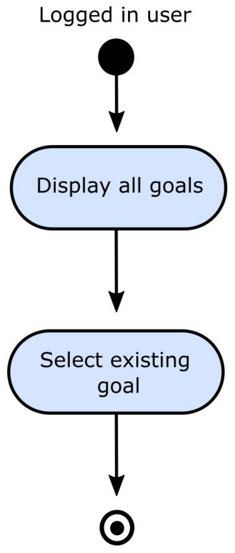
Edit Goal



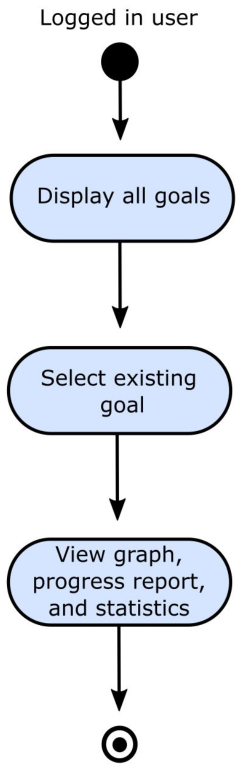
Edit Account Information



View Goal



View Graph



4. Dependency description

4.1 Intermodule dependencies

Describes the dependencies each module of HealthOut has with one another, as some modules need other modules to have completed, or at least ran, before they themselves can begin running or complete.

4.1.1 Register Account intermodule dependencies

The Register Account module is codependent upon the Activate Email module. This module will need to call the Activate Email module to send the user an activation email to their email address. When the Activation Email module returns true after the user clicks their activation link inside their activation email, then the Register Account module can continue with its process.

4.1.2 Activate Email intermodule dependencies

This Activation Email module is codependent upon either the Register Account module or the Change Email module. This module is called by both modules for the user to activate an email address, and returns true to them after they have done so.

4.1.3 Delete Account intermodule dependencies

The Delete Account module is dependent upon the Register Account and the Login modules. This module requires that the user first have registered an account through the Register Account module, and have logged in through the Login module.

4.1.4 Recover Password intermodule dependencies

The Recover Password module is dependent upon the Register Account module. This module requires that the user first have registered an account with HealthOut through the Register Account module.

4.1.5 Login intermodule dependencies

The Login module is dependent upon the Register Account module. This module requires that the user first have registered an account with HealthOut through the Register Account module.

4.1.6 Logout intermodule dependencies

The Logout module is dependent upon the Register Account and the Login modules. This module requires that the user first have registered an account through the Register Account module, and have logged in through the Login module.

4.1.7 Change Email intermodule dependencies

The Change Email module is dependent upon the Register Account and the Login modules. This module requires that the user first have registered an account through the Register Account module, and have logged in through the Login module. This module will need to call the Activate Email module to send the user an activation email to their newly desired email address. When the Activation Email module returns true after the user clicks their activation link inside their activation email, then the Change Email module can continue with its process.

4.1.8 Change Password intermodule dependencies

The Change Password module is dependent upon the Register Account and the Login modules. This module requires that the user first have registered an account through the Register Account module, and have logged in through the Login module.

4.1.9 Display 3rd Party Apps intermodule dependencies

The Display 3rd Party Apps module is dependent upon the Register Account and the Login modules. This module requires that the user first have registered an account through the Register Account module, and have logged in through the Login module.

4.1.10 Register App intermodule dependencies

The Register App module depends upon the Register Account, Login, and the Display 3rd Party Apps modules. This module requires that the user first have registered an account through the Register Account module, logged in through the Login module, and have had their registered/unregistered 3rd party health apps displayed on the Register Apps page through the Display 3rd Party Apps module.

4.1.11 Unregister App intermodule dependencies

The unregister App module depends upon the Register Account, Login, and the Display 3rd Party Apps modules. This module requires that the user first have registered an account through the Register Account module, logged in through the Login module, and have had their registered/unregistered 3rd party health apps displayed on the Register Apps page through the Display 3rd Party Apps module.

4.1.12 Update Logs intermodule dependencies

The update Logs module depends upon the Register Account, Login, and the Register App modules. This module requires that the user first have registered an account through the Register Account module, logged in through the Login module, and have at least one 3rd party health app registered to HealthOut through the Register App module.

4.1.13 Display Goals to Edit intermodule dependencies

The Display Goals to Edit module depends upon the Register Account, Login, and the Register App modules. This module requires that the user first have registered an account through the Register Account module, have log through the Login module, and have at least one 3rd party health app registered through the Register App module.

4.1.14 Set Goal intermodule dependencies

The Set Goal module depends upon the Register Account, Login, Register App, and the Display Goals to Edit modules. This module requires that the user first have registered an account through the Register Account module, logged in through the Login module, have at least one 3rd party health app registered through the Register App module, and to have had their goals (or lack thereof) displayed on the Edit Goals page through the Display Goals to Edit module.

4.1.15 Change Goal intermodule dependencies

The Change Goal module depends upon the Register Account, Login, Register App, Set Goal, and the Display Goal to Edit modules. This module requires that the user first have registered an account through the Register Account module, logged in through the Login module, have at least one 3rd party health app registered through the Register App module, have at least one goal set through the Set Goal module, and to have had the goal(s) displayed on the Edit Goals page through the Display Goal to Edit module.

4.1.16 Remove Goal intermodule dependencies

The Remove Goal module depends upon the Register Account, Login, Register App, Set Goal, and the Display Goal to Edit modules. This module requires that the user first have registered an account through the Register Account module, logged in through the Login module, have at least one 3rd party health app registered through the Register App module, have at least one goal set through the Set Goal module, and to have had the goal(s) displayed on the Edit Goals page through the Display Goal to Edit module.

4.1.17 Display Goals to Main intermodule dependencies

The Display Goals to Main module depends upon the Register Account, Login, Register App, and the Set Goal modules. This module requires that the user first have registered an account through the Register Account module, logged in through the Login module, have at least one 3rd party health app registered through the Register App module, and have at least one goal set through the Set Goal module.

4.1.18 Display Progress Graph intermodule dependencies

The Display Progress Graph module depends upon the Register Account, Login, Register App, Set Goal, and the Display Goals to Main modules. This module requires that the user first have registered an account through the Register Account module, logged in through the Login module, have at least one 3rd party health app registered through the Register App module, have at least one goal set through the Set Goal module, and to have had the goal(s) displayed on the Main Menu page.

4.2 Data dependencies

4.2.1 User data entity dependencies

The User data entity is a completely independent data entity, though all the other data entities are dependent upon it.

4.2.2 3rd Party App data entity dependencies

The 3rd Part App data entity depends upon the User data entity. This data entity relies upon the **user_id** from the User data entity to travers, pull, and edit data in the HealthOut database.

4.2.3 App Log data entity dependencies

The App Log data entity depends upon the 3rd party app and User data entities. Being a subclass of the 3rd Party App data entity, this data entity can only exist as an extension of it, and by proxy also depends upon the **user_id** from the User data entity.

4.2.4 Goal data entity dependencies

The Goal data entity depends upon all other data entities. This data entity relies upon the **user_id** from the User data entity to travers and edit data in the HealthOut database, upon **registered** from the 3rd Party App data entity to tell if an app is registered, and upon all attributes from the App Log data entity to display goals and progress graphs to the user.

4.3 Traceability table

User story	Use Case(s)	Activity Diagram(s)	GUI(s)	User Manual section(s)
As a user, I want to register for HealthOut so that I have an account to access the app	Register Account	Register Account	UI 02 UI 01	Register an Account
As a user, I want to login to HealthOut so that I can have access to its features	Login	Login Recover Password	UI 01 UI 11 UI 03	Login to Account Recover Password
As a user, I want to logout of HealthOut so that my information is secure	Logout	Log out	UI 09 UI 13 UI 01	Logout of Account
As a user, I want to register a health app to HealthOut so I can use that app's data for my goal	Register App	Register App	UI 03 UI 07 UI 08	Register App to HealthOut
As a user, I want to unregister a health app from HealthOut, so its data is not used for my goal	Unregister App	Unregister App	UI 03 UI 07	Unregister app to HealthOut
As a user, I want to set a daily /weekly /monthly /yearly goal for the number of steps per day so that I can fulfill my goal	Set Goal	Edit Goal	UI 03 UI 05 UI 06	Add Health Goal
As a user, I want to set a daily /weekly /monthly /yearly goal for	Set Goal	Edit Goal	UI 03 UI 05 UI 06	Add Health Goal

the number of miles walked per day so that I can fulfill my goal				
As a user, I want to set a daily /weekly /monthly /yearly goal for the number of calories consumed so that I can fulfill my goal	Set Goal	Edit Goal	UI 03 UI 05 UI 06	Add Health Goal
As a user, I want to set a daily /weekly /monthly /yearly goal for the number of calories burned so that I can fulfill my goal	Set Goal	Edit Goal	UI 03 UI 05 UI 06	Add Health Goal
As a user, I want to set a daily /weekly /monthly /yearly goal for my blood pressure so that I can see if I meet my goal	Set Goal	Edit Goal	UI 03 UI 05 UI 06	Add Health Goal
As a user, I want to set a daily /weekly /monthly /yearly goal for my pulse so that I can see if I meet my goal	Set Goal	Edit Goal	UI 03 UI 05 UI 06	Add Health Goal
As a user, I want to be able to change my goals at any point, so it better fits my needs	Change Goal	Edit Goal	UI 03 UI 05 UI 06	Change Health Goal
As a user, I want to be able to	Remove Goal	Edit Goal	UI 03 UI 05	Remove Health Goal

remove a goal at any point if I no longer interested in that goal			UI 14	
As a user, I want to view my progress towards a goal, graphically, so that I know if it's fulfilled	View Goal View Progress Graph for Goal	View Goal View Graph	UI 03 UI 04	View Health Goals View Progress Graph for Goal

5. Interface description

5.1 Module interface

Describes how the modules in HealthOut will communicate with one another, outlining the inputs and outputs.

5.1.1 Register Account module interface

Input:

1. The user will input the below parameters pertaining to their account.
    - Desired email address (String)
    - Desired password (String)
    - Confirmation of desired password (String)
2. Later, the Activate Email module will input a boolean as to whether the user has activated their email address.

Output:

1. Pertaining to user's input
    - If the desired email address is invalid (does not follow 'prefix@domain' format) then the module will output an "Invalid email address" error to the Register App page.
    - If the desired password is invalid (not between 5 - 50 characters) then the module will output an "Invalid password" error to the Register App page.
    - If the two passwords do not match then the module will output an "Password mismatch" error to the Register App page.
    - Otherwise, the module will try to gain access to the HealthOut database.
2. Pertaining to the HealthOut database
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a "Try again later" error to the Register App page.
    - If while searching through the **User table**, the email address is already taken then the module will output an "Email address taken" error to the Register App page.
    - Otherwise, the module will output the email address to the Activate Email module.
3. When the Activate Email module returns true that the email has been activated, then module will create an account for the user in the **User table** in the HealthOut database, and output that the user's account has been registered. [5]

5.1.2 Activate Email module interface

Input:

1. Either the Register Account module or the Change Email module will input an email address (String).
2. Later, the activation link will input a true boolean that the email has been activated.

Output:

1. First the module will output an activation email to the email address that contains an activation link.
2. If the user clicks the activation link inside their activation email, then after the module receives the signal, it will output a true boolean that the email has been activated to either the Register Account module or the Change Email module. [5]

5.1.3 Delete Account module interface

Input:

1. The user's id (Integer) will be inputted to this module by the class that calls it.

Output:

1. Module will try to gain access to Healthout database
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a "Try again later" error to the screen.
    - Otherwise, the module will remove the user from the **User table** and output that the account has been successfully deleted. [5]

5.1.4 Recover Password module interface

Input:

1. The user will input their email address (String).

Output:

1. Pertaining to user's input
    - If the email address is invalid (does not follow 'prefix@domain' format) then the module will output an "Invalid email address" error to the Login page.
    - Otherwise, the module will try to gain access to HealthOut's database.
2. Pertaining to the HealthOut database
    - If the module cannot gain access to the HealthOut database through the Microsoft Azure API, then the module will output a "Try again later" error to the Login page.
    - Otherwise, the module will search the **User table** to see if the email address matches any accounts
3. Pertaining to the email search
    - If the email address does not match any account in the **User table** then the module will output a "Email not found" error to the Login page.
    - Otherwise, the module will pull the that account's password and output a recovery email to the email address that contains the password to that account. [5]

5.1.5 Login module interface

Input:

1. The user will input the below parameters pertaining to their account.
    - Email address (String)
    - Password (String)

Output:

1. Pertaining to user's input
    - If the email address is invalid (does not follow 'prefix@domain' format) then the module will output an "Invalid email address" error to the Login page.
    - If the password is invalid (not between 5 - 50 characters) then the module will output an "Invalid password" error to the Login page.
    - Otherwise, the module will try to gain access to the HealthOut database.
2. Pertaining to the HealthOut database
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a "Try again later" error to the Login page.
    - If while searching through the **User table**, the email address does not match any account, then the module will output an "Account not found" error.
    - If while searching through the **User table**, the email address is found, but the password does not match, then the module will output an "Incorrect password" error.
    - Otherwise, the module will log the user in. [5]

5.1.6 Logout module interface

Input:

1. None

Output:

1. The module will log the user out. [5]

5.1.7 Change Email module interface

Input:

1. The user's id (Integer) will be inputted to this module by the class that calls it.
2. The user will input the below parameters pertaining to their account.
    - Current password (String)
    - Newly desired email address (String)
3. Later, the Activate Email module will input a boolean as to whether the user has activated their email address.

Output:

1. Pertaining to user's input

- If the current password is invalid (not between 5 - 50 characters) then the module will output an “Invalid password” error to the Edit Account page.
- If the newly desired email address is invalid (does not follow ‘prefix@domain’ format) then the module will output an “Invalid email address” error to the Edit Account page.
- Otherwise, the module will try to gain access to the HealthOut database.

2. Pertaining to the HealthOut database
   - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a “Try again later” error to the Edit Account page.
   - If the password does not match the user’s password in the **User table,** then the module will output a “Incorrect password” error.
   - If while searching through the **User table**, the email address is already taken then the module will output an “Email address taken” error to the Edit Account page.
   - Otherwise, the module will output the email address to the Activate Email module.
3. When the Activate Email module returns true that the email has been activated, then module will update the user’s email address in the **User table** in the HealthOut database, and output that the user’s account has been registered. [5]

5.1.8 Change Password module interface

Input:

1. The user’s id (Integer) will be inputted to this module by the class that calls it.
2. The user will input the below parameters pertaining to their account.
   - Current password (String)
   - Newly desired password (String)
   - Confirmation of newly desired password (String)

Output:

1. Pertaining to user’s input
   - If the current password is invalid (not between 5 - 50 characters) then the module will output an “Invalid password” error to the Edit Account page.
   - If the newly desired password is invalid (not between 5 - 50 characters) then the module will output an “Invalid password” error to the Edit Account page.
   - If the two passwords do not match then the module will output an “Password mismatch” error to the Edit Account page.
   - Otherwise, the module will try to gain access to the HealthOut database.
2. Pertaining to the HealthOut database
   - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a “Try again later” error to the Edit Account page.
   - If the password does not match the user’s password in the **User table,** then the module will output a “Incorrect password” error.

- Otherwise, the module will update the user's password in the **User table.** [5]

5.1.9 Display 3rd Party Apps module interface

Input:

1. None

Output:

1. Pertaining to the 3rd Party App data entities that this module will loop through.
    - If the registered boolean is true, then the module will output that app to the Register App page as registered.
    - Otherwise, the module will output that app to the Register App page as unregistered. [5]

5.1.10 Register App module interface

Input:

1. The following parameters will be inputted to this module by the class that calls it.
    - User's id (Integer)
    - App id (Integer)
2. The user may be asked to input the following parameters pertaining to the app they are trying to register.
    - Username to app (String)
    - Email to app (String)
    - Password to app (String)

Output:

1. Module will try to gain access to Healthout database
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a "Try again later" error to the Register App page.
    - Otherwise, the module will see if the row in the **API table** that matches the user id and app id has a username, email, and password stored.
2. If there is a username, email, and password stored, they will be sent to that app's API.
    - If they come back as valid with the user's log data for that app, then the module will update the row in the **API table** as registered, add the log data to the corresponding **CompatibleAppLogs table**, and the module will output the app as registered.
    - Otherwise, the module will remove the username, email, and password from the **API table.**
3. If there is not a username, email, and password stored, the user will be prompted to enter them. Afterwards they will be sent to that app's API.
    - If they come back as valid with the user's log data for that app, then the module will update the row in the **API table** as registered, add the log data to the corresponding **CompatibleAppLogs table**, and the module will output the app as registered.

    - Otherwise, the module will output an “Invalid login information for app” error to the Register App page. [5]

5.1.11 Unregister App module interface

Input:

1. The following parameters will be inputted to this module by the class that calls it.
    - User’s id (Integer)
    - App id (Integer)

Output:

1. Module will try to gain access to Healthout database
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a “Try again later” error to the Register App page.
    - Otherwise, the module will update the row in the **API table** that matches the user id and app id as unregistered, and then output that the app has been successfully unregistered. [5]

5.1.12 Update Logs module interface

Input:

1. The user’s id (Integer) will be inputted to this module by the class that calls it.

Output:

1. Module will try to gain access to Healthout database
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a “Try again later” error to the screen.
    - Otherwise, the module will search through the **API table** chronologically for all rows that match the user’s id and hold a true boolean value for registered.
2. Pertaining to rows in the **API table** that match the user’s id and hold a true boolean value for registered. Then the username, email, and password also stored in that row will be sent to the app API that corresponds to the app id also stored in that row.
    - If they come back as valid with the user’s log data for that app, then the module will add the log data to the corresponding **CompatibleAppLogs table**.
    - Otherwise, the module will remove the username, email, and password from the **API table** and update the registered as a false boolean value. [5]

5.1.13 Display Goals to Edit module interface

Input:

1. None

Output:
- Module will loop through all Goal data entities and output the data for each goal to the Edit Goals page. [5]

5.1.14 Set Goal module interface

Input:
1. The user's id (Integer) will be inputted to this module by the class that calls it.
2. The user will input the below parameters pertaining to the goal's specifications.
    - Select app (String)
    - Select goal type (String)
    - Target for goal (Integer)
    - Select time period (String)

Output:
1. Pertaining to the target for goal entered by the user
    - If target for goal is not an integer, then the module will output a "Invalid target" error to the Specify Goal page.
    - Otherwise, the module will try to gain access to the HealthOut database.
2. Pertaining to the HealthOut database.
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a "Try again later" error to the Specify Goal page.
    - Otherwise, the module will search through the **Goal table** chronologically to see if the goal will be a duplicate.
3. Pertaining to whether goal is a duplicate
    - If the goal is a duplicate, then the module will output a "Duplicate goal" error to the Specify Goal page.
    - Otherwise, the goal will be added to the **Goal table** and the module will output that the goal was successfully set.[5]

5.1.15 Change Goal module interface

Input:
1. The user's id (Integer) will be inputted to this module by the class that calls it.
2. The user may change the below parameters pertaining to the goal's specifications.
    - Select app (String)
    - Select goal type (String)
    - Target for goal (Integer)
    - Select time period (String)

Output:
1. Pertaining to the target for goal entered by the user
    - If target for goal is not an integer, then the module will output a "Invalid target" error to the Specify Goal page.

        - Otherwise, the module will try to gain access to the HealthOut database.
2. Pertaining to the HealthOut database.
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a “Try again later” error to the Specify Goal page.
    - Otherwise, the module will search through the **Goal table** chronologically to see if the goal was changed or is now a duplicate.
3. Pertaining to whether goal was changed or is now a duplicate
    - If the goal was not changed, then the module will output a “Goal not changed” error on the Specify Goal page.
    - If the goal is now a duplicate, then the module will output a “Duplicate goal” error to the Specify Goal page.
    - Otherwise, the goal will be changed in the **Goal table** and the module will output that the goal was successfully changed. [5]

5.1.16 Remove Goal module interface

Input:

1. The following parameters will be inputted to this module by the class that calls it.
    - User’s id (Integer)
    - App id (Integer)
    - Goal id (Integer)
    - Period if (Integer)

Output:

1. Module will try to gain access to Healthout database
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a “Try again later” error to the Edit Goals page.
    - Otherwise, the module will remove the row in the **Goal table** that matches the user id, app id, goal id, and the period id, and then output that the goal has been successfully removed. [5]

5.1.17 Display Goals to Main module interface

Input:

1. None

Output:

1. Module will loop through all Goal data entities and output the data for each goal to the Main Menu. [5]

5.1.18 Display Progress Graph module interface

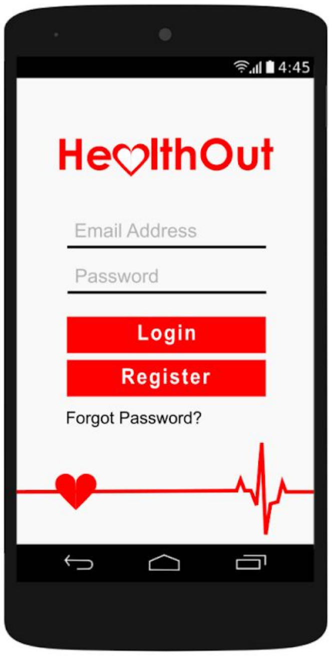
Input:

1. The following parameters will be inputted to this module by the class that calls it.
    - User's id (Integer)
    - App id (Integer)
    - Goal id (Integer)
    - Period if (Integer)
2. Goal data entity for the goal selected on the Main Menu page

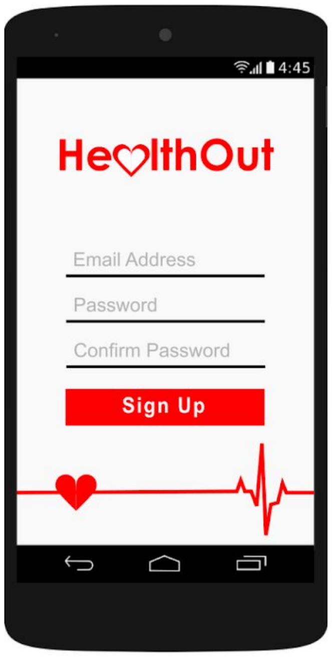
Output:

1. Module will try to gain access to Healthout database
    - If the module can not gain access to the HealthOut database through the Microsoft Azure API, then the module will output a "Try again later" error to the Progress Graph page.
    - Otherwise, the module will search through the Goal table chronologically for the rows that match the user's id, app id, goal id, and period id. Then the module will pull the needed log data from the corresponding app's CompatibleAppLogs table and Module will output a graph of the user's progression for that goal to the Progress Graph page. [5]

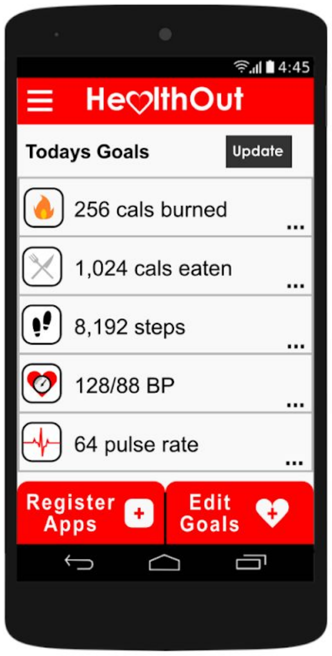
5.2 User interface



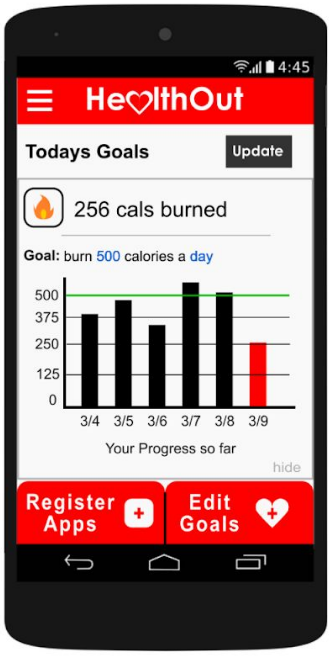
UI 01 - Login page



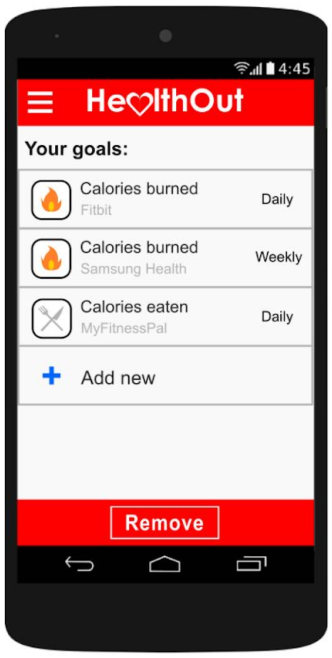
UI 02 - Register account page



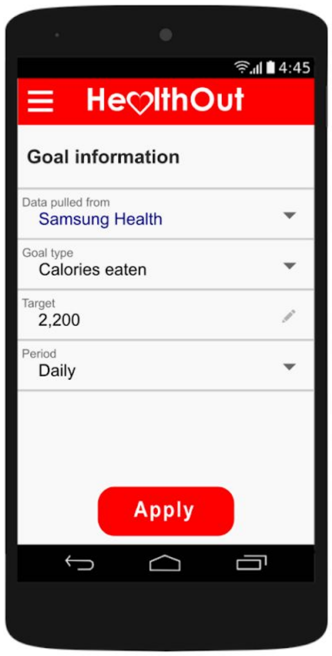
UI 03 - Main menu page



UI 04 - Progress graph page



UI 05 - Edit goals page



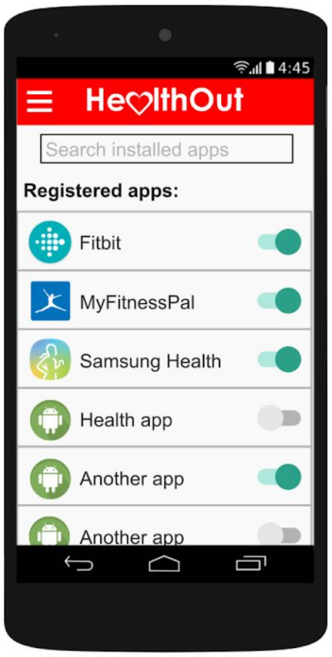
Data pulled from
Samsung Health

Goal type
Calories eaten

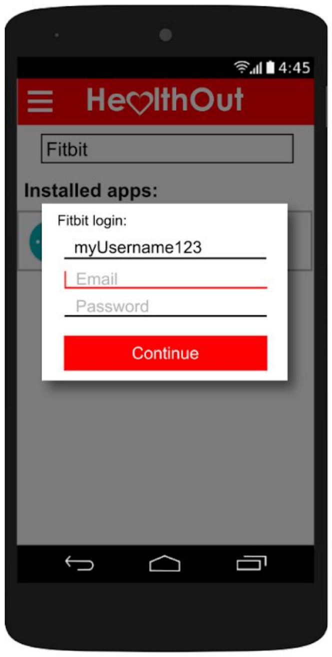
Target
2,200

Period
Daily

UI 06 - Specify goal page

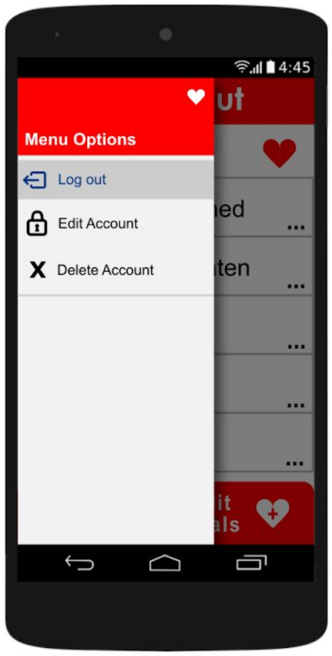


UI 07 - Register app page

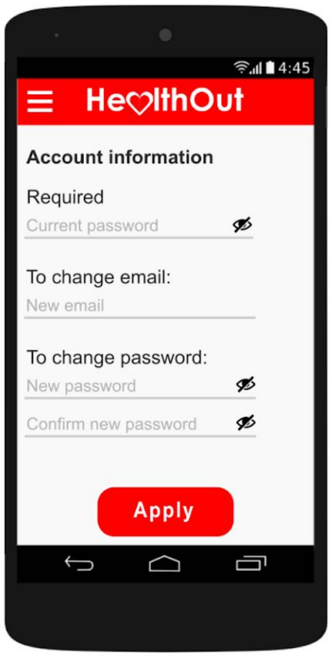


Fitbit login:
myUsername123

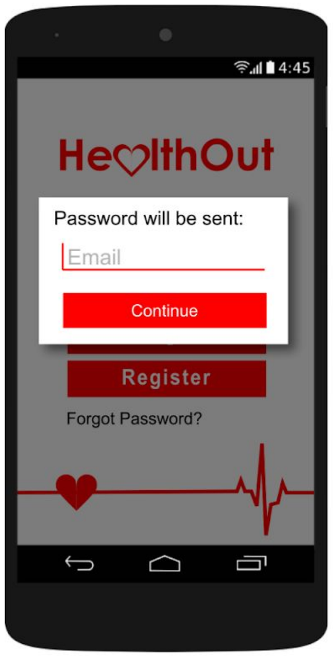
UI 08 - App login prompt



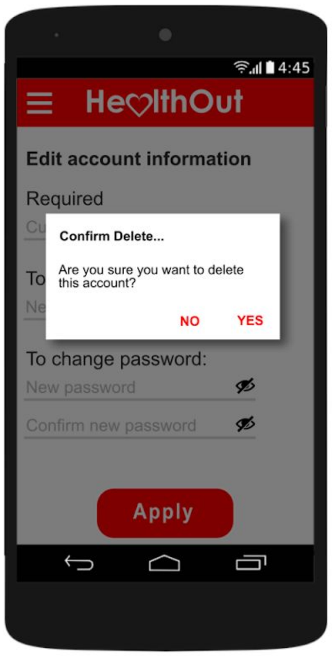
UI 09 - Account options sidebar



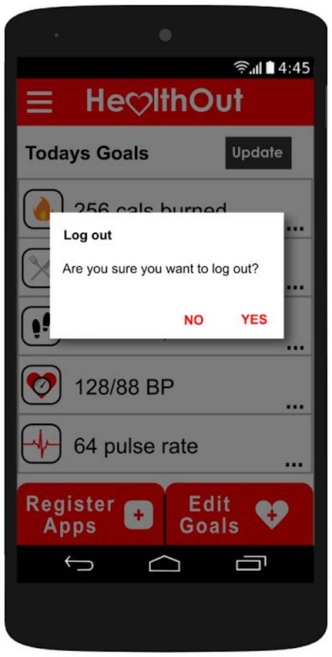
UI 10 - Edit account page



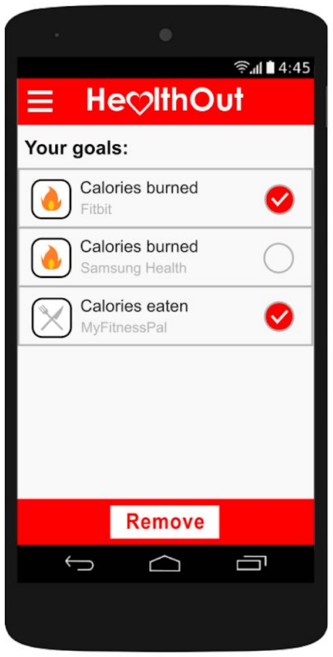
UI 11 - Email prompt



UI 12 - Confirm delete

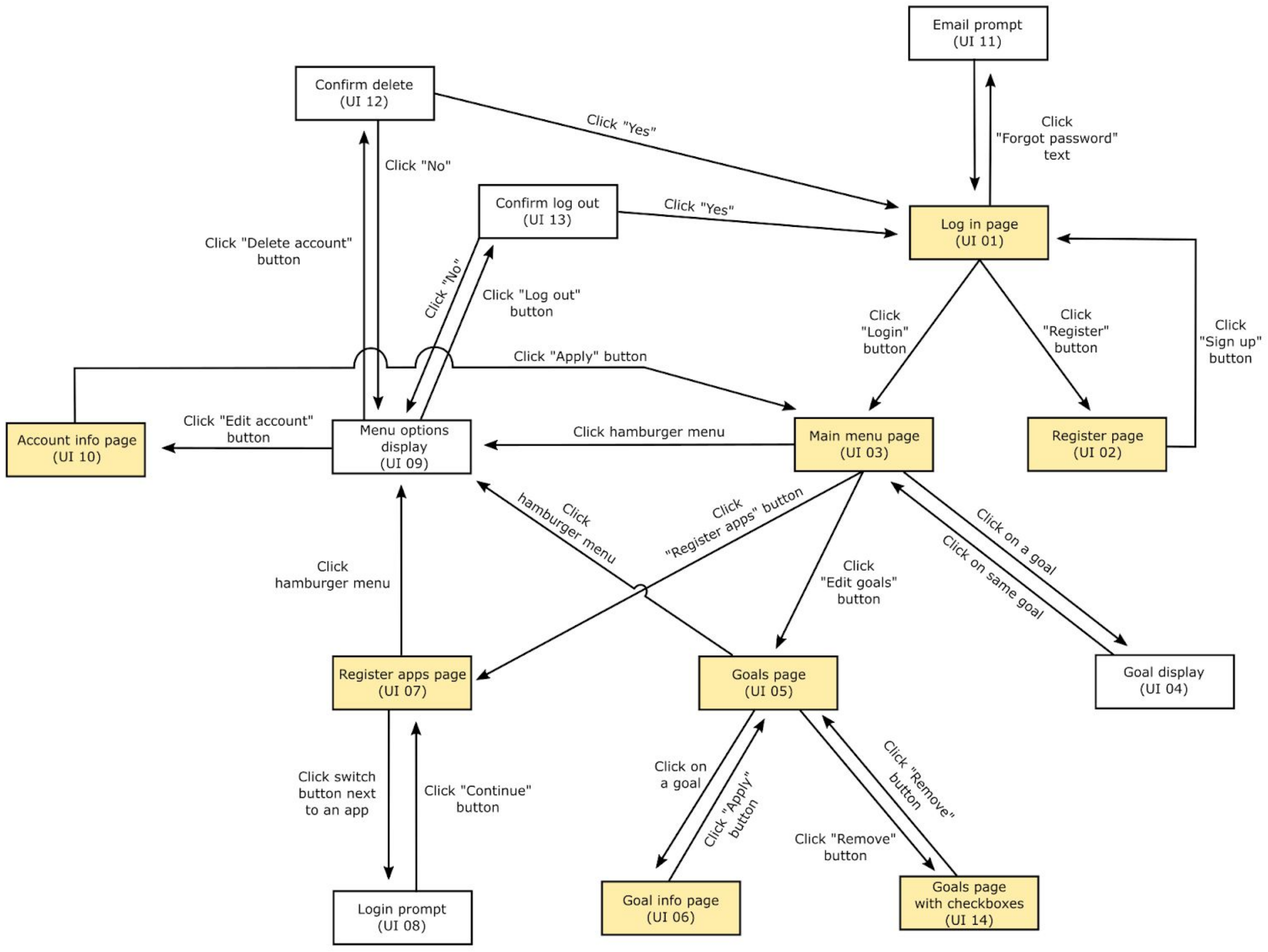


UI 13 - Confirm logout



UI 14 - Remove goals page

5.3 GUI transition diagram



5.4 User manual

Part 1: Getting Started

Download the App

1. Open the App Store and search for HealthOut.
2. Download and install the HealthOut mobile app to your desired android device.
3. Open the HealthOut mobile app. [4]

Register an Account

1. Go to the Login page.
2. Once at the login page, click the “Register” button. This will take you to the Register Account page.

3. Click the “Email Address” textbox and enter your desired email.
4. Click the “Password” textbox and enter your desired password.
5. Click the “Confirm Password” textbox and confirm your password.
6. Click the “Sign Up” button.
7. If the email address was valid, and both passwords were valid and match, you will receive an activation email. If invalid, retype your desired email and password again before re-clicking the “Sign Up” button
8. Go to your email address, open the activation email, and click the activation link. If you did not receive an activation email, try repeating the register process over again. [4]

<u>Login to Account</u>

1. Go to the Login page
2. Enter in your email and password registered to your account and click the “Login” button.
3. If your the email address and password you entered was correct, then you will be logged in and taken to the main menu. If incorrect, retype your email and password before clicking the “Login” button again. [4]

<u>Recover Password</u>

1. Go to the Login page.
2. Click the text that says “Forgot Password?”
3. A prompt will appear asking for your email address. Enter it and click the “Continue” button.
4. Go to your email address and open the recover forgotten password email. This will contain the password to your account. [4]

Part 2: Manage Account

<u>Logout of Account</u>

1. On any page while logged in, click the hamburger menu button (three horizontal lines) in the top right corner. This will open a sidebar menu with account options.
2. Click the “Logout” button. You will be prompted if you are sure.
3. Click the text that says “Yes”. This will log you out and take you back to the login page. [4]

<u>Change Email</u>

1. On any page while logged in, click the hamburger menu button (three horizontal lines) in the top right corner. This will open a sidebar menu with account options.
2. Click the “Edit Account” button. This will take you to the Edit Account page.
3. Click the “Current password” textbox and enter your current password.
4. Click the “New email” textbox and enter your new desired email.
5. Click the “Apply” button.
6. If the Current password you entered is correct, and the new email you

entered is valid, you will now receive an activation email to the new email address you entered. Otherwise, try entering your current password and new desired email again, before re-clicking the “Apply” button.
7. Go to your email address, open the activation email, and click the activation link. If you did not receive an activation email, try repeating the change email process over again. [4]

<u>Change Password</u>

1. On any page while logged in, click the hamburger menu button (three horizontal lines) in the top right corner. This will open a sidebar menu with account options.
2. Click the “Edit Account” button. This will take you to the Edit Account page.
3. Click the “Current password” textbox and enter your current password.
4. Click the “New password” textbox and enter your new desired password.
5. Click the “Confirm new password” textbox and confirm your new desired password.
6. Click the “Apply” button.
7. If the current password is correct, and both new desired passwords are valid and match, your password will be updated. If invalid, retype your current password and both new desired passwords again, before re-clicking the “Apply” button. [4]

<u>Delete Account</u>

1. On any page while logged in, click the hamburger menu button (three horizontal lines) in the top right corner. This will open a sidebar menu with account options.
2. Click the “Delete Account” button. You will be prompted if you are sure.
3. Click the text that says “Yes”. This will delete your account and take you back to the login page. [4]

Part 3: Manage 3rd Party Health Apps

<u>Register App to HealthOut</u>

1. Go to the Register Account page.
2. Click the switch button next to the app you wish to register. This will cause a pop up prompt asking you to enter your email and password for that app.
3. Click the “Email” textbox and enter your email for the app.
4. Click the “Password” textbox and enter your password for the app.
5. If a “Username” textbox appears, click it and enter your username for the app.
6. Click the “Continue” button (there may be some loading afterwards while we connect to that app’s API).
7. If the email, password, and username (if applies) you entered is correct, the app you selected will be registered. Try repeating the register app

process [4]

Unregister App to HealthOut

1. Go to the Register Account page.
2. Click the switch button next to the app you wish to unregister. This will unregister the app. [4]

Part 4: Manage Goals

Add Health Goal

1. Go to the Edit Goals page.
2. Click the “Add new” button. This will take you to the Specify Goal page.
3. Click the “Data pulled from” drop down box and select the health app you wish to pull data from for the goal (If the app you desire does not appear, make sure you have it registered).
4. Click the “Goal type” drop down box and select the type of goal you want.
5. Click the “Target” box and enter your desired target for the goal.
6. Click the “Period” drop down box and select the time period you want for the goal.
7. Click the “Apply” button. Now your goal will be viewable at the main menu. [4]

Change Health Goal

1. Go to the Edit Goals page.
2. Click on the goal you wish to edit. This will take you to the Specify Goal page for that particular goal.
3. Click on whichever aspects of the goal you wish to edit and make the changes you desire.
4. Click the “Apply” button. Now your goal will be updated. [4]

Remove Health Goal

1. Go to the Edit Goals page.
2. Click the remove button at the bottom of the screen. This will present a checkbox next to each health goal.
3. Select which goals you want to delete by checking off the desired boxes.
4. Click the remove button again. This will remove all the goals you have checked off. [4]

Part 5: View Goals and Progress Graphs

View Health Goals

1. Go to the Main Menu page.
2. All goals are displayed at the main (you may need to scroll down to view a goal if you have multiple goals). [4]

View Progress Graph for Goal

1. Go to the Main Menu page.
2. Click on the goal you want to view a progress graph for. This will take you to the progress graph for that particular goal. [4]

6. Detailed design

6.1 Module detailed design

Detailed descriptions for the design of each module for HealthOut; outlining what algorithms, methods, parameters, entities, and conditions are involved.

6.1.1 Register App module detail

Class: User
Attributes: **logged_in**, **user_id**, **email**, **password**.

Method: registerAccount(String email, String password, String confirm)

Precondition: user has inputs an email address (String), password (String), and confirmation password (String) into the Register App page and clicks "Sign up" button.

Algorithm:

1. Checks if inputted email is valid.
    - If false -- prompt "invalid email"
    - If true -- continues
2. Checks if inputted password is valid
    - If false -- prompt "invalid password"
    - If true -- continues
3. Make sure both inputted passwords match.
    - If false -- prompt "password mismatch"
    - If true -- continues
4. Calls verifyRegister(email) method in Database
    - Returns false -- prompt "email already registered"
    - Returns true -- continues
5. Calls activateEmail(email) method in Activate class from the Activate Email module and moves user to Login page.
    - Returns true -- continues (only possibility)
6. Calls registerAccountToTable(email, password) method in Database.
    - Returns false -- prompt "try activation link again later"
    - Returns true -- display "Account activated" to Login page

Postcondition: user will have an account registered to HealthOut.

Class: Database
Attributes: None

Method: verifyRegister(String email)

Precondition: called by registerAccount(email, password, confirm)
Algorithm:
1. Tries to gain access to HealthOut database
    - If false -- prompt "try again later"
    - If true -- continue
2. Checks database to see make sure email is not already assigned to another account
    - If false -- return false
    - If true -- return true

Postcondition: will know if email is available or not

Method: registerAccountToTable(String email, String password)
Precondition: called by registerAccount(email, password, confirm)
Algorithm:
1. Tries to gain access to HealthOut database
    - If false -- prompt "try again later"
    - If true -- continue
2. Tries to creates account and adds to the database
    - If false -- return false
    - If true -- return true

Postcondition: will know if account was successfully added to the database [6]

6.1.2 Active Email module detail

Class: Activate
Attributes: none
Method: activateEmail(String email)
Precondition: either called by Register App or Change Email module
Algorithm:
1. Sends activate email with activation link to email address
2. Receives signal when activation link is clicked
3. Returns true (only possibility)

Postcondition: email will be activated [6]

6.1.3 Delete Account module detail

Class: User
Attributes: **logged_in**, **user_id**, **email**, **password**.
Method: deleteAccount()
Precondition: user has created an account and logged in
Algorithm:
1. Calls delete(user_id) method in Database
    - Returns false -- prompt "try again later"

- Returns true -- moves user to Login page

<u>Postcondition:</u> user's account is deleted[6]

<u>Class:</u> Database
<u>Attributes:</u> none
<u>Method:</u> delete(Integer user_id)
<u>Precondition:</u> called by deleteAccount()
<u>Algorithm:</u>

1. Tries to gain access to HealthOut database
    - If false -- prompt "try again later"
    - If true -- continue
2. Tries to remove account from database
    - If false -- return false
    - If true -- returns true

<u>Postcondition:</u> will known if user's account is deleted [6]

6.1.4 Recover Password module detail

<u>Class:</u> User
<u>Attributes:</u> **logged_in**, **user_id**, **email**, **password**.
<u>Method:</u> recoverPassword(String email)
<u>Precondition:</u> user has clicked "forgot password?" on login page and inputs an email address.
<u>Algorithm:</u>

1. Calls retrievePassword(email)
2. Sends email containing password to the inputted email address

<u>Postcondition:</u> user will receive email containing their password.

<u>Class:</u> Database
<u>Attributes:</u> None
<u>Method:</u> retrievePassword(String email)
<u>Precondition:</u> called by recoverPassword(email)
<u>Algorithm:</u>

1. Tries to gain access to HealthOut database
    - If false -- prompt "try again later"
    - If true -- continue
2. Pulls and returns password for the inputted email address

<u>Postcondition:</u> will return password

6.1.5 Login module detail

<u>Class:</u> User
<u>Attributes:</u> **logged_in**, **user_id**, **email**, **password**.
<u>Method:</u> login(String email, String password)

<u>Precondition:</u> user has entered an email address and password into the Login page and clicks “Login” button.
<u>Algorithm:</u>

1. Checks if email is valid.
    - If false -- prompt “invalid email”
    - If true -- continues
2. Checks if password is valid
    - If false -- prompt “invalid password”
    - If true -- continues
3. Calls verifyLogin(email, password) method in Database
    - If false -- prompt “incorrect email or password”
    - If true -- sets **logged_in** to true, **user_id** to the user’s id, **email** to the user’s email, and **password** to the user’s password. Then user is taken to the Main Menu page.

<u>Postcondition:</u> user is logged into HealthOut.

<u>Class:</u> Database
<u>Attributes:</u> None
<u>Method:</u> verifyLogin(String email, String password)
<u>Precondition:</u> called by registerAccount(email, password, confirm)
<u>Algorithm:</u>

1. Tries to gain access to HealthOut database
    - If false -- prompt “try again later”
    - If true -- continue
2. Checks if the inputted email matches any accounts in the database
    - If false -- return false
    - If true -- continues
3. Checks if the password matches for account
    - If false -- return false
    - If true -- return true

<u>Postcondition:</u> will known if user will be logged in

6.1.6 Logout module detail

<u>Class:</u> User
<u>Attributes:</u> **logged_in**, **user_id**, **email**, **password**.
<u>Method:</u> logout()
<u>Precondition:</u> user has registered an account, logged in, clicks hamburger menu button, clicks “Logout” button.
<u>Algorithm:</u>

1. Sets **logged_in** to false and moves user to Login page.

<u>Postcondition:</u> user is logged out of HealthOut.

6.1.7 Change Email module detail

Class: User

Attributes: **logged_in**, **user_id**, **email**, **password**.

Method: changeEmail(String password, String email)

Precondition: user has inputs a current password (String) and email address (String) into the Edit Account page and clicks “Apply” button.

Algorithm:

1. Checks if inputted email is valid.
    - If false -- prompt “invalid email”
    - If true -- continues
2. Checks if current password is correct.
    - If false -- prompt “incorrect password”
    - If true -- continues
3. Calls verifyChangeEmail(user_id, email) method in Database
    - Returns false -- prompt “email already registered to an account”
    - Returns true -- continues
4. Calls activateEmail(email) method in Activate class from the Activate Email module and moves user to Main Menu page.
    - Returns true -- continues (only possibility)
5. Calls changeEmailInTable(user_id, email) method in Database.
    - Returns false -- prompt “try activation link again later”
    - Returns true -- set **email** to new email and display “Email changed” to Main Menu page

Postcondition: user will have their email changed.

Class: Database

Attributes: None

Method: verifyChangeEmail(Integer user_id, String email)

Precondition: called by changeEmail(password, email)

Algorithm:

1. Tries to gain access to HealthOut database
    - If false -- prompt “try again later”
    - If true -- continue
2. Checks database to see make sure email is not already assigned to another account
    - If false -- return false
    - If true -- return true

Postcondition: will know if email is available or not

Method: changeEmailInTable(Integer user_id, String email)

Precondition: called by registerAccount(email, password, confirm)

Algorithm:

1. Tries to gain access to HealthOut database
    - If false -- prompt “try again later”

                - If true -- continue
            2. Tries to change email to account in database
                - If false -- return false
                - If true -- return true

Postcondition: will know if email was successfully changed in the database [6]

6.1.8 Change Password module detail

Class: User

Attributes: **logged_in**, **user_id**, **email**, **password**.

Method: changePassword(String currentPasword, String newPassword, String confirmPassword)

Precondition: user has inputs a current password (String), a new password (String), and a confirmation password (String) into the Edit Account page and clicks “Apply” button.

Algorithm:

1. Checks if inputted new password is valid.
    - If false -- prompt “invalid new password”
    - If true -- continues
    -
2. Make sure both inputted passwords match.
    - If false -- prompt “password mismatch”
    - If true -- continues
3. Checks if password is correct.
    - If false -- prompt “incorrect password”
    - If true -- continues
4. Calls changePasswordInTable(user_id, newPassword) method in Database
    - Returns false -- prompt “try again later”
    - Returns true -- set **password** to new password, move to Main Menu page, and display “Password changed”.

Postcondition: user will have their password changed.

Class: Database

Attributes: None

Method: changePasswordInTable(Integer user_id, String email)

Precondition: called by changePassword(currentPasword, newPassword, confirmPassword)

Algorithm:

1. Tries to gain access to HealthOut database
    - If false -- prompt “try again later”
    - If true -- continue
2. Tries to change password to account in database

- If false -- return false
- If true -- return true

Postcondition: will know if password was successfully changed in the database [6]

6.1.9 Display 3rd Party App module detail

Class: User
Attributes: **logged_in**, **user_id**, **email**, **password**.
Method: displayAllApps()
Precondition: user opens Registered Apps page
Algorithm:

1. Loops through all 3rd Party App data entities and calls the displayToRegistered() method in each 3rd Party App class

Postcondition: user will have all registered and unregistered apps displayed on the Register App page.

Class: 3rd Party App
Attributes: **app_id**, **app_name**, **registered**, **app_username**, **app_email**, **app_password**.
Method: displayAppToRegistered()
Precondition: called by displayAllApps
Algorithm:

1. Displays app to Register App page, as well as a switch button that:
    - If registered is false -- is off.
    - If registered is true -- is on.

Postcondition: app will be correctly displayed to Register App page.

6.1.10 Register App module detail

Class: 3rd Party App
Attributes: **app_id**, **app_name**, **registered**, **app_username**, **app_email**, **app_password**.
Method: registerApp(Integer app_id)
Precondition: user has had their 3rd Party Apps displayed and clicks switch next to an unregistered app.
Algorithm:

1. Calls checkAppInDatabase(user_id, app_id)
    - If returns false -- asks user to input username, email, and password for the app and then calls resgisterAppToDatabase(username, email, password)
    - If returns true -- continues
2. Calls updateLog(username, email, password)
    - If returns false -- prompt "incorrect login information"
    - If returns true -- continues
3. Calls updateLogDataToDatabase (user_id, app_id, steps_walked, miles_walked, calories_burned, calries_consumed, pulse, blood_pressure, epoch_timestamp)

        - If returns false -- prompt “try again later”
        - If returns true -- turn switch button on

    Postcondition: user will have the app registered to their account and their log data for that app added to the database.

  Method: updateLog(String username, String email, String password)

    Precondition: called by rsgiterApp(app_id).

    Algorithm:

    1. Tries to gain access to app’s API
        - If false -- prompt “try again later”
        - If true -- continue
    2. Sends login to user to see if it is correct
        - If false -- will return that it is invalid
        - If true -- will return the user’s log data for that app
    3. Sends log data to the App Log data entity

    Postcondition: user’s log data will be stored in the App Log data entity

Class: Database

Attributes: none

  Method: checkAppInDatabase(Integer user_id, Integer app_id)

    Precondition: called by registerApp(app_id).

    Algorithm:

    1. Tries to gain access to HealthOut database
        - If false -- prompt “try again later”
        - If true -- continue
    2. Checks if the app already has a username, email, or password stored for the app.
        - If false -- returns false
        - If true -- returns true

    Postcondition: will know if user has a username, email, or password stored for the app.

  Method: resgisterAppToDatabase(String username, String email, String password)

    Precondition: called by registerApp(app_id).

    Algorithm:

    1. Tries to gain access to HealthOut database
        - If false -- prompt “try again later”
        - If true -- continue
    2. Tries to add the user’s username, email, and password for the app to the database.
        - If false -- returns false
        - If true -- returns true

    Postcondition: will know if user has their username, email, or password added to the database.

Method: updateLogDataToDatabase (Integer user_id, Integer app_id, Integer steps_walked, Double miles_walked, Integer calories_burned, Integer calries_consumed, Integer pulse, String blood_pressure, Integer epoch_timestamp)

Precondition: called by App Log data entity

Algorithm:

1. Tries to gain access to HealthOut database
    - If false -- prompt "try again later"
    - If true -- continue
2. Tries to add the user's log data to the database.
    - If false -- returns false
    - If true -- returns true

Postcondition: will know if user has their log data added to the database.

6.1.11 Unregister App module detail

Class: 3rd Party App

Attributes: **app_id**, **app_name**, **registered**, **app_username**, **app_email**, **app_password**.

Method: unregisterApp(Integer app_id)

Precondition: user has had their 3rd Party Apps displayed and clicks switch next to a registered app.

Algorithm:

1. Calls unregisterAppInTable(user_id, app_id)
    - If return false -- prompt "try again later"
    - If returns true -- turns switch button for app off

Postcondition: user will have the app unregistered.

Class: Database

Attributes: none

Method: unregisterAppInTable(Integer user_id, Integer app_id)

Precondition: called by unregisterApp(app_id).

Algorithm:

1. Tries to gain access to HealthOut database
    - If false -- prompt "try again later"
    - If true -- continue
2. Tries to set app as unregistered in the database.
    - If false -- returns false
    - If true -- returns true

Postcondition: will know if app is set to unregistered in the database.

6.1.12 Update Logs module detail

Class: 3rd Party App

Attributes :**app_id**, **app_name**, **registered**, **app_username**, **app_email**, **app_password**.

Method: updateAllLogData()

Precondition: user has at least one app registered and clicks "Update" button.

Algorithm:

1. Calls retrieveRegisteredTable(user_id, registered)
2. Loops for all registered apps, calling updateLog(username, email, password)
    - If returns false -- unregisters app
    - If returns true -- stories log data in App Log data entity
3. Calls updateLogDataToDatabase (user_id, app_id, steps_walked, miles_walked, calories_burned, calries_consumed, pulse, blood_pressure, epoch_timestamp)
    - If returns false -- prompt "try again later"
    - If returns true -- continues
4. updates goal on Main Menu page

Postcondition: user will have log data for all registered apps updated.

Method: updateLog(String username, String email, String password)

Precondition: called by updateAllLogData()

Algorithm:

1. Tries to gain access to app's API
    - If false -- prompt "try again later"
    - If true -- continue
2. Sends login to user to see if it is correct
    - If false -- will return that it is invalid
    - If true -- will return the user's log data for that app
3. Sends log data to the App Log data entity

Postcondition: user's log data will be stored in the App Log data entity

Class: Database

Attributes: none

Method: retrieveRegisteredTable(Integer user_id, Boolean registered)

Precondition: called by updateAllLogData()

Algorithm:

1. Tries to gain access to HealthOut database
    - If false -- prompt "try again later"
    - If true -- continue
2. Pulls and returns a table of all apps the user has registered

Postcondition: returns table of all apps the user has registered.

Method: updateLogDataToDatabase (Integer user_id, Integer app_id, Integer steps_walked, Double miles_walked, Integer calories_burned, Integer calries_consumed, Integer pulse, String blood_pressure, Integer epoch_timestamp)

Precondition: called by App Log data entity

Algorithm:

1. Tries to gain access to HealthOut database
   - If false -- prompt “try again later”
   - If true -- continue
2. Tries to add the user’s log data to the database.
   - If false -- returns false
   - If true -- returns true

Postcondition: will know if user has their log data added to the database.

6.1.13 Display Goals to Edit module detail

Class: User
Attributes: **logged_in**, **user_id**, **email**, **password**.

Method: displayAllGoalsToEdit()

Precondition: user opens Edit Goals page
Algorithm:

1. Checks if user has any apps registered
   - If false -- display “Please register an app”
   - If true -- continues
2. Loops through Goal data entities and calls the displayGoalToEdit() method in each Goal class

Postcondition: user will have all goals displayed on the Edit Goals page.

Class: Goal
Attributes: **app_id**, **appPulledFrom**, **goal_id**, **goal_type**, **target_value**, **period_id**, **period_length**, **progress.**

Method: displayGoalToEdit()

Precondition: called by displayAllGoalsToEdit
Algorithm:

1. Displays goal to the Edit Goals page

Postcondition: goal will be displayed to Edit Goals page.

6.1.14 Set Goal module detail

Class: Goal
Attributes: **app_id**, **appPulledFrom**, **goal_id**, **goal_type**, **target_value**, **period_id**, **period_length**, **progress.**

Method: setGoal(Integer app_id, Integer goal_id, Integer period_id, Integer target_value)

Precondition: user is on Specify Goal page, selects an app, goal type, period, enters target value, and clicks “Apply” button
Algorithm:

1. Calls addGoalToTable(user_id, app_id, goal_id, period_id, target_value)
   - If returns false -- prompts “try again later”

- If returns true -- display new goal to Edit Goals page

Postcondition: goal will be added database and displayed to Edit Goals page.

Class: Database

Attributes: none

Method: addGoalToTable(Integer user_id, Integer app_id, goal_id, Integer period_id, String target_value)

Precondition: called by setGoal(app_id, goal_id, period_id, target_value)

Algorithm:

1. Tries to gain access to HealthOut database
   - If false -- prompt “try again later”
   - If true -- continue
2. Tries to add goal to database
   - If false -- returns false
   - If true -- returns true

Postcondition: will know if goal was added to the database.

6.1.15 Change Goal module detail

Class: Goal

Attributes: **app_id**, **appPulledFrom**, **goal_id**, **goal_type**, **target_value**, **period_id**, **period_length**, **progress.**

Method: changeGoal(Integer app_id, Integer goal_id, Integer period_id, Integer target_value)

Precondition: user is on Specify Goal page, selects an app, goal type, period, enters target value, and clicks “Apply” button

Algorithm:

1. Calls changeGoalInTable(user_id, app_id, goal_id, period_id, target_value)
   - If returns false -- prompts “try again later”
   - If returns true -- displays updated goal to Edit Goals page

Postcondition: goal will be changed in database and updated to Edit Goals page.

Class: Database

Attributes: none

Method: changeGoalInTable(Integer user_id, Integer app_id, goal_id, Integer period_id, String target_value)

Precondition: called by changeGoal(app_id, goal_id, period_id, target_value)

Algorithm:

1. Tries to gain access to HealthOut database
   - If false -- prompt “try again later”
   - If true -- continue
2. Tries to change goal in database

            - If false -- returns false
            - If true -- returns true

        Postcondition: will know if goal was changed in the database.

6.1.16 Remove Goal module detail

Class: User
Attributes: **logged_in**, **user_id**, **email**, **password**.

Method: removeAllSelectedGoals()
Precondition: user opens Edit Goals page, clicks “Remove” button, clicks checkbox for one or more goals, then clicks “Remove” button again

```
Algorithm:
    1. Loops through all selected goals and calls removeGoal(app_id,
       goal_id, period_id, target_value)
    2. Displays updated Edit Goals page
Postcondition: all selected goals will be removed.
```

Class: Goal
Attributes: **app_id**, **appPulledFrom**, **goal_id**, **goal_type**, **target_value**, **period_id**, **period_length**, **progress.**

Method: removeGoal(Integer app_id, Integer goal_id, Integer period_id, Integer target_value)

```
Precondition: called by removeAllSelectedGoals()
Algorithm:
    1. Calls removeGaolInTable(user_id, app_id, goal_id, period_id,
       target_value)
           ● If returns false -- prompts “try again later”
           ● If returns true -- continues
Postcondition: goal will be displayed to Edit Goals page.
```

Class: Database
Attributes: none

Method: removeGaolInTable(Integer user_id, Integer app_id, Integer goal_id, Integer period_id, Integer target_value)

```
Precondition: called by removeGoal(app_id, goal_id, period_id,
target_value)
Algorithm:
    1. Tries to gain access to HealthOut database
           ● If false -- prompt “try again later”
           ● If true -- continue
    2. Tries to remove goal from database
           ● If false -- returns false
           ● If true -- returns true
Postcondition: will know if goal was removed from the database.
```

6.1.17 Display Goal to Main module detail

Class: User

Attributes: **logged_in**, **user_id**, **email**, **password**.

Method: displayAllGoalsToMain()

Precondition: user opens Main Menu page

Algorithm:

1. Checks if user has any apps registered
   - If false -- display "Please register an app"
   - If true -- continues
2. Checks if user has any goals set
   - If false -- display "Please set a goal"
   - If true -- continues
3. Loops through Goal data entities and calls the displayGoalToMain() method in each Goal class

Postcondition: user will have all goals displayed on the Main Menu page.

Class: Goal

Attributes: **app_id**, **appPulledFrom**, **goal_id**, **goal_type**, **target_value**, **period_id**, **period_length**, **progress.**

Method: displayGoalToMain()

Precondition: called by displayAllGoalsToMain

Algorithm:

1. Displays goal to the Main Menu page

Postcondition: goal will be displayed to Main Menu page.

6.1.18 Display Progress Graph module detail

Class: Goal

Attributes: **app_id**, **appPulledFrom**, **goal_id**, **goal_type**, **target_value**, **period_id**, **period_length**, **progress.**

Method: displayGraphToMain()

Precondition: user opens Main Menu page and clicks on a goal

Algorithm:

1. Loops for all instances of log data for a corresponding app to the goal, calling retrieveAppLog(user_id, app_id, goal_type)
2. When Displays progress graph for goal to Main Menu page

Postcondition: user will have a progress graph for the goal displayed.

Class: Database

Attributes: none

Method: retrieveAppLog(Integer user_id, Integer app_id, Integer goal_type)

Precondition: called by displayGraphToMain()

Algorithm:

1. Tries to gain access to HealthOut database

        - If false -- prompt “try again later”
        - If true -- continue
    2. Pulls an instances of log data for a corresponding app to the goal and returns the data

Postcondition: will pull an instances of log data for a corresponding app to the goal

6.2 Data detailed design

Detailed descriptions for the design of each data entity for HealthOut; outlining what methods and attributes are used.

6.2.1 User data entity detail

The User data entity holds the data related to a particular user’s account. This data entity has four attributes, being the **logged_in, user_id, email**, and **password** attributes. The **logged_in** attribute is a Boolean from the User table that holds whether or not a user is logged into the app. The **user_id** attribute is an Integer from the User table that holds a unique number to the user that is used to traverse through the HealthOut database. The **email** attribute is a String from the User table that holds the particular user’s email address. The **password** attribute is a String from the User table that holds the particular user’s password.

6.2.2 3rd Party App data entity detail

The 3rd Party App data entity holds the data related to a particular compatible 3rd party health app. This data entity has seven attributes, being the **app_id, app_name, registered**, **app_username**, **app_email**, and **app_password** attributes. The **app_id** attribute from the API table is an Integer that holds a unique number to a particular app that is used to traverse through the HealthOut database. The **app_name** attribute is a String from the App table that holds the particular 3rd party health app’s name. The **registered** attribute is a Boolean from the API table that holds whether or not the particular 3rd party health app is registered. The **app_username** attribute is a String from the API table that holds the user’s username to the particular 3rd party health app. The **app_email** attribute is a String from the API table that holds the user’s email address to the particular 3rd party health app. The **app_password** attribute is a String from the API table that that holds the user’s password to the particular 3rd party health app.

6.2.3 App Log data entity detail

The App Log data entity is a subclass of the 3rd Party App data entity. This data entity holds the log data related to the user’s account for a particular registered 3rd party health app. This data entity has eight attributes, being the **steps_walked**, **miles_walked**, **calories_burned**, **calories_consumed**, **pulse**, **blood_pressure**, and **epoch_timestamp** attributes. The **steps_walked** attribute is an Integer from a particular CompatibleAppLogs table that holds the log data pertaining to steps walked from the particular 3rd party health app. The **miles_walked** attribute is a Double from a particular CompatibleAppLogs table that holds the log data pertaining to miles walked from the particular 3rd party health app. The **calories_burned** attribute is an Integer from a particular CompatibleAppLogs table that holds the log data pertaining to calories burned from the particular 3rd party health app. The **calories_consumed** attribute is an Integer from a particular CompatibleAppLogs table that holds the log data pertaining to calories consumed from the particular 3rd party health app. The **pulse** from a particular CompatibleAppLogs table attribute is an Integer that holds the log data pertaining to pulse from the particular 3rd party health app. The **blood_pressure** attribute is a String from a particular CompatibleAppLogs table that holds the log data pertaining to blood pressure from the particular 3rd party health app. The **epoch_timestamp** attribute is an Integer from a particular CompatibleAppLogs table that holds the epoch time for when the log data for the particular app was last updated.

6.2.4 Goal data entity detail

The Goal data entity holds the data related to a particular goal. This data entity has nine attributes, being the **app_id, appPulledFrom**, **goal_id, goal_type**, **target_value, period_id, period_length**, and **progress** attributes. The **app_id** attribute is an Integer from the Goal table that holds a unique number to a particular app that is used to traverse through the HealthOut database. The **appPulledFrom** attribute is a String from the App table that holds the selected 3rd party health app for the particular goal. The **goal_id** attribute is an Integer from the Goal table that holds a unique number to a particular goal type that is used to traverse through the HealthOut database. The **goal_type** attribute is a String from the Type table that holds the selected goal type for the particular goal. The **target_value** attribute is a String from the Goal table that holds the target for the particular goal. The **period_id** attribute is an Integer from the Goal table that holds a unique number to a particular time period that is used to traverse through the HealthOut database. The **period_length** attribute is a String from the Period table that holds the time period for the particular goal. The **progress** attribute is a Integer from a particular CompatibleAppLogs table that holds the log data for to the selected goal type for the selected 3rd party health app.

7. Meeting minute notes & responsibilities

Meeting Notes

Meeting 1 (03/15/19):

- GUI transition and use case diagrams were made
- Introduction of the SDD
- Connected group to the GitHub project
- Three subsystems and seventeen modules were thought up and the user manual was finished.

  Attendance: perfect attendance

Meeting 2 (03/22/19): (online meeting)

- Worked on the subsystems, modules, processes, and data entities decomposition descriptions
- Team decided to merge the Registered Apps database and 3rd Party Users database into a single database named **3rd Party Apps database**
- Team realized that HealthOut has an additional secondary actor, being the API of Microsoft Azure, which hosts our **Users database.**

  Attendance: N/A (online meeting)

Meeting 3 (03/29/19):

- Discussed options for databases: merged all databases into one external database.
- Began constructing an ERD for the application.
- Finalized plans for data organization within the application.
- Reevaluated group participation and noted the apparent withdrawal of one of our members.
- Added 18th module to descriptions

  Attendance: perfect attendance

Meeting 4 (04/05/19):

- Traceability table was made
- Use case and Sequence diagrams were made
- Interface descriptions were made
- Detailed design was made, finished the SDD

  Attendance: perfect attendance

Responsibilities

Aaron Jordan
In charge of refactoring, and will work on User's and 3rd Party App subsystem.

Quinn Poyneer
In charge of HealthOut's database, and will work on User's and Health Goals subsystem.

Cody Price
In charge of establishing access with 3rd party app APIs, will work on 3rd Party App and Health Goals subsystems.

Rafal Ryczek
In charge of graphic design, layouts, and will work on User's and Health Goals subsystems.