<Team 50>

<PuffAway>
Software Design Specification

Version <3.0>

Revision History

Date	Version	Description	Author
25/03/2020	3.0	Completed Section 1	Hung Truong
01/04/2020	3.0	Completed Section 2	Yusra Irfan & Abdulaziz Chalya
03/04/2020	3.0	Completed Section 3	Yusra Irfan, Abdulaziz Chalya, Hashim Abu Sharkh, Spencer Vecile, and Hung Truong
05/04/2020	3.0	Completed Section 4	Hung Truong & Hashim Abu Sharkh
05/04/2020	3.0	Completed Section 5	Yusra Irfan, Abdulaziz Chalya, Hashim Abu Sharkh, Spencer Vecile, and Hung Truong

Table Of Contents

1. Introduction

1.1 Purpose of this document

- The purpose of this document is to showcase the thought process, design, and decisions made when creating the product. Its objective is to document the details and allow investors, clients, and those necessary to understand the product and its creation.

1.2 Scope of the development project

- The scope of the project is to create a self-help product that assists users in quitting or minimizing their vaping habits. As vaping habits grow in today's society and Communities, we wanted to create a product that will help those who want to change their bad habit. With this being said there are a few objectives our product has:
    1. Help users record/journal any of the components within vaping
    2. Walk users through the 4-step solution outlined by Dr.Jeffrey M. Schwartz book "The 4-Step Solution"
    3. Help users to set goals and achieve them
    4. Motivate and congratulate users upon their path to quitting

With these objectives in mind, we provide the benefit of having a safe and secure system in which users can use to achieve their goals. With the use of technology, people will not need someone to keep track of their habits or actions, instead, they can rely on the software/application that we provide.

1.3 Definitions, acronyms, and abbreviations

CPU - A central processing unit (**CPU**), also called a central processor or main processor, is the electronic circuitry within a computer that executes instructions that make up a computer program.

IOS - **IOS** is one of the most popular mobile operating system developed and created by Apple Inc

ML - Machine Learning is an **application** of Artificial Intelligence (AI) which empowers software to learn, explore, and envisage outcomes automatically without human interference

UI - The user interface (**UI**) is the point of human-computer interaction and communication in a device. This can include display screens, keyboards, a mouse and the appearance of a desktop.

UX - Short form of User Experience, which means the overall experience of a person using a product such as a website or computer application, especially in terms of how easy or pleasing it is to use.

1.4 References

Software Requirement Specification (SRS):

https://drive.google.com/file/d/1ZRyNSnbYmCcmd1XGDJAsiJG25vhrSloj/view?usp=sharing

User Manual:

https://docs.google.com/document/d/148OSDa-aUJCiGvcTtUfAmFMUvW9fb4Evx5nUvKh0hMI/edit?usp=sharing

1.5 Overview of document

The following is a brief description of what can be found in this document.

Logical Architecture:

The logical architecture section shows what the application should do with an overall use case diagram as well as the architecture of our application. To show the architecture of the application we have drawn a package diagram and explained what the different sections are and how they interact with the backend.

Detailed Description Of Component :

As the name of this section indicates, it describes the component using structural and behavioural diagrams. It also describes the interactions between classes and how the non-relational database is structured.

Design Rationale:

The design rationale explains the thought process that the team had while making design decisions of the application. This includes the stack and other key factors whilst developing the product.

Review Forms:

This section includes how the team members have stayed reliable, communicated clearly, does more than what is asked, and stayed committed when completing this project.

2. LOGICAL ARCHITECTURE

2.1 Overview

Figure A: Main Use Case Diagram

2.2 Puffaway

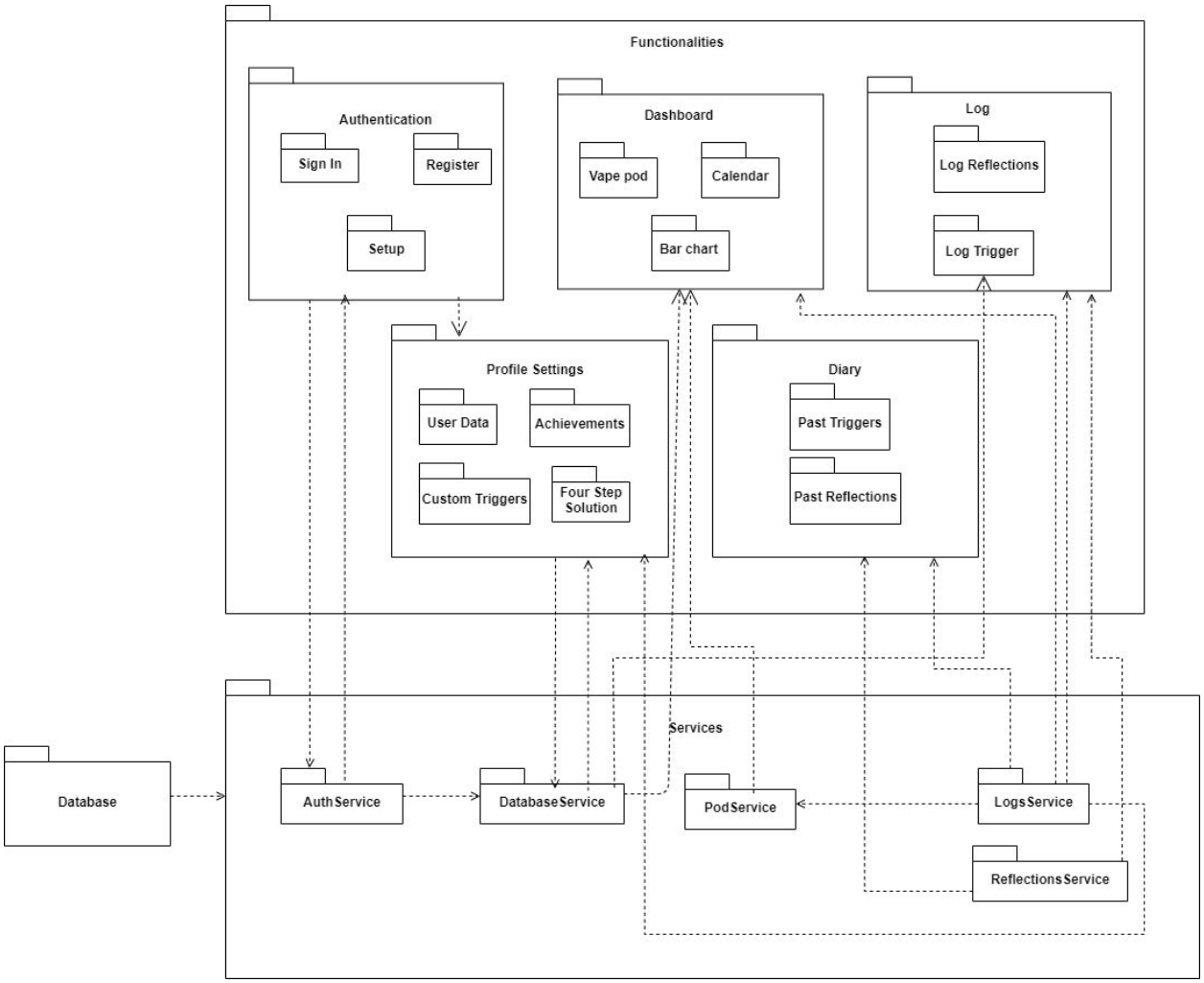

Figure B: Package Diagram

- Database and Services
    - Services contain functions that communicate with the firebase database.
- Our application has 5 main functionalities.
    - Authentication contains 3 components: sign in, register, and the setup page. Each component requires user information to authenticate them by communicating with the database. AuthService allows the components to send and receive information about the user.
    - Dashboard contains 3 components: the vape pod, calendar, bar chart. Vape pod information is received from the current goal of the user, which is received from the database services. Calendar information is received from the logs services that contains a list of dates that the triggers were logged at to display. Bar chart information is received from thePodService which communicates with the database to get the information for the user
    - Log contains two components: log triggers and log reflection. Logging triggers requires

Logs services to send the triggers to the database and Database service to get the custom triggers from the user. Logging reflections requires Reflection services to send the reflections to the database.

- Profile Settings has 4 components: user data, achievements, custom triggers, four step solution. The user data component receives the data from the database service to display to the user, and for the user to change their personal information, the data is sent to the database through the database service as well. Achievements depend on the users last trigger log, to receive that information we use log services. To add a custom trigger to the database, the information is sent through the logServices. Four step solutions do not depend on any service but is an essential part of the personal settings functionality
- Diary has two components. it displays the Trigger and reflections logs. Trigger logs depend on the Logs services to get the list of users triggers and the reflection logs depend on the reflection services to get a list of reflections from the database.

3. DETAILED DESCRIPTION OF COMPONENTS

3.1 PuffAway Class Diagrams

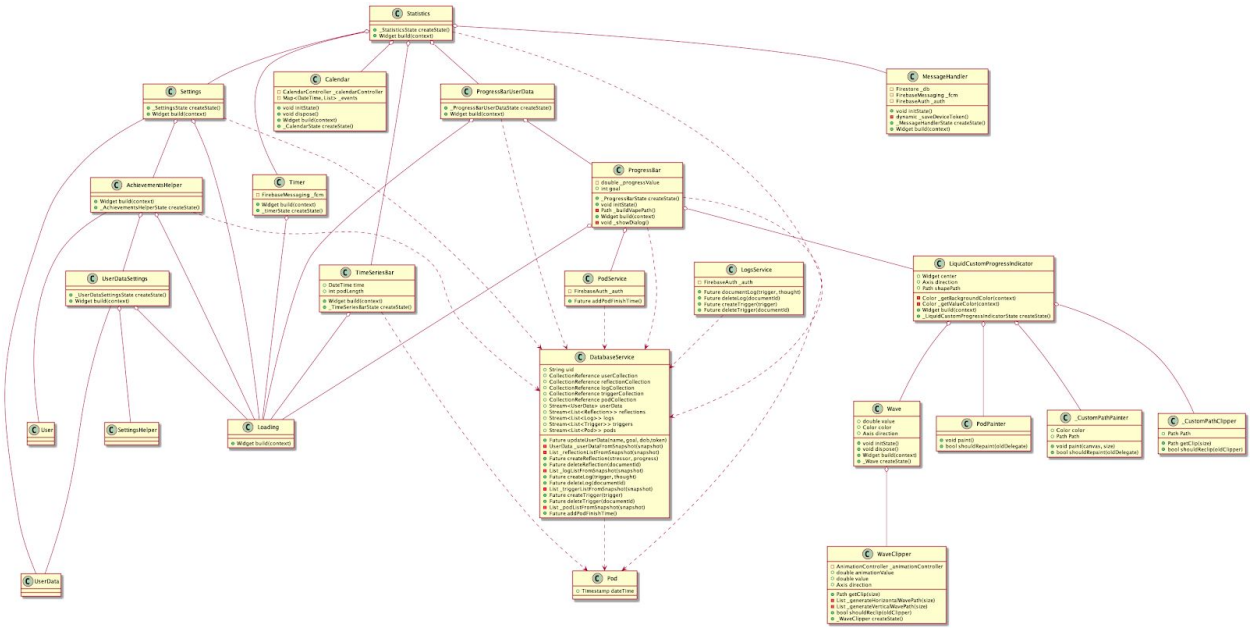

Figure D: Dashboard Class Diagram

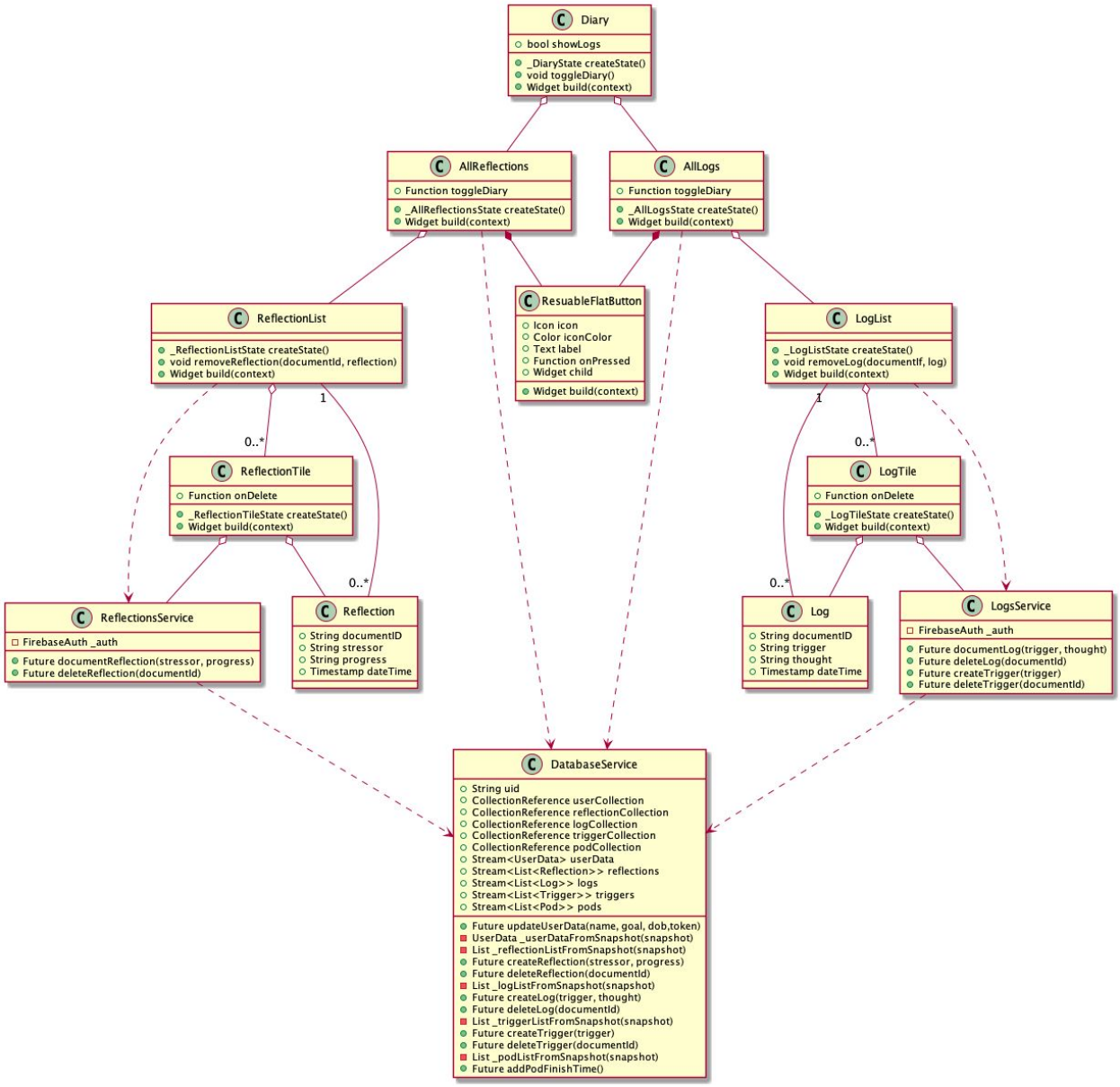

Figure E: Diary Class Diagram

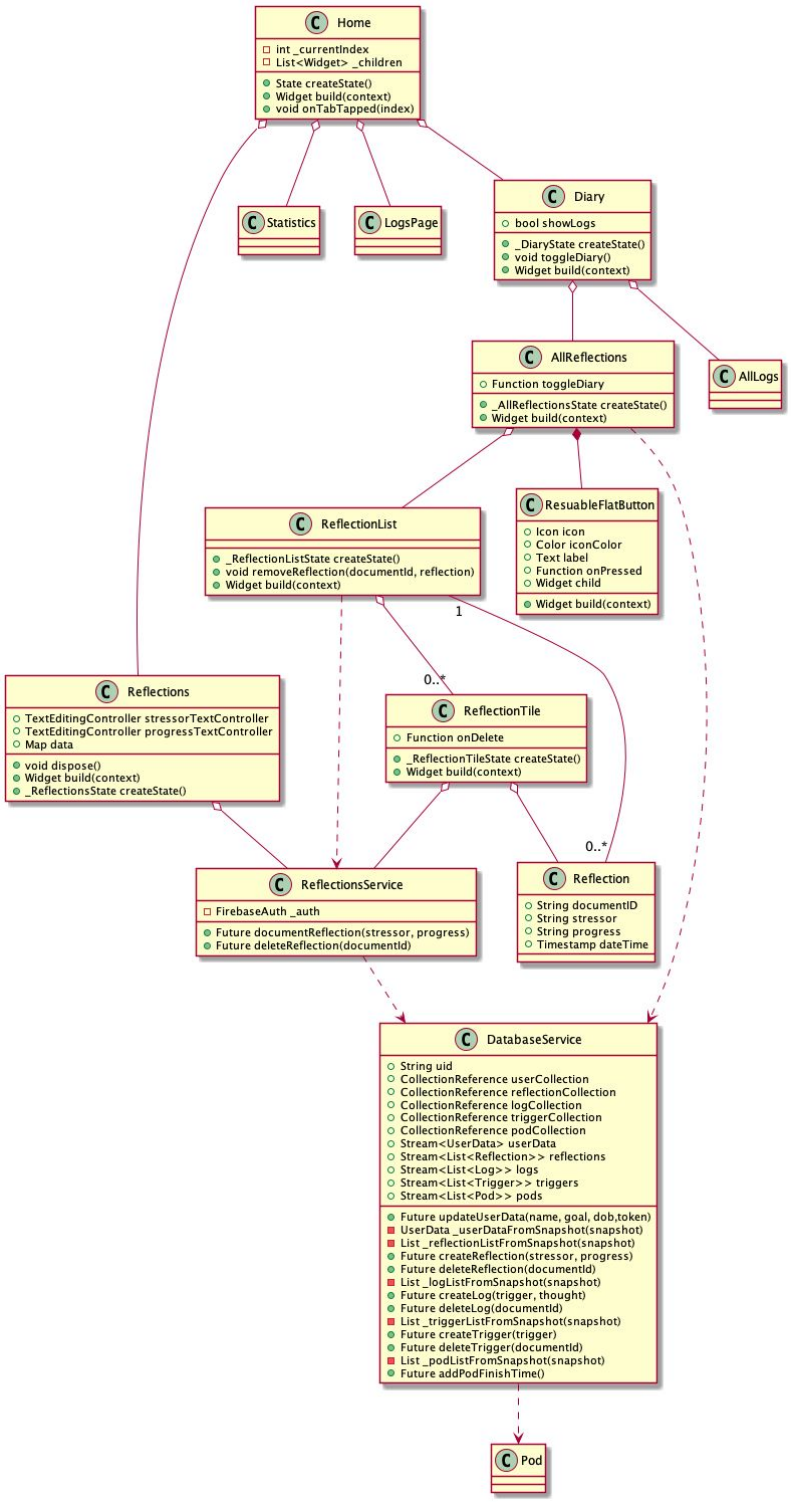

Figure F: Logging Reflection Class Diagram

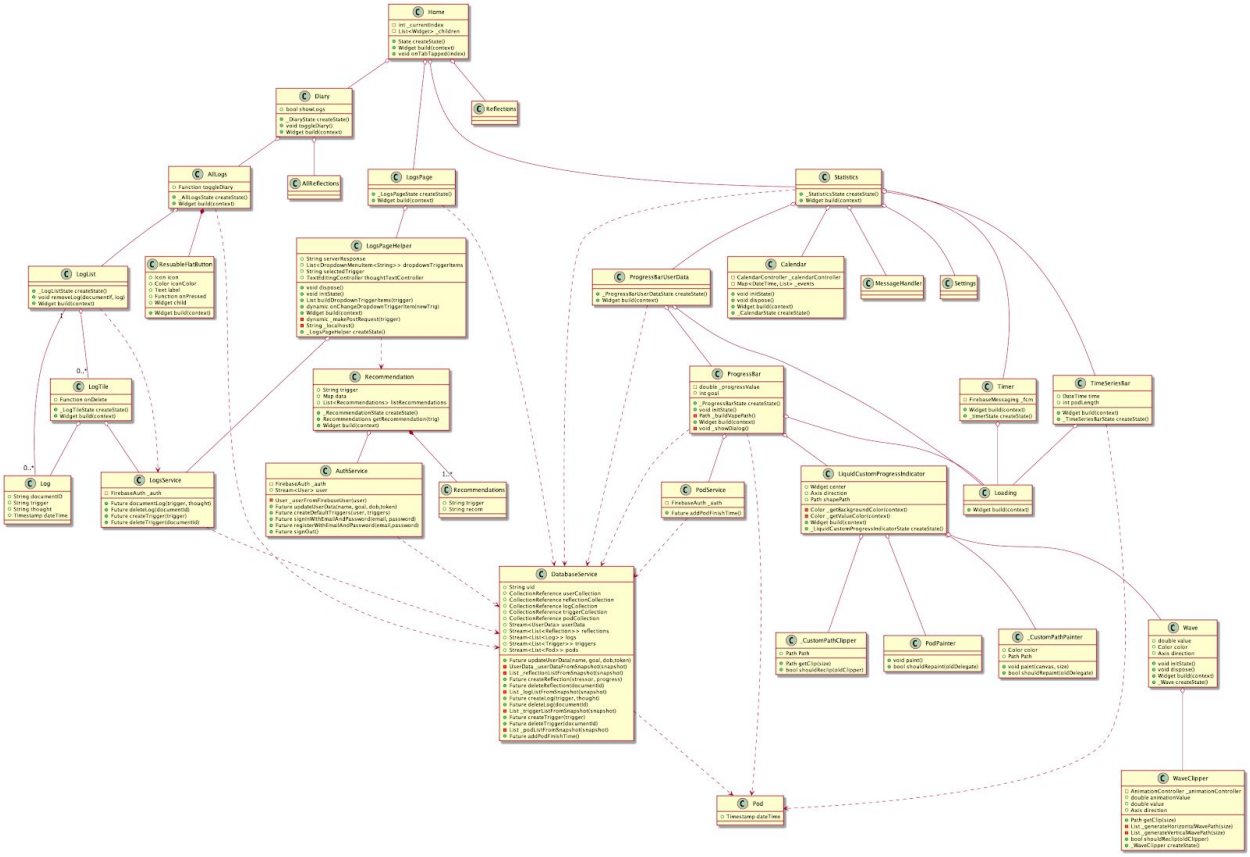

Figure G: Logging Trigger Class Diagram

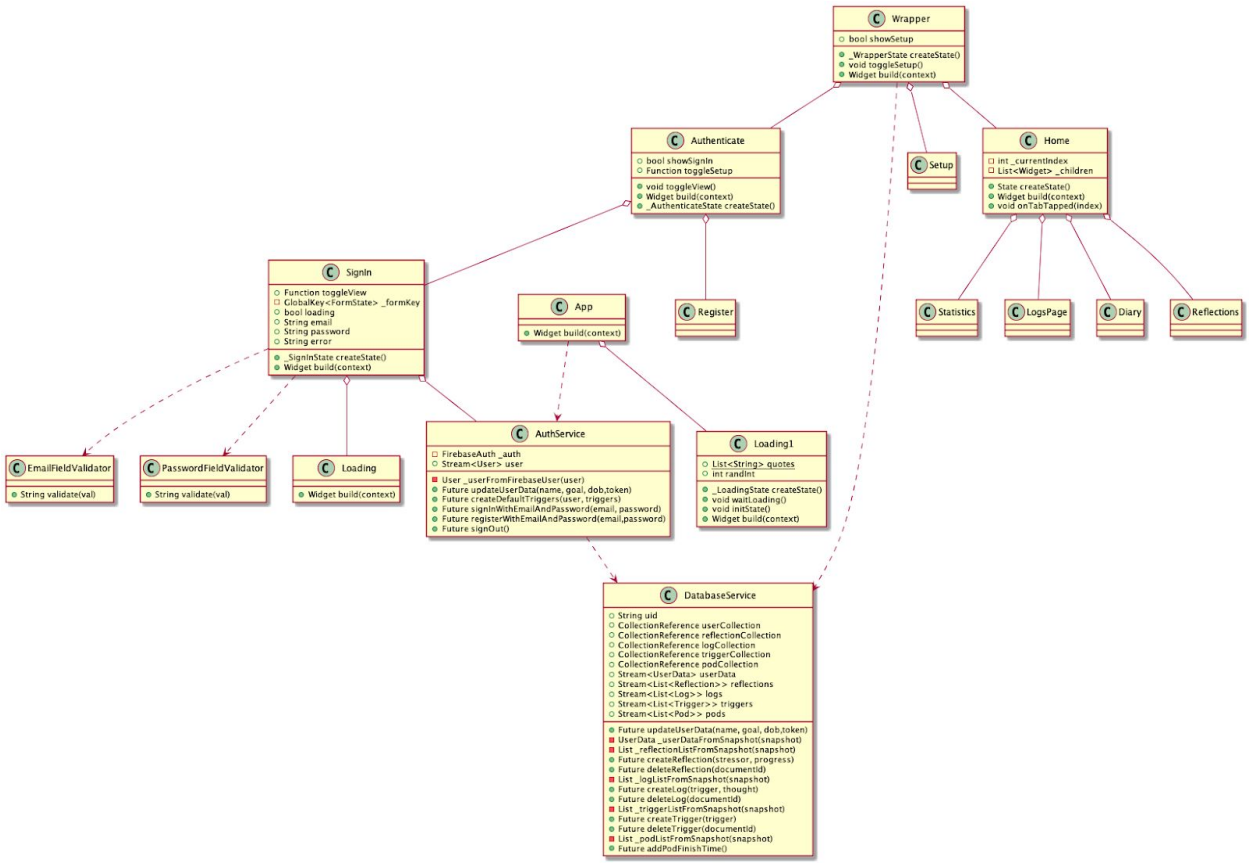

Figure H: Login Class Diagram

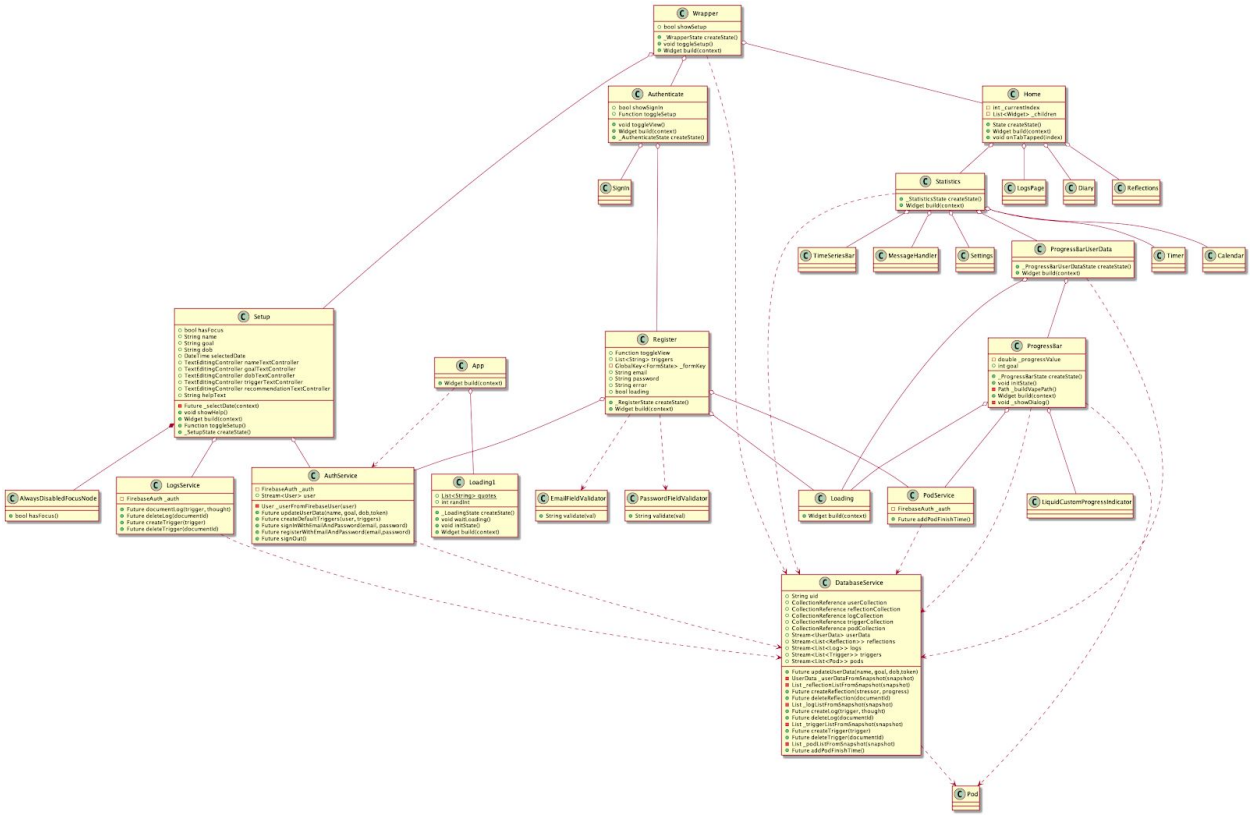

Figure I: Register Class Diagram

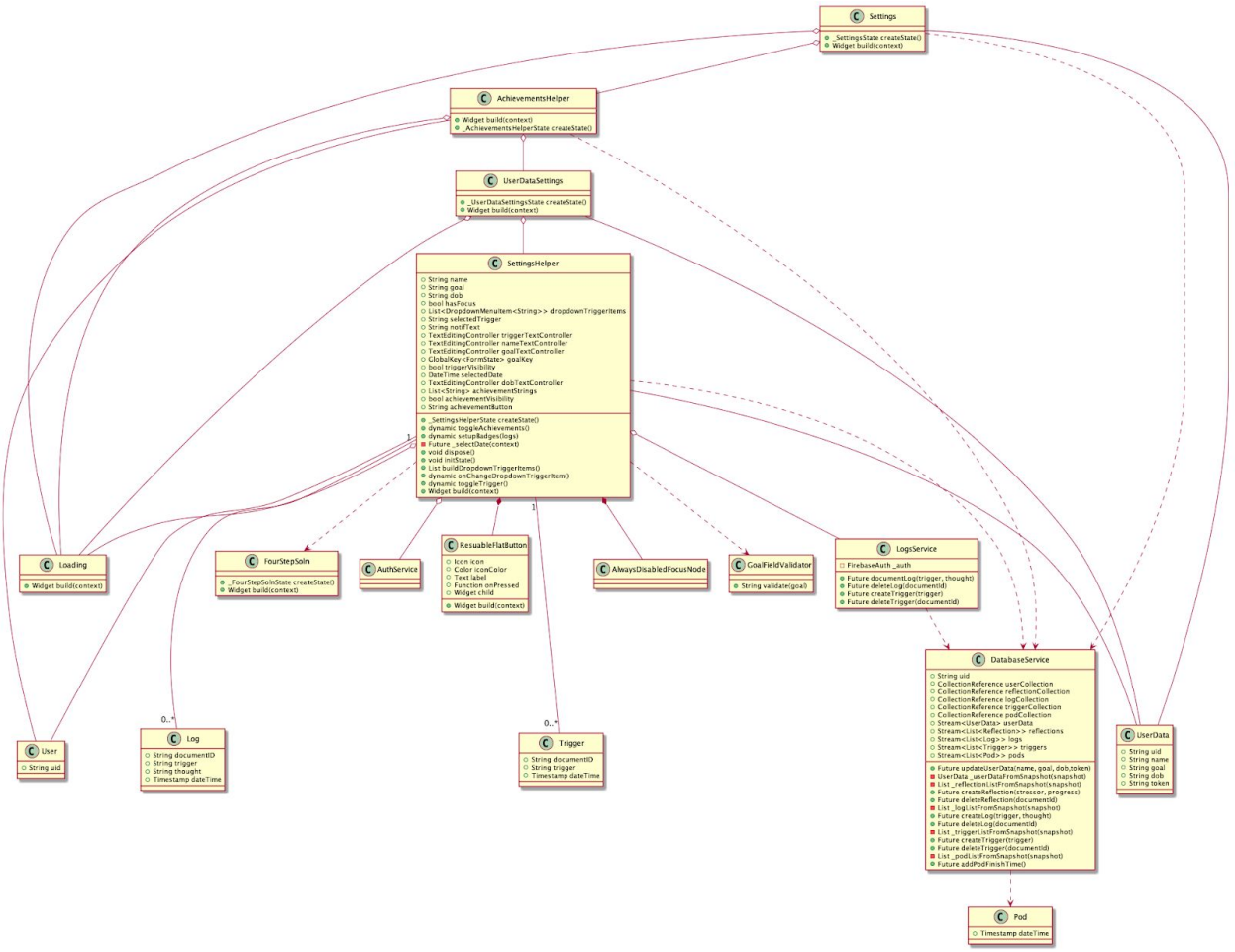

Figure J: Settings Class Diagram

3.1.1 Achievements ~ Component

- Grabs the current signed in user and sends in the logs to the settings to show the badges accordingly

3.1.1.1 Achievements - Attributes

- No Attributes

3.1.1.2 Achievements - Operations

Name	Description
build(context)	flutter widget function to build the widget tree of that component. The function sends in the data for the user logs or calls the loading widget if the user doesn't exist .
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.1.3 Achievements Design Specification/Constraints

- None

3.1.1.4 Achievements - States and Transitions

- Check “Figure G: Settings” in the Appendix

3.1.2 UserDataSettings ~ Component

- If the user exists it sends in the user data to the settings page when calling the component, other wise it loads to the loading widget

3.1.2.1 UserDataSettings - Attributes

- No attributes

3.1.2.2 UserDataSettings - Operations

Name	Description
createState()	built in flutter constructor to initialize the class. This is in every widget component.
build(context)	flutter widget function to build the widget tree of that component. The function gets the user data from the context of the page and if the user exists it is called the Settings widget, otherwise it calls the loading widget. This is checked constantly and as soon as the user if found, the settings widget is called using the user data.

3.1.2.3 UserDataSettings - Design Specification/Constraints

- None

3.1.2.4 UserDataSettings - States and Transitions

- No Diagram

3.1.3 ReflectionList ~ Component

- Uses a list view builder and user data to make a separate ReflectionTiles widget for each

reflection logged by the user. It also allowed the user to remove a reflection from the database.

3.1.3.1 ReflectionList - Attributes

- No attributes

3.1.3.2 ReflectionList - Operations

Name	Description
createState()	built in flutter constructor to initialize the class. This is in every widget component.
removeReflection(documentId)	Parameters: document id (reflection doc id) and Reflection service (to call a function from the database) This function calls the reflection service class's deleteReflection function to delete a specific reflection using the id.
build(context)	flutter widget function to build the widget tree of that component. The function gets the reflections from the provider and makes the ReflectionTile widget for each item in the list

3.1.3.3 ReflectionList - Design Specification/Constraints

- No constraints

3.1.3.4 ReflectionList States and Transitions

- No diagram

3.1.4 AllReflections ~ Component

- This component only exists to use a stream provider to get a reflections list of the user. This widget has a child widget of ReflectionList that will use the reflections logs by getting it from the provider. The component also has a button to toggle between trigger logs and reflection logs

3.1.4.1 AllReflections - Attributes

Name	Type	Description
toggleDiary	Function	Used to toggle between reflections and trigger logs. This function attribute was passed in from the Diary component.

3.1.4.2 AllReflections - Operations

Name	Description
createState()	built in flutter constructor to initialize the class. This is in every widget component.
build(context)	flutter widget function to build the widget tree of that component. The function uses a stream provider to get a reflections list of the user. This widget has a body of ReflectionList that will use the reflections logs by getting it from the provider

3.1.4.3 AllReflections - Design Specification/Constraints

- No constraints

3.1.4.4 AllReflections - States and Transitions

- Check “Figure L: Diary” in the Appendix

3.1.5 Authenticate ~ Component

- Component to check the authentication status of the user. Shows the login/register pages if the user isn't signed in.

3.1.5.1 Authenticate - Attributes

Name	Type	Description
showSignIn	bool	If the user is signing in or not (registering instead)
toggleSetup	Function	Used to figure out if the setup page should show up or not depending on if the user is registering or signing in.

3.1.5.2 Authenticate - Operations

Name	Description
toggleView()	toggle between signin page and register page
createState()	built in flutter constructor to initialize the class. This is in every widget component.
build(context)	flutter widget function to build the widget tree of that component. The function uses the showSignIn variable to decide if it should load the sign in widget or the register widget

3.1.5.3 Authenticate - Design Specification/Constraints

- No constraints

3.1.5.4 Authenticate - States and Transitions

- No diagram

3.1.6 Calendar ~ Component

- Display the calendar based on specific events

3.1.6.1 Calendar - Attributes

Name	Type	Description
_calendarController	CalendarController	Used in the imported TableCalendar widget to control what happens to the displayed calendar
_events	Map<DateTime, List>	To mark events on the calendar. In our case, we will mark the trigger logs.

3.1.6.2 Calendar - Operations

Name	Description
initState()	built in function to initialize variables in a widget component when its initialized
dispose()	build in function to edit specific variables or functions when the widget

	or component is disposed.
createState()	built in flutter constructor to initialize the class. This is in every widget component.
build(context)	Flutter widget function to build the widget tree of that component. Gets the logs of the user from the provider and marks the calendar with events. TableCalendar widget is then called. If the logs don't exist, loading widget is shown.

3.1.6.3 Calendar - Design Specification/Constraints

- None

3.1.6.4 Calendar - States and Transitions

- Check “Figure J: Dashboard Calendar” in the Appendix

3.1.7 ReusableFlatButton ~ Component

- Component to use whenever a flat button is needed.

3.1.7.1 ReusableFlatButton - Attributes

Name	Type	Description
icon	Icon	Icon of the button
iconColor	Color	color of the button
label	Text	text label in the button
onPressed	Function	on pressed function of the button
child	Widget	child of the button if needed

3.1.7.2 ReusableFlatButton - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component. if the button has an icon, it displays an icon flat button otherwise it displays a normal flat button with all the attributes.

3.1.7.3 ReusableFlatButton - Design Specification/Constraints

- None

3.1.7.4 ReusableFlatButton - States and Transitions

- No diagram

3.1.8 Timer ~ Component

- Component to show time since last hit on the screen and subscribe to firebase notifications based on that time

3.1.8.1 Timer - Attributes

Name	Type	Description
fcm	FirebaseMessaging	reference to firebase messaging to subscribe to different notifications based on the timer

3.1.8.2 Timer - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component. Displays a container with the time last hit. this value is calculated in this function too
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.8.3 Timer - Design Specification/Constraints

- None

3.1.8.4 Timer - States and Transitions

- Check “Figure K: Dashboard Vape Pod” in the Appendix

3.1.9 TimeSeriesBar ~ Component

- Component to make the bar chart for how long the pods are lasting for the user

3.1.9.1 TimeSeriesBar - Attributes

Name	Type	Description
time	DateTime	represents the pod start date
podLength	int	pod used in days.

3.1.9.2 TimeSeriesBar - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component. Return bar chart (Pod start date vs Duration lasted in days)
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.9.3 TimeSeriesBar - Design Specification/Constraints

- The values aren’t extremely different so they display properly in one graph

3.1.9.4 TimeSeriesBar - States and Transitions

- No Diagram

3.1.10 Loading ~ Component

- Component for the loading page. This loading page shows quotes when the app is first started

3.1.10.1 Loading - Attributes

Name	Type	Description
quotes	List<String>	A list of quotes to choose from
randint	int	random number generated to choose a quote from the list.

3.1.10.2 Loading - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component. Shows the quote and a loading spin kit
createState()	built in flutter constructor to initialize the class. This is in every widget component.
waitLoading()	Shows the loading page for 3 secs
initState()	calls the waitLoading function when the widget is initiated

3.1.10.3 Loading - Design Specification/Constraints

- The app loads within 3 seconds

3.1.10.4 Loading - States and Transitions

- No Diagram

3.1.11 Pod ~ Model

Model for a pod. This is used to create a pod and retrieve one from the database

3.1.11.1 Pod - Attributes

Name	Type	Description
dateTime	TimeStamp	represents the pod start date

3.1.11.2 Pod - Operations

- No functions

3.1.11.3 Pod - Design Specification/Constraints

- Must use data types available on Firebase

3.1.11.4 Pod - States and Transitions

- No Diagram

3.1.12 User ~ Model

- Model to retrieve a user from firebase authentication

3.1.12.1 User - Attributes

Name	Type	Description
uid	String	unique id for the user thats authenticated

3.1.12.2 User - Operations

- No Operations

3.1.12.3 User - Design Specification/Constraints

- Must use data types available on Firebase

3.1.12.4 User - States and Transitions

- No Diagram

3.1.13 UserData ~ Model

- Model for the user in the firebase database. This is connected to the authenticated user using the uid

3.1.13.1 UserData - Attributes

Name	Type	Description
uid	String	unique id for the user from the database
name	String	name of the user
goal	String	current qoal of the user
dob	String	date of birth of the user
token	String	token used to send notifications

3.1.13.2 UserData - Operations

- No operations

3.1.13.3 UserData Design Specification/Constraints

- Must use data types available on Firebase

3.1.13.4 UserData - States and Transitions

- No diagram

3.1.14 Trigger ~ Model

- Model for the trigger entered by the user

3.1.14.1 Trigger - Attributes

Name	Type	Description
documentId	String	unique id for the trigger from the database
trigger	String	Trigger entered by the user
dateTime	TimeStamp	time the trigger was added

3.1.14.2 Trigger - Operations

- No operations

3.1.14.3 Trigger - Design Specification/Constraints

- Must use data types available on Firebase

3.1.14.4 Trigger - States and Transitions

- No diagram

3.1.15 App ~ Component

- Main Component to run the app

3.1.15.1 App - Attributes

- No attributes

3.1.15.2 App - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component. Return The loading quote widget to start the app

3.1.15.3 App - Design Specification/Constraints

- None

3.1.15.4 App - States and Transitions

- No diagram

3.1.16 LogList ~ Component

- Uses a list view builder and list of logs to make a separate LogTiles widget for each trigger logged by the user. It also allowed the user to remove a logged trigger from the database.

3.1.16.1 LogList - Attributes

- No attributes

3.1.16.2 LogList - Operations

Name	Description
createState()	built in flutter constructor to initialize the class. This is in every widget component.
removeLog(documentId)	Parameters: document id (reflection doc id) and Log service (to call a function from the database) This function calls the log service class's deleteLogfunction to delete a specific trigger log using the id.
build(context)	flutter widget function to build the widget tree of that component. The function gets the log from the provider and makes the LogTile widget for each item in the list

3.1.16.3 LogList - Design Specification/Constraints

- None

3.1.16.4 LogList - States and Transitions

- Check "Figure L: Diary" in the Appendix

3.1.17 AllLogs ~ Component

- This component exists to use a stream provider to get a reflections list of the user. This widget has a child widget of LogList that will use the trigger logs by getting it from the provider. The component also has a button to toggle between trigger logs and reflection logs

3.1.17.1 AllLogs - Attributes

Name	Type	Description

toggleDiary	Function	Used to toggle between reflections and trigger logs. This function attribute was passed in from the Diary component.

3.1.17.2 AllLogs - Operations

Name	Description
createState()	built in flutter constructor to initialize the class. This is in every widget component.
build(context)	flutter widget function to build the widget tree of that component. The function uses a stream provider to get a trigger list of the user. This widget has a body of LogsList that will use the trigger logs by getting it from the provider

3.1.17.3 AllLogs - Design Specification/Constraints

- None

3.1.17.4 AllLogs - States and Transitions

- Check “Figure L: Diary” in the Appendix

3.1.18 LogsPage ~ Component

- Component has a stream provider to provide a list of custom triggers for the user. The provider will be called from the child widget LogsPageHelper, which will prompt the user to log the trigger

3.1.18.1 LogsPage - Attributes

- No attributes

3.1.18.2 LogsPage - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component. Calls the LogPageHelper as a child widget.
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.18.3 LogsPage - Design Specification/Constraints

- None

3.1.18.4 LogsPage - States and Transitions

- No diagram

3.1.19 ProgressBarUserData ~ Component

- Component to provide user data to the ProgressBar component through a stream provider

3.1.19.1 ProgressBarUserData - Attributes

- No attributes

3.1.19.2 ProgressBarUserData - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component. Uses a stream provider to provide user data for its child component ProgressBar.
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.19.3 ProgressBarUserData - Design Specification/Constraints

- None

3.1.19.4 ProgressBarUserData - States and Transitions

- No diagram

3.1.20 Wave ~ Component

- Component that shows the wave with the help of WaveClipper Component

3.1.20.1 Wave - Attributes

Name	Type	Description
value	double	how much the pod is filled
color	Color	color of the wave in the pod
direction	Axis	controls the animation of the wave

3.1.20.2 Wave - Operations

Name	Description
initState()	built in function to initialize variables in a widget component when it's initialized. Initializes the animation controller
dispose()	build in function to edit specific variables or functions when the widget or component is disposed. disposes the animation controller
createState()	built in flutter constructor to initialize the class. This is in every widget component.
build(context)	Flutter widget function to build the widget tree of that component. Animates the wave by using curves. It uses ClipPath widget to make the wave

3.1.20.3 Wave - Design Specification/Constraints

- Pod has to be big enough to see the wave animation clearly

3.1.20.4 Wave - States and Transitions

- Check “Figure K: Dashboard Vape Pod” in the Appendix

3.1.21 WaveClipper ~ Class

- Class that extends CustomClipper (imported from flutter material library)

3.1.21.1 WaveClipper - Attributes

Name	Type	Description
animationValue	double	the value of the animation
value	double	pod value - percentage
direction	Axis	Direction of the wave
animationController	AnimationController	to control the animation

3.1.21.2 WaveClipper - Operations

Name	Description
getClip(size)	gets the size as a parameter, checks the direction and generates either horizontal or vertical wave path by calling their respective functions
createState()	built in flutter constructor to initialize the class. This is in every widget component.
_generateHorizontalWavePath(size)	returns a list of horizontal coordinates to draw
_generateVerticalWavePath(size)	returns a list of vertical coordinates to draw
shouldReclip(oldClipper)	checks if there is a new value for the animation using the old clipper

3.1.21.3 WaveClipper - Design Specification/Constraints

- Pod has to be big enough to see the wave animation clearly

3.1.21.4 WaveClipper - States and Transitions

- No Diagram

3.1.22 PodPainter ~ Class

- Class to paint to the pod, extends custom painter

3.1.22.1 PodPainter - Attributes

- No attributes

3.1.22.2 PodPainter - Operations

Name	Description
paint()	paints the new pod using the canvas and paint widgets provided by flutter
shouldRepaint(oldDelegate)	returns if the new pod should be painted or not by checking the old delegate

3.1.22.3 PodPainter - Design Specification/Constraints

- None

3.1.22.4 PodPainter - States and Transitions

- Check “Figure K: Dashboard Vape Pod” in the Appendix

3.1.23 LiquidCustomProgressIndicator ~ Component

- Component to make the pod component of the dashboard

3.1.23.1 LiquidCustomProgressIndicator - Attributes

Name	Type	Description
center	Widget	The widget to show in the center of the progress indicator.
direction	Axis	The direction the liquid travels.
shapePath	Path	The path used to draw the shape of the progress indicator. The size of the progress indicator is controlled by the bounds of this path.

3.1.23.2 LiquidCustomProgressIndicator - Operations

Name	Description
_getBackgroundColor(context)	gets the background color of the app
_getValueColor(context)	gets accent color of the app
build(context)	Flutter widget function to build the widget tree of that component. Builds the full vape pod by calling different widgets
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.23.3 LiquidCustomProgressIndicator - Design Specification/Constraints

- None

3.1.23.4 LiquidCustomProgressIndicator - States and Transitions

- No diagram

3.1.24 Wrapper ~ Components

- Component to wrap the app, this checks if the user is authenticated or not, if not authenticated it shows the Authenticate component. If the user is authenticated, it chooses between either showing the setup page of the Home page, depending on if the user logged in or just registered.

3.1.24.1 Wrapper - Attributes

Name	Type	Description
showSetup	bool	boolean to show setup or not, used in toggleSetup function

3.1.24.2 Wrapper - Operations

Name	Description
toggleSetup()	to toggle the setup variable depending on if the user is on the register page or sign in
build(context)	Flutter widget function to build the widget tree of that component. This widget returns by checking if the user is authenticated or not, if

	not authenticated it shows the Authenticate component. If the user is authenticated, it chooses between either showing the setup page of the Home page, depending on if the user logged in or just registered.
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.24.3 Wrapper - Design Specification/Constraints

- None

3.1.24.4 Wrapper - States and Transitions

- No diagram

3.1.25 Home ~ Components

- Component to show the navigation bar at the bottom of the screen. This is called when the user logs in

3.1.25.1 Home - Attributes

Name	Type	Description
currentIndex	int	Represents the index of the page to show
_children	List<Widget>	Widgets in the nav bar

3.1.25.2 Home - Operations

Name	Description
onTapped()	changes the currentIndex based on the icon tapped
build(context)	Flutter widget function to build the widget tree of that component. Returns the bottom navigation bar and the current tapped widget
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.25.3 Home - Design Specification/Constraints

- There's few enough pages so that they all fit comfortable in the navigation bar

3.1.25.4 Home - States and Transitions

- No diagram

3.1.26 Statistics ~ Components

- Component to make the dashboard. It includes all the components, for example the pod, calendar, graph, etc

3.1.26.1 Statistics - Attributes

- No attributes

3.1.26.2 Statistics - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component.

	Returns the dashboard with all the components, for example the pod, calendar, graph, etc
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.26.3 Statistics - Design Specification/Constraints

- None

3.1.26.4 Statistics - States and Transitions

- No Diagram

3.1.27 FourStepSoln ~ Components

- Component to show the instructions and 4 step solutions. Called from the settings page.

3.1.27.1 FourStepSoln - Attributes

- No attributes

3.1.27.2 FourStepSoln - Operations

Name	Description
build(context)	Flutter widget function to build the widget tree of that component. Return a widget with text components that explain the application
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.27.3 FourStepSoln - Design Specification/Constraints

- None

3.1.27.4 FourStepSoln - States and Transitions

- No Diagram

3.1.28 Diary ~ Components

- Component to allow the user to see past logs and reflections

3.1.28.1 Diary - Attributes

Name	Type	Description
showLogs	bool	determines if logs or reflections page is showing

3.1.28.2 Diary - Operations

Name	Description
toggleDiary()	method to switch the view between logs and reflections
build(context)	built in flutter constructor to build the widget. This is in every widget component.

3.1.28.3 Diary - Design Specification/Constraints

- None

3.1.28.4 Diary - States and Transitions

- Check “Figure L: Diary” in the Appendix

3.1.29 Setup ~ Components

- Component to assist the user when setting up their account and initial settings for the app

3.1.29.1 Setup - Attributes

Name	Type	Description
hasFocus	bool	used for the dob selection to see if it is clicked or not
name	string	name of the user
goal	string	the user defined goal in days for how long they would like a pod to last
dob	string	the date of birth of the user
selectedDate	DateTime	the currently selected date on the datepicker
nameTextController	TextEditingController	text field controller
goalTextController	TextEditingController	text field controller
dobTextController	TextEditingController	text field controller
triggerTextController	TextEditingController	text field controller
recommendationTextController	TextEditingController	text field controller
helpText	string	text explaining how to use the setup page
toggleSetup	Function	function passed into constructor so setup page wont be visible again after initial setup is complete

3.1.29.2 Setup - Operations

Name	Description
_selectDate()	method to select date from date picker pop up
showHelp()	function to show help text explaining how to use setup page
build(context)	built in flutter constructor to build the widget. This is in every widget component.
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.29.3 Setup - Design Specification/Constraints

- None

3.1.29.4 Setup - States and Transitions

- Check “Figure O: Register” in the Appendix

3.1.30 MessageHandler ~ Components

- Component that allows push notifications to be sent to the user

3.1.30.1 MessageHandler - Attributes

Name	Type	Description
_db	Firestore	contains a link to our firestore instance
_fcm	FirebaseMessageing	contains a link to the firebase messaging console
_auth	FirebaseAuth	contains a link to out firebase authentication instance

3.1.30.2 MessageHandler - Operations

Name	Description
initState()	initialises widget and configures default notification settings
build(context)	built in flutter constructor to build the widget. This is in every widget component.
_saveDeviceToken()	saves device token to database so notifications can be sent to individual devices
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.30.3 MessageHandler - Design Specification/Constraints

- None

3.1.30.4 MessageHandler - States and Transitions

- No Diagram

3.1.31 DatabaseService ~ Class

- Class that allows communication with the database

3.1.31.1 DatabaseService - Attributes

Name	Type	Description
uid	String	ID of the users that is generated by Firebase
userCollection	Firestore	Reference to the user collection in Firestore
reflectionCollection	Firestore	Reference to the reflection collection in Firestore
logCollection	Firestore	Reference to the logs collection in Firestore
triggerCollection	Firestore	Reference to the triggers collection in Firestore
podCollection	Firestore	Reference to the pods collection in Firestore

3.1.31.2 DatabaseService - Operations

Name	Description
DatabaseService({uid})	Constructor used to get the uid of the connected user
updateUserData()	Updates the user data in the database
_userDataFromSnapshot (snapshot)	Map the data from the retrieved snapshot into a UserData type

userData	Returns data about the logged in user back from the database
_reflectionListFromSnapshot(snapshot)	Map the data from the retrieved snapshot into a list of Reflections
reflections	Returns all the user's reflections from the database
createReflections(stressor,progress)	Creates the reflection in the diary using the stressor and progress inputted
deleteReflection()	Deletes the reflection once swiped by the user
_logListFromSnapshot(snapshot)	Map the data from the retrieved snapshot into a list of Logs
logs	Returns all the user's logs from the database
createLogs(trigger,thought)	Creates the log in the diary using the trigger and thought inputted
deleteLog()	Deletes the log once swiped by the user
_triggerListFromSnapshot(snapshot)	Map the data from the retrieved snapshot into a list of Triggers
triggers	Returns all the user's triggers from the database
createTrigger(trigger)	Creates a trigger in the database for the user
deleteTrigger()	Deletes the trigger
_podListFromSnapshot(snapshot)	Map the data from the retrieved snapshot into a list of Pods
pods	Returns all the user's pod information from the database
addPodFinishTime()	Used to update the pod finish time once the pod icon is clicked

3.1.31.3 DatabaseService - Design Specification/Constraints

- None

3.1.31.4 DatabaseService - States and Transitions

- No Diagram

3.1.32 Settings ~ Component

- Component that creates and manages the profile settings page

3.1.32.1 Settings - Attributes

Name	Type	Description
name	String	Holds name of User
goal	String	Holds User's goal
dob	String	Holds User's date of birth
hasFocus	Bool	Manages the focus of the page, as in what textbox is currently activated
_log	LogsService	Contains all the logs of the User
dropdownTriggerItems	List<DropdownMenuItem<String>>	Contains all the triggers of the User, default and custom
selectedTrigger	String	Holds trigger selected from dropdown
notifText	String	Text to output when this component sends a notification
triggerTextController	TextEditingController	Controller to manage trigger dropdown list

nameTextController	TextEditingController	Controller to manage name textbox
goalTextController	TextEditingController	Controller to manage goal textbox
goalKey	GlobalKey<FormState>	Key to manage the form that contains the goal textbox
triggerVisibility	Bool	Flag for hiding/showing input text boxes for adding a new trigger
selectedDate	DateTime	Holds User's date of birth in DateTime format
achievementStrings	List<String>	List of badge icon file names
achievementVisibility	Bool	Flag for hiding the entire page when showing achievements
achievementButton	String	Text on the button to hide/show achievements

3.1.32.2 Settings - Operations

Name	Description
setupBadges(logs)	Set which badge to be unlocked and which to be locked
build(context)	Built in flutter constructor to build the widget. This is in every widget component.
toggleAchievements()	Hide/show achievements
createState()	Built in flutter constructor to initialize the class. This is in every widget component.
_selectDate	Manage calendar when selecting date of birth
dispose()	Clean up widget when closing the settings page
initState()	Setup state when opening the page
buildDropdownTriggerItems()	Populate triggers dropdown list
onChangeDropdownTriggerItem()	Update selectedTrigger string when trigger is picked
toggleTrigger()	Hide/show input text boxes to add a custom trigger

3.1.32.3 Settings - Design Specification/Constraints

- None

3.1.32.4 Settings - States and Transitions

- Check "Figure G: Settings" in the Appendix

3.1.33 LogsPageHelper ~ Component

- Component to construct frontend widget for logging page

3.1.33.1 LogsPageHelper - Attributes

Name	Type	Description
_log	LogsService	contains all the User's logs
dropdownTriggerItems	List<DropdownMenuItem<String>>	List of all the User's triggers
selectedTrigger	String	contains selected trigger from dropdown
thoughtTextController	TextEditingController	Controller for thought input text box

3.1.33.2 LogsPageHelper - Operations

Name	Description
initState()	initialises widget when page is called
build(context)	built in flutter constructor to build the widget. This is in every widget component.
dispose()	cleans up widget when leaving the page
createState()	built in flutter constructor to initialize the class. This is in every widget component.
buildDropdownTriggerItems(trigger)	Fill in the dropdown menu with User's triggers
_makePostRequest(trigger)	Send selected trigger to the database
localhost()	returns address of flask app

3.1.33.3 LogsPageHelper - Design Specification/Constraints

- None

3.1.33.4 LogsPageHelper - States and Transitions

- Check "Figure N: Logging Trigger" in the Appendix

3.1.34 LogTile ~ Component

Component that shows one trigger log and its descriptions

3.1.34.1 LogTile - Attributes

Name	Type	Description
log	Log	Contains one trigger log of the user
onDelete	Function	Instance of on delete function to delete that trigger log from the diary
_log	LogService	Instance of the LogService class to do operations on the log using the database

3.1.34.2 LogTile - Operations

Name	Description
build(context)	built in flutter constructor to build the widget. This is in every widget component. This function will show a card with one log and its descriptions
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.34.3 LogTile - Design Specification/Constraints

- None

3.1.34.4 LogTile - States and Transitions

- Check "Figure N: Logging Trigger" in the Appendix

3.1.35 Log ~ Model

- Service class that connects the front end to the database

3.1.35.1 Log - Attributes

Name	Type	Description
documentId	String	Id of trigger log, generated by the database
trigger	String	The name of the trigger
thought	String	The thought of the trigger
dateTime	TimeStamp	The time the trigger was logged

3.1.35.2 Log - Operations

- No operations as it's a model

3.1.35.3 Log - Design Specification/Constraints

- Data types must be available in Firebase

3.1.35.4 Log - States and Transitions

- No Diagram

3.1.36 LogsService ~ Class

- Service class that connects the front end to the database

3.1.36.1 LogsService - Attributes

Name	Type	Description
_auth	FirebaseAuth	Contains an instance to the authenticated user

3.1.36.2 LogsService - Operations

Name	Description
documentLog (trigger, thought)	Function to add a trigger log in the database
deleteLog(documentId)	Function to delete a login the database using its id
createTrigger(trigger)	to create a custom trigger for the user
deleteTrigger(documentId))	to delete a custom trigger that the user added

3.1.36.3 LogsService - Design Specification/Constraints

- None

3.1.36.4 LogsService - States and Transitions

- No Diagram

3.1.37 Reflections ~ Component

- Component to add reflections to the diary. It contains different text fields

3.1.37.1 Reflections - Attributes

Name	Type	Description
stressorTextController	TextEditingController	text field controller for the stressor
progressTextController	TextEditingController	text field controller for the progress
_reflection	ReflectionsService	Instance of the RefectionsService class to do operations on the reflections using the database

3.1.37.2 Reflections - Operations

Name	Description
build(context)	built in flutter constructor to build the widget. This is in every widget component. This function will allow the user to enter reflections.
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.37.3 Reflections - Design Specification/Constraints

- None

3.1.37.4 Reflections - States and Transitions

- Check “Figure M: Logging Reflection” in the Appendix

3.1.38 ReflectionsService ~ Class

- Component to add reflections to the diary. It contains different text fields

3.1.38.1 ReflectionsService - Attributes

Name	**Type**	**Description**
_auth	FirebaseAuth	Contains an instance to the authenticated user

3.1.38.2 ReflectionsService - Operations

Name	Description
documentReflection(stressor, progress)	Function to add a reflection log in the database
deleteReflection(documentId)	Function to delete a reflections log in the database using its id

3.1.38.3 ReflectionsService - Design Specification/Constraints

- None

3.1.38.4 ReflectionsService - States and Transitions

- No Diagram

3.1.39 ReflectionTile ~ Component

- Component that shows one Reflection log and its descriptions

3.1.39.1 ReflectionTile - Attributes

Name	Type	Description
reflection	Reflection	Contains one reflection log of the user
onDelete	Function	Instance of on delete function to delete that reflection log from the diary
_reflection	ReflectionsService	Instance of the ReflectionsService class to do operations on the reflection log using the database

3.1.39.2 ReflectionTile - Operations

Name	Description
build(context)	built in flutter constructor to build the widget. This is in every widget component. This function will show a card with one reflection log and its descriptions
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.39.3 ReflectionTile - Design Specification/Constraints

- None

3.1.39.4 ReflectionTile - States and Transitions

- Check “Figure L: Diary” in the Appendix

e

3.1.40 Reflection ~ Model

- Service class that connects the front end to the database

3.1.40.1 Reflection - Attributes

Name	Type	Description
documentId	String	Id of trigger log, generated by the database
stressor	String	The stressor of the reflection
progress	String	The progress of the reflection
dateTime	TimeStamp	The time the reflection was logged

3.1.40.2 Reflection - Operations

- No operations as its a model

3.1.40.3 Reflection - Design Specification/Constraints

- Data types must exist in Firebase as well

3.1.40.4 Reflection - States and Transitions

- No Diagram

3.1.41 Sign In ~ Component

- Component that shows the sign in page

3.1.41.1 Sign In - Attributes

Name	Type	Description
_auth	AuthService	Contains the authentication details
toggleView	Function	Instance of togglView function th
_formKey	GlobalKey<FormState>	Reference to the login form
loading	bool	States if the page should show the loading widget or not
email	String	reference to the email field
password	String	reference to the password field
error	String	to display the error

3.1.41.2 Signin - Operations

Name	Description
build(context)	built in flutter constructor to build the widget. This is in every widget component. It displays the input form and the submit button
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.41.3 SignIn - Design Specification/Constraints

- Must validate email and password format

3.1.41.4 Signin - States and Transitions

- Check “Figure Q: Sign In” in the Appendix

3.1.42 AuthService ~ Class

- Class to get the authenticated user from the database

3.1.42.1 AuthService - Attributes

Name	Type	Description
_auth	FirebaseAuth	contains a link to out firebase authentication instance
user	Stream<User>	Instance of the user provided by the database

3.1.42.2 AuthService - Operations

Name	Description
_userFromFirebaseUser(user)	gets the user from the database
updateUserData(name, goal,	updates the data for the user

dob,token)	
createDefaultTriggers(user, triggers)	creates all the default triggers for a new user
signInWithEmailAndPassword (email, password)	signs in the user with the email and [password provided in the parameters
registerWithEmailAndPasswor d(email,password)	registers the user with the email and [password provided in the parameters
signOut()	Signs out the user

3.1.42.3 AuthService - Design Specification/Constraints

- None

3.1.42.4 AuthService - States and Transitions

- No Diagram

3.1.43 PodService ~ Class

- Service class to communicate with the database regarding the pod information for the user

3.1.43.1 PodService - Attributes

Name	Type	Description
_auth	FirebaseAuth	contains a link to out firebase authentication instance

3.1.43.2 PodService - Operations

Name	Description
addPodFinishTime()	adds a new finish time for the pod for the specific user

3.1.43.3 PodService - Design Specification/Constraints

- None

3.1.43.4 PodService - States and Transitions

- No Diagram

3.1.44 ProgressBar ~ Component

- Component that shows the progress for the vape pod

3.1.44.1 ProgressBar - Attributes

- No attributes

3.1.44.2 ProgressBar - Operations

Name	Description
build(context)	built in flutter constructor to build the widget. This is in every widget component. It shows the wave on the vape pod but getting the information of the user

createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.44.3 ProgressBar - Design Specification/Constraints

- None

3.1.44.4 ProgressBar - States and Transitions

- Check “Figure K: Dashboard Vape Pod” in the Appendix

3.1.45 Register ~ Component

- This is the registration page that handles new users registering for the first time. It satisfies the functional requirement of creating a user.

3.1.45.1 Register - Attributes

Name	Type	Description
email	String	This email attribute is used to uniquely identify the user when registering
password	String	The password attribute is used to verify the user
error	String	Used to display a message when the user enters an account that is already being used or an error.
_pod	PodService	to set the initial state of the pod
_auth	AuthService	Contains the authentication details
toggleView	Function	Instance of togglView function th
_formKey	GlobalKey<FormState>	Reference to the login form
loading	bool	States if the page should show the loading widget or not

3.1.45.2 Register - Operations

Name	Description
build(context)	built in flutter constructor to build the widget. This is in every widget component. It shows the Register fields and buttons
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.45.3 Register - Design Specification/Constraints

- None

3.1.45.4 Register - States and Transitions

- Check “Figure O: Register” in the Appendix

3.1.46 Recommendation ~ Component

- Component that shows the recommendations on the page

3.1.46.1 Recommendation - Attributes

Name	Type	Description
trigger	String	trigger entered by the user to match a recommendation
data	Map	data from the backend
listRecommendations	List<Recommendations>	List of all the recommendations
rec	Recommendations	Reference to the recommendations object

3.1.46.2 Recommendation - Operations

Name	Description
getRecommendation(trig)	gets the recommendation from a list using the given trigger
build(context)	built in flutter constructor to build the widget. This is in every widget component. This will show the recommendation on the page
createState()	built in flutter constructor to initialize the class. This is in every widget component.

3.1.46.3 Recommendation - Design Specification/Constraints

- Machine learning server is running, otherwise it will use standard recommendations

3.1.46.4 Recommendation - States and Transitions

- Check “Figure N: Logging Trigger” in the Appendix

3.1.47 Recommendations ~ Model

- Model for the recommendations

3.1.47.1 Recommendations - Attributes

Name	Type	Description
trigger	String	trigger text
recom	String	corresponding recommendation of the trigger

3.1.47.2 Recommendations - Operations

- No operations

3.1.47.3 Recommendations - Design Specification/Constraints

- Data types must exist in Firebase as well

3.1.47.4 Recommendations - States and Transitions

- No Diagram

3.1.48 UserRepository ~ Class

- Class that handles user sign in and sign out

3.1.48.1 UserRepository - Attributes

Name	Type	Description
auth	FirebaseAuth	Firebase object to authorize users
_user	FirebaseUser	Firebase user
_status	Status	Status of user eg. Authenticating, Unauthenticated
status	Status	Enumerator of all possible statuses
user	FirebaseUser	Firebase user object

3.1.48.2 UserRepository - Operations

Name	Description
signIn(email, password)	authenticates user with their email and password
signOut()	signs user out of the app
onAuthStateChanged(firebaseUser)	checks if user variable is populated, and does so if needed

3.1.48.3 UserRepository - Design Specification/Constraints

- None

3.1.48.4 UserRepository - States and Transitions

- No Diagram

3.2 Interaction Diagrams

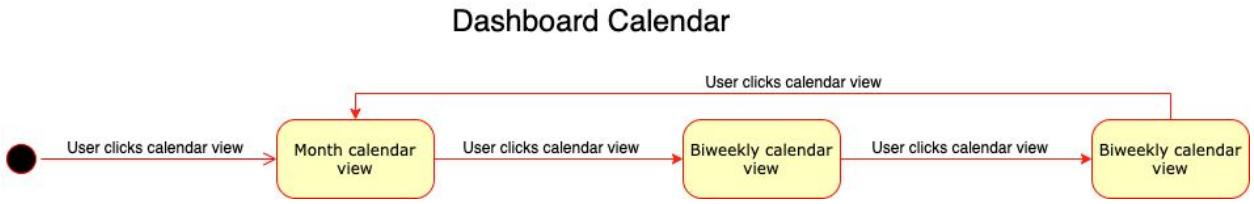

Figure J: Dashboard Calendar

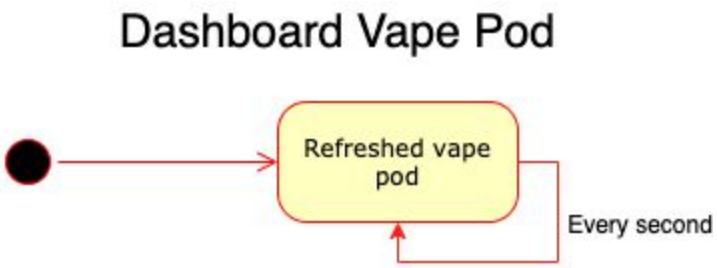

Figure K: Dashboard Vape Pod

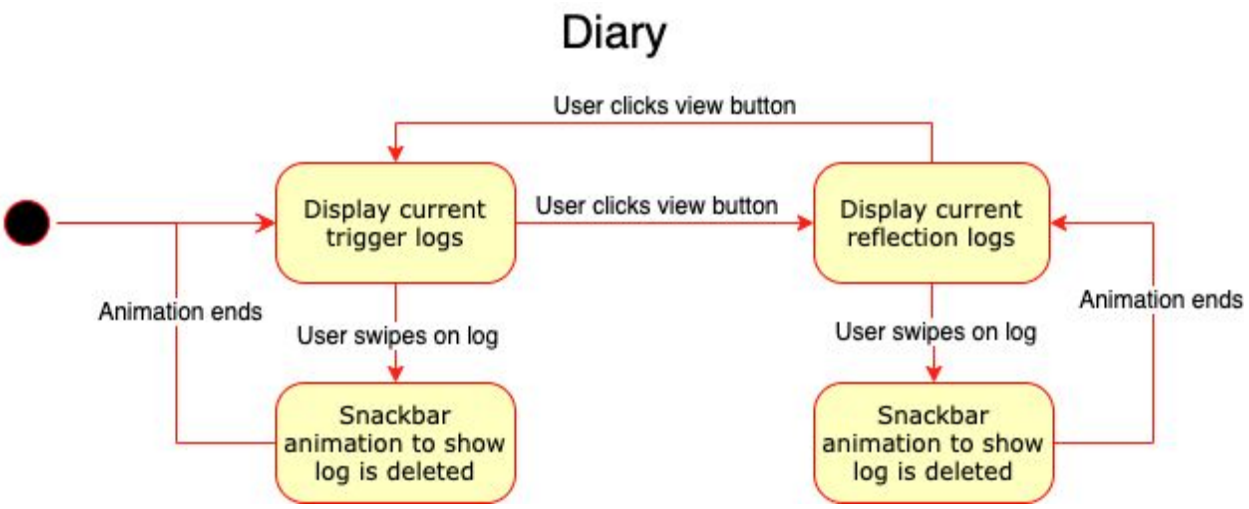

Figure L: Diary

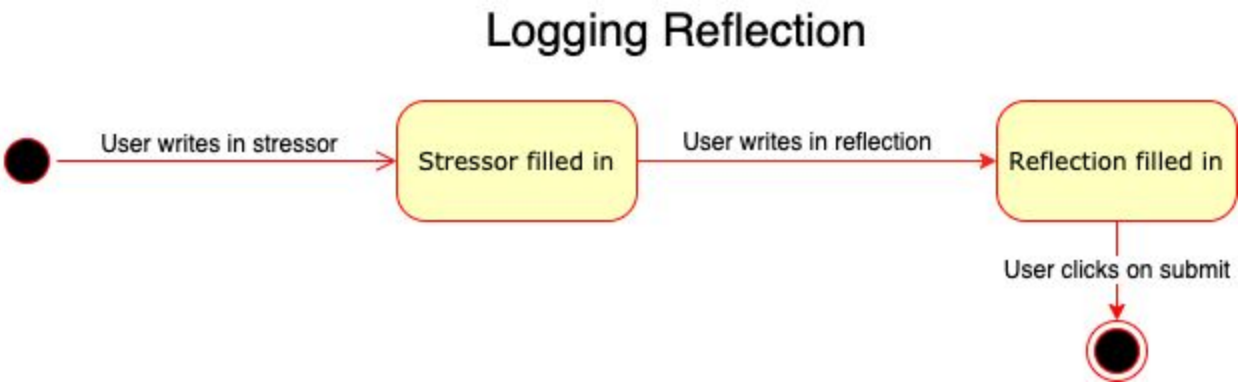

Figure M: Logging Reflection

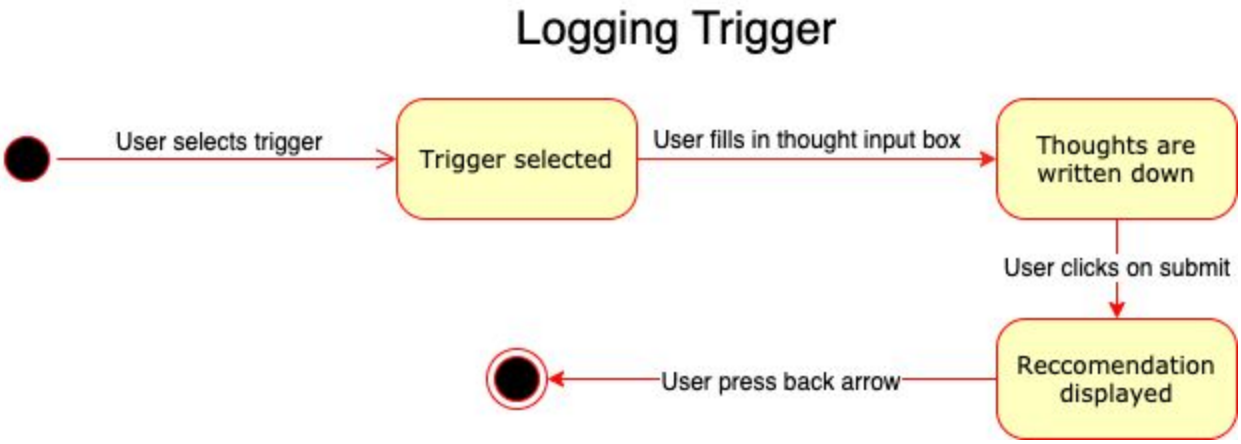

Figure N: Logging Trigger

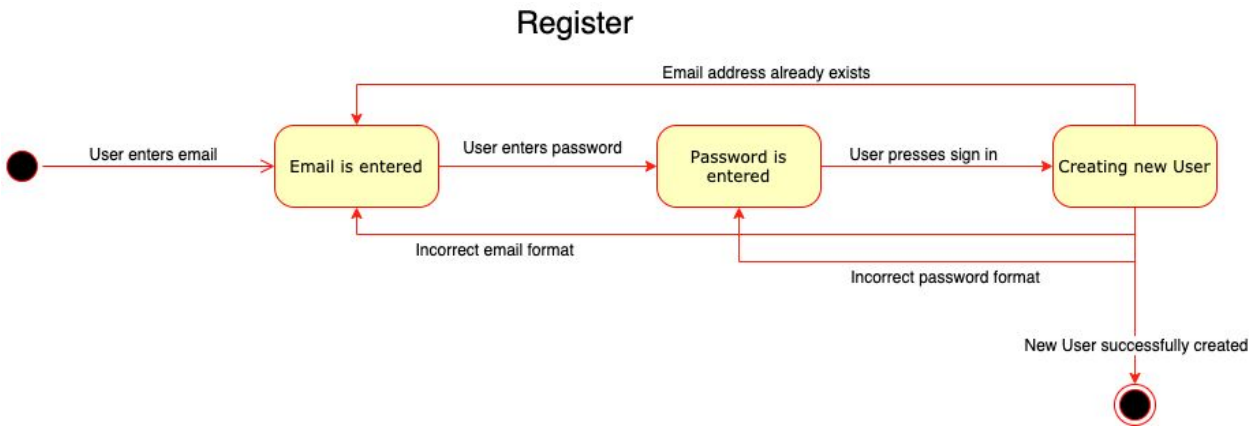

Figure O: Register

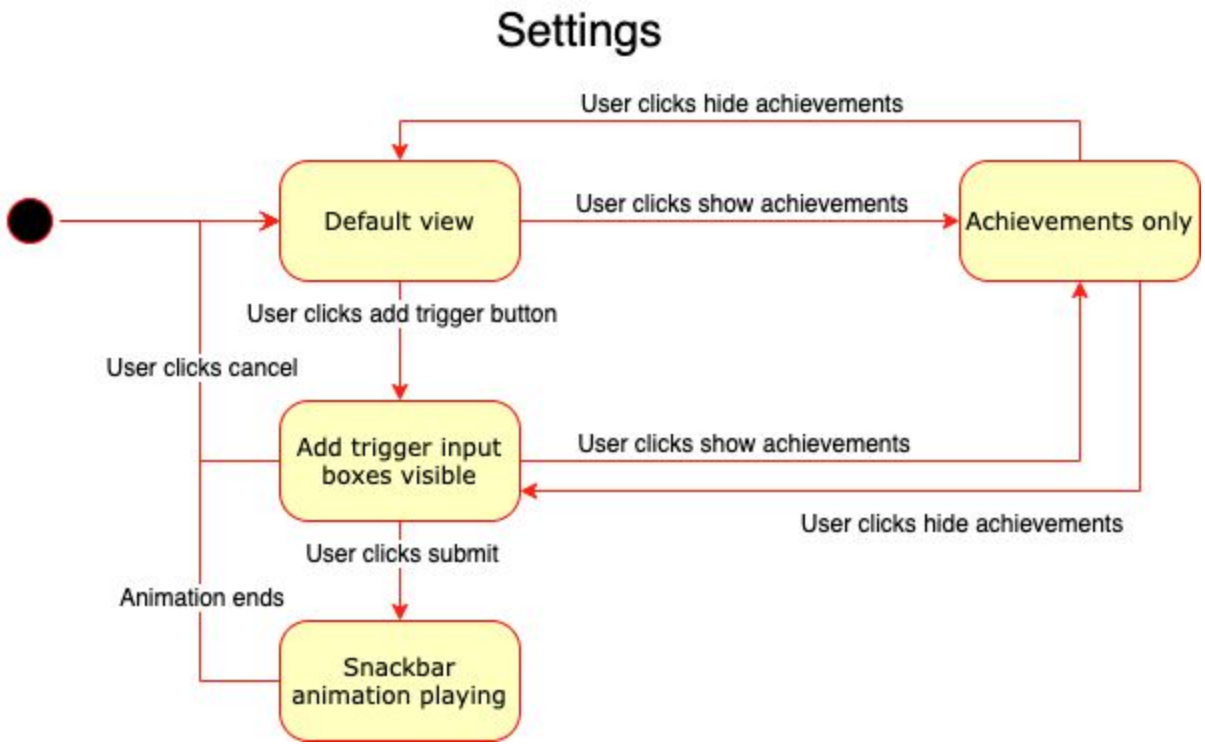

Figure P: Settings

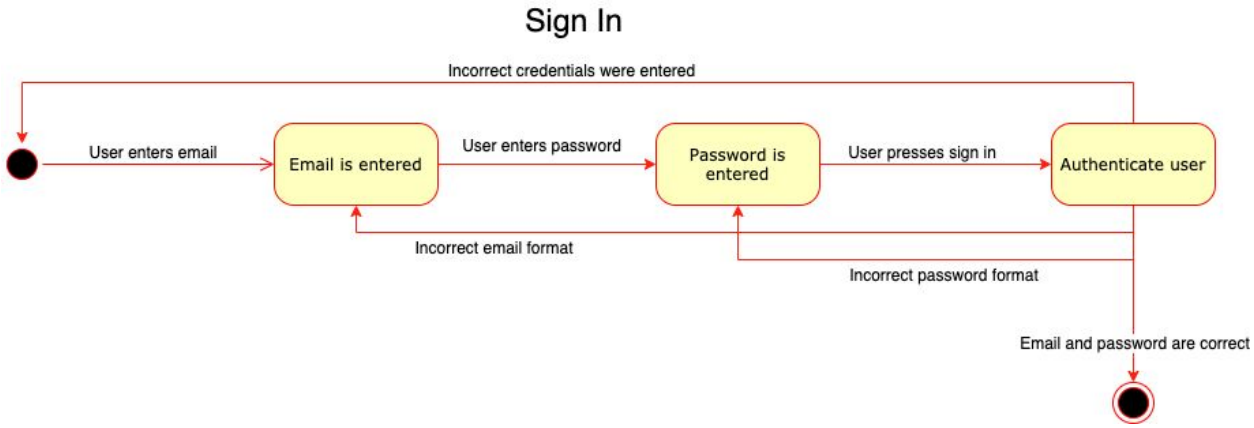

Figure Q: Sign In

3.3 Database Considerations & Schema

In the design process, both MySQL and Firebase were considered as our top choices for the backend. Traditional RDBMSs encounter challenges in handling big data, a feature that is needed in our application due to the projected amount of user activity. On the other hand, NoSQL provides alternative data stores that can handle this huge volume of data and scalability. It also has many more advantageous characteristics. NoSQL include the following characteristics:

- Simple and flexible non-relational data models
- Has the ability to scale horizontally over many commodity servers.
- Provide high availability
- Support BASE(Basically Available, Soft state, Eventually consistent) systems

This led us to choose Firebase. As NoSQL is not relational, normalization is a foreign concept and usually relations are created to increase speed of specific queries. However a conceptual ER diagram was created to aid with the creation of the backend. Firebase also stores passwords safely and hashes them, and does not allow access to them by the owners of the database.

3.3.1 PuffAway Non-relational Database Diagram:

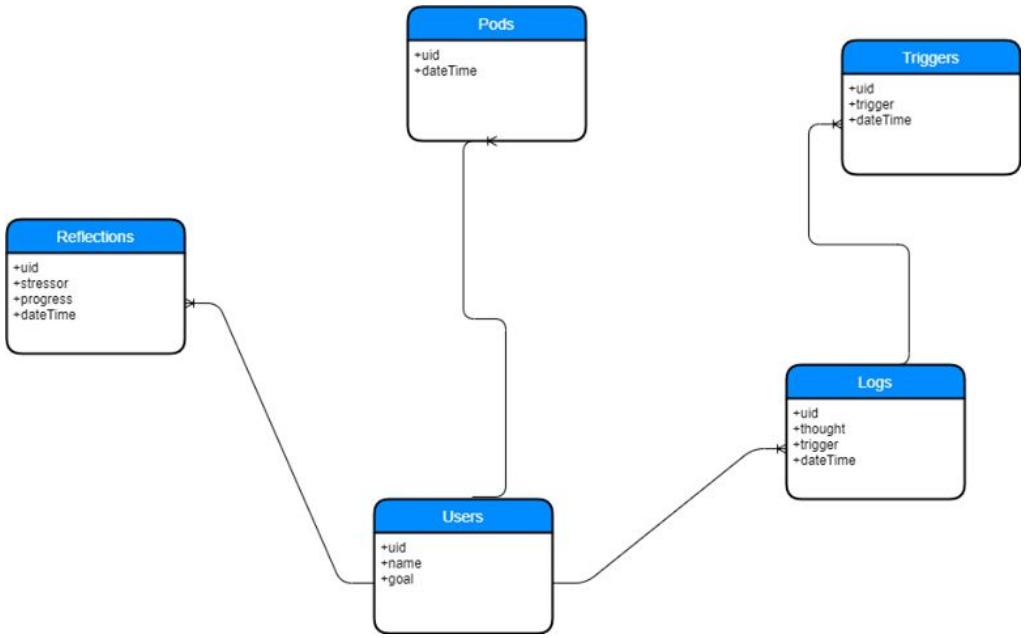

Figure R: Database Schema

4. DESIGN RATIONALE

In terms of design rationale, this product took a lot of thought and research to build. Some of the main requirements that played a part in our choices include security, portability, usability, robustness, and gamification.

Before diving deep into the design rationale for the software stack and component design, some of the major trade offs in terms of development should be addressed. The first tradeoff we would like to point out is maintainability vs efficiency, specifically a design that is very easy to maintain, may not be the most efficient. Our approach to making a decision about this trade off was quite simple, looking at what our application needed. Since we do not have high intensity functions that require top level of efficiency we decided that maintainability was more important. We wanted the code base and the development of the system to be easily maintained for the developers. This project was the first time the team has built a mobile application and we knew that our knowledge and experience would be very limited. Building something that is very efficient requires advanced knowledge, and understanding of the inner workings of applications. Since we could not gain that advanced knowledge without proper experience we decided that maintainability is more worthwhile.

Another design decision we had to make was in terms of generating data and storing data. Since our application required a record of actions a user has taken it made the most sense to store the data. One of the cons about storing data vs generating data was the memory limitations. Thankfully using a remote database such as Firebase/Firestore it is capable to scale up to our needs.

When designing the system and its components the team had a discussion of class responsibilities. Since we wanted to satisfy the requirement of maintainability, it was important for the team to have the same idea class roles. Some of the key distinctions were between model, provider, service, and component classes. These all had their specific roles in which the team knew about so that we could coordinate the development. Model classes were used to represent the data structure of the backend. This meant that they contained no functions and only attributes. Service classes handle retrieving information from the database and serve functionalities. For example we have a log service that has encapsulated all the functions regarding the database and logs. Components handle the frontend view of the component itself and use providers to get the information from the backend. Providers are the link between frontend components to the service classes that get the information from the backend. These were the key distinctions we made when structuring the codebase and application .

The following headings and sections are the other design decision that the team had to go through in the making of the application.

Buttons and text fields of the same style

A design choice that we made when styling and laying out the UI/UX was to use the same style of button and text fields. To do this we used Flutters constant widgets. They allow you to keep the same styling for any widget. We used this to ensure the usability of the application is similar to all pages. If a user is able to navigate the first page they will be able to navigate the rest. Our UI/UX also is made to look similar to generic applications. This is to ensure that we present things that are expected and nothing is out of the ordinary.

Gamification and Goals

A nonfunctional requirement we had was gamification. Having a reward/achievement system that acts as an external motivator for the user completing their goal is important so the user does not get bored. It also helps them stay motivated to complete their goal of minimizing or quitting vaping. To do this we had several ideas, we could give the user a character they can personalize, badges, or simple notifications. To keep things simple and still encourage the user we decided to have badges and simple notifications. We also have a visual vape pod that shows the user how far they are from their goal.

Another design decision we had to make was the goals. How were we going to measure the goal metric wise? We could have done the logging per hit and made the goal hits per day. It seemed consistent but when asking users about a goal like this in the surveys, they thought it was very limiting. Certain days it can rise or fall so realistically the hits per day are inconsistent from user to user. This is why we came up with another metric, the time it takes for the pod itself to last. If the user was taking longer time to finish the pod that means that they are vaping less. It is a very easy and intuitive way to measure the goal once again satisfying the usability aspect.

Vape pod & Visualization aspect

A design feature we are very proud of in this product is the visualization of the user's progress. As one of our requirements is usability, we thought that it would make the most sense if we had some intuitive components that our application could be known for. This intuitive component for us would be the virtual pod that mimics the amount of vape juice that should be left at a given time. Its ratio is determined by the goal that a user has set. For example, you wish to finish a pod within 2 weeks, after 1 week the pod will be at half of its capacity. The goal of this pod is to help you achieve your goal. The intuitive aspect is that a user should match their pod with this one. Since it is so visual it is very easy to compare the pod in real life to this virtual one. To ensure that it was usable we even asked some of our potential users that also helped us in a survey early in development. This feature was widely accepted and praised.

Flutter/Dart as Application Code Stack.

When choosing which application development stack to use we had a few important things in mind. Since we are building this application to help and reach the most number of people the requirement of portability is critical. Some of the options that we looked at in terms of development environment included Android Studio Code and Flutter/Dart. Comparing the two we quickly realized that Flutter/Dart was built for creating cross-platform native apps. The interesting idea of Flutter is that it's a UI Toolkit that is capable of adapting its views to the device that is hosting it. If it is on an Android phone vs IOS phone the look of each component will be different. It is even powerful enough to handle different screen sizes and ratios. If we stuck with Android Studios we would not have been able to build an application on a single code base and have it be available to both IOS and Android users. Although maintainability wasn't a key requirement in our product, it should be noted that choosing Flutter/Dart allowed for high maintainability. Since it is a single code base that can be used to make a native application, it would be easier to manage if we were to expand our development into a Swift codebase (for IOS) and Android Studio (for Android).

Firebase/Firestore for Backend Database

Another important consideration in terms of design involved having a local database or remote. We decided to use a remote database just because we thought it would be easier to

maintain all the data from different users. A remote database can be more powerful in the function, memory space, and CPU cycles it offers. Security-wise although local is the safest option, some remote options offered their own encryption. Some other alternatives that we considered were Dart's built-in database features, SQL, and Firebase/Firestore. Since Firebase/Firestore offers its own encryption and security that are used in the industry we went with it. Firebase/Firestore offers a secure HTTPS channel for passing data through. As well as its own encryption algorithm called Scrypt. The database also has its own built-in security which protects the resting data. It was the perfect fit for our requirements outlined in the SRS. Another reason why we decided to use Firebase/Firestore was for maintainability. Although that isn't a requirement, it was something we briefly considered. Since everything was to be connected to the database backend server, it would be easy to pinpoint errors instead of looking at the local storage of the program. Overall Firebase/Firestore was the best decision for us given our requirements and scenario.

StreamProviders vs StreamBuilders

Another key design decision was choosing between using StreamProviders and StreamBuilders. Both of these classes deal with retrieving data from the database but they both do it in different ways. When using a StreamProvider class, all descendent classes get access to the data Stream. Conversely, a class that contains a StreamBuilder only allows access to the data Stream from the same class. The main difference came down to if we wanted the data to flow through the whole app, parts of it or only in one class. This was decided each time data was retrieved from the database. For example, once a user logs in, we have a StreamProvider that provides the userID to other classes for authorization purposes. This had to be done since the userID was needed in other classes.

Machine Learning Considerations

The reason we wanted to use machine learning was just to give more personalization and robustness to the recommendations. Without the machine learning aspect some of the recommendations would not have made sense. So to do this we had to consider some tools such as language, server, and machine learning model.

Language: Python is a stable and flexible language that contains the necessary tools used by developers. It is simple, it is consistent. With all the complicated algorithms involved in machine learning, simplicity is needed. Python is also flexible, it offers the option to choose between object-oriented, functional programming and many more. There is also much more support for it in comparison to the other Machine languages (like R, matlab etc.) We discussed the advantages of R and how it has a comprehensive statistical analysis package but its steep learning curve prevented us from using it. In regards to matlab, its cost of license was a huge disadvantage.

Server: Flask Two languages were considered for the server, NodeJS, Flask and Django. As our machine learning model was trained in python, it was optimal to use either Flask or Django as it is easier to communicate with python from within Flask or Django. And between Flask and Django, Flask was chosen as it is a light and micro framework whereas Django is a full-stack web framework which we did not need.

Model: The pipeline I chose to construct forms the basis of a bag of words, tailored to the very small training dataset I had to construct the model with. After cleaning my data of english stopping words and removing stems and tenses to make similar words equivalent mathematically, I ran a TFIDF otherwise known as term frequency inverse document frequency optimization. This counts the frequency of words as well as the frequency it appears in the entire document as a mathematical inverse. I chose the ngram range of (1,1) as a parameter as it is recommended for smaller datasets with similar words. This

vectorizes my training data into the matrices which my ML algorithm can understand. Next, because of the multidimensional nature of NLP, I selected the kbest dimensions with words within it using the chi squared distribution. This was chosen as it easily picks out a sum of the squares of k independent standard normal random variables, which selects the single dimension which best captures the training data we want the model to learn from. Finally, now that we have made our data linear, we can perform a simple linear classification. The support vector classifier was chosen over the logistic regression as it is better used to deal with data that does not necessarily fall on the same plane (which is the more likely scenario given that our input vectors are difficult to interpret as is the case for deep learning in general). This creates a great classification model that will only improve with training data size based on the bag of words framework. The results of the models prediction was then used to map the triggers to the correct recommendation in our recommendation dataset.

5. REVIEW FORMS

5. 1 Yusra Irfan's Teamwork Retrospect

5.1.1 Always reliable.

According to me, being always reliable means doing whatever is needed to be done to be successful. Finishing work in a set time frame is key. Therefore, I always finished my work before the assigned time and checked the group chat to keep updated on the progress. Overall, I think I was a very reliable teammate. I will try my best to do work on time in the future and stay a reliable teammate.

5.1.2 Communicates with confidence.

Communicating ideas is an essential part of being in a team. I took the role of team lead in this project, so I communicate my ideas while asking others for theirs. Sometimes, when we talked about too many things, ideas got lost. So, we would make more detailed meeting minutes. I was usually concerned about everyone not feeling comfortable to communicate their ideas. Next time I will try to be more considerate of everyone and give them more constructive criticism while communicating with others.

5.1.3 Does more than asked.

Taking the role of team lead was very challenging and I still took it. Assigning tasks to everyone while taking into consideration various things, like expertise and other commitments took a lot of effort and time. I definitely did more than asked. Next time I will make a more concrete schedule for everyone to follow.

5.1.4 Adapts quickly and easily.

When we were writing the sprint retrospective each time, I reflected on what we did wring and need to improve on. I adapted quickly to the feedback and adapted. One of the biggest challenges was constantly referring and updating Jira. I could always quickly adapt to changes, especially to accomplish a specific goal, I hope to get better at it in the future by practicing and reflecting.

5.1.5 Displays genuine commitment.

I displayed genuine commitment by being always reliable and trying my best when working. I always completed work on time, talked to my teammates to get their opinion on different things, for example my designs. This was really easy to demonstrate as my teammates were also my friends and we have worked on multiple things together in the past. This really gave me more incentive to put my 110% into the work. This in turn created a nice and positive environment to work in.

5.2 Hung Truong's Teamwork Retrospect:

5.2.1 Always Reliable

To be reliable in my eyes means that you are always on call and available when needed. The things that I say, I would do within the time frame that is set for me. If not, I lose trust and reliability. This is why I always estimate the time in which I need to complete work and then add a few hours or more time to account for complications. By completing my work to the best of my ability and on time as well as staying on-call 24/7. I showed how I was always reliable. I hope to continue my work ethic and what I do to stay reliable on future teams.

5.2.2 Communicates With Confidence

Through being part of many teams and organizations, I realized that communication is an important aspect. Being able to communicate clearly and concisely saves time and effort as well as improve efficiency within the team. I have communicated with confidence by understanding the idea of "no idea is a bad idea" and trusting my teammates. By trusting them I was able to share ideas that I perhaps was scared of sharing. To encourage teammates to share their ideas I also give words of encouragement or constructive but respectful comments on ideas. I never shoot ideas down but ask them to be put to the side and to be potentially used later on. I hope to continue these communication aspects in my future teams.

5.2.3 Does More Than Asked

As the product owner I had to be creative and think outside the box. I brought on ideas that I had learned in a job placement such as a user survey to understand our potential user/client more. This was one thing that I did which was above and beyond. While assigning tasks and speaking with teammates if there is hesitation I always ask if there is any way I could help. This was my way of doing more than what is asked. Perhaps in the future I could include more planning and research to help teammates get a better understanding of their task.

5.2.4 Adapts Quickly and Easily

Going through feedback and sprint reviews and retrospectives I had to adjust my actions and adapt to the situation. An example of how I have adapted is in terms of the documentation on Jira. Through meetings with the TA I realized that the documentation on Jira wasn't enough and that it could be improved. I quickly hopped into action and did what my best to correct the situation. I have always been able to quickly adapt and change, as I understand it is to accomplish a goal. I hope to keep my open and quick mindset for future teams

5.2.5 Displays Genuine Commitment

The team that we chose were handpicked and genuine friends. Because of this we were able to work and stay committed to each other. Since we were all good friends, shared the same course, and the same workload it was easy for each other to empathize or relate to one another. It created a very positive work environment and helped us stay 110% committed. That being said this isn't replicatibale across any team. But I hope to carry the positive mindset that I had to my future teams.

5.3 Abdulaziz Chalya's Teamwork Retrospect:

5.3.1 Always Reliable

Being reliable means you follow through with your responsibilities consistently. You let the team know in a timely manner if you're unable to do so. To be reliable, I set deadlines for myself and communicate them to the team during our meetings. I make these deadlines appropriate, proportional to the effort required yet accounting for possible blockades. By meeting deadlines consistently with quality work, my team trusted me with more and more responsibilities over time. I hope to continue the work ethic I developed to my future teams.

5.3.2 Communicates With Confidence

From my experience with teamwork, I've come to realize communication is one of the most important aspects of being a great teammate. My ability to communicate with confidence stems from my belief in my own abilities. If I notice something off, I speak up to let my team know. I also trust them to take my thoughts seriously, to see me as an equal. It's important to share your ideas, but also allow others to speak up as well. Our team culture was to take everyone's ideas seriously so that everyone felt comfortable to share their thoughts.

5.3.3 Does More Than Asked

As a developer, my main focus was to make the app as great as possible. Although most of the great ideas were thought up by my teammates, I volunteered to execute them, such as the survey. I ended up doing a large amount of the final report even though my main priority was supposed to be the app. The reason for that is because I realized the team needed more help, so I stepped up to the task. In the future, I hope we gain a better understanding of the work ahead of us, so no one has to put new work on their plate midway through the project.

5.3.4 Adapts Quickly and Easily

As the end of the 3rd sprint approached, I came to realize my teammates needed more hands for the final reports. I had to switch around my study schedule to make time for the reports. There were also technical issues when making the demo video that I had to fix last-minute. These events proved to my team I can adapt quickly and easily.

5.3.5 Displays Genuine Commitment

I trusted my team to be committed to the project because of my experience of working with them in the past. I also proved my own commitment to the team by working on the app and reporting for long hours. Since all of my teammates are in the same program as me, our other deadlines are the same as well, thus our commitment was equal and fair. I hope to continue this mindset to any future team projects.

5.4 Hashim Abu Sharkh's Teamwork Retrospect:

5.4.1 Always Reliable

I was always reliable day in and out as I always got the job done, kept my word and provided consistent quality work. Keeping my word is a core value of mine. I met my deadlines and whenever I was asked something I would do it quickly with confidence and would prioritize the project. If someone were to ask me about a task I completed I would also communicate it with confidence and teach it properly.

5.4.2 Communicates With Confidence

I communicated my ideas throughout the whole project by suggesting meaningful things that helped our app. I suggested a custom recommendation machine learning model for the vape triggers entered by the users. Generally, we had a good team environment which allowed for sharing ideas easily without any fear. In future team projects I would like to speak up more and communicate more efficiently and concisely to make the lives of my teammates easier.

5.4.3 Does More Than Asked

I would always go above and beyond. An example of that is that I created the machine learning model for our group, something that was not asked by me. I gathered the dataset from our survey takers, did my research on machine learning models and statistics and created it. I then connected a python backend with little documentation on it. However, after lots of research I could not figure out how to package the frontend and backend without a cloud server or our own server.

5.4.4 Adapts Quickly and Easily

First, using Flutter was something I needed to get adapted to quickly as it had scarce documentation. I had to watch a lot of videos in my free time to grasp the core idea behind flutter and flutter's backend. And then Flutter suddenly clicked and all the 'puzzle' pieces were connected in my brain.
I also had to adapt with my new teammates, as I never worked with them before. It was a fun experience and totally was worth it.
Moreover, at the end of sprint 2, we had a bug the day before the demo. I stayed calm, relaxed and woke up early in the morning knowing that I will be able to fix it. And my confidence allowed me to fix it and we were complimented by the TA's that day which made me happy.

5.4.5 Displays Genuine Commitment

Displaying genuine commitment was a priority for me since day 1. I wanted to have a good team environment and wanted to forge strong connections with my team members.

5.5 Spencer Vecile's Teamwork Retrospect:

5.5.1 Always Reliable

I demonstrate this by always getting the work done that I say I will do on time and to a high-quality standard. I also try to be available at all times of the day and night, but I wasn't always. To improve for next time, I will try to monitor group chats more and reply faster.

5.5.2 Communicates With Confidence

If I have an idea I always speak up and if it is not understood I take the time to explain why it is the right thing to do. I actively participate in all group discussions and always speak my mind. I also listen to other opinions and try to understand and accept different viewpoints. Sometimes I get stuck with an idea that I can't let go of and I will try to be better at just letting it go next time.

5.5.3 Does More Than Asked

I generally like a challenge so I will take on more complex parts of the workload even though I will probably end up pulling my hair out over it. If I have a good idea and it's going to be difficult I generally don't shy away from it even if there is an easier option. For next time I think sometimes it's better to keep

things simpler especially under tight time constraints. I sometimes found myself spending too much time on a problem when there were other things that needed to be done.

5.5.4 Adapts Quickly and Easily

During the project our team members generally started to specialize in different parts of app development and if someone was busy and didn't have time to help, I tried my best to learn that different part and solve problems myself. I didn't put as much effort into project planning on Jira as I probably should have next time, I will try to put more effort into that.

5.5.5 Displays Genuine Commitment

I displayed commitment to this project by doing very long 12h+ days whenever needed or staying up into the night to meet deadlines. I also enjoyed creating relationships with group members and giving and receiving help from them. Sometimes after several long days in a row I was ok with just accepting what we had rather than what I wanted features to look like so next time I will try to space the work more evenly so I don't burn out.