Fitness Friend

A Modern Approach to Fitness

Group 6

Prepared By: Amber Haynes, Tiyon King, Jenna Krause, Mya Odrick,
Devvrat Patel, Andrew Rezk, Maria Rios, Shivani Sunil, Hedaya Walter

May 7, 2020 Software Engineering

Table of Contents

Summary of Changes **5**

Customer Statement of Requirements **6**

Glossary of Terms **8**

System Requirements **9**

Enumerated Functional Requirements 9

Enumerated Non-Functional Requirements 10

User Interface Requirements 10

Functional Requirements Specification **11**

Stakeholders 11

Actors and Goals 11

Use Cases 12

System Sequence Diagrams 14

Effort Estimation **17**

Domain Analysis **17**

Domain Model 17

Concept Definitions 19

Association Definitions 19

Attribute Definitions 20

Traceability Matrix 21

System Operation Contracts 22

Mathematical Models 23

Interaction Diagrams **24**

Diagrams 24

Design Patterns 28

Class Diagram and Interface Specification **29**

Class Diagram 29

Data Types and Operation Signatures 30

Traceability Matrix 32

Design Patterns 33

Object Constraint Language Contract Specification 33

Calorie Tracker: 33

Search Foods 33

System Architecture and System Design **34**
Architecture Styles 34
Identifying Subsystems 35
Mapping Subsystems to Hardware 35
Persistent Data Storage 35
Network Protocol 36
Global Control Flow 36
Hardware Requirements 37
Algorithms and Data Structures **37**
User Interface Design and Implementation **38**
Design of Tests **40**
History of Work, Current Status, and Future Work **43**
References **44**
Individual Breakdown **46**

Summary of Changes

- **Customer Statement of Requirements**
  We have removed an initial planned feature, the SOS function, because we have recognized that many smartwatches already have this feature implemented in it.
- **Glossary of Terms**
  We have added a new term, ‘sync,’ to be defined. We have also removed the term ‘alert’ to reflect the removal of the SOS feature. ‘Reminders’ was changed to ‘notifications.’
- **System Requirements**
  We have completely redid this section from scratch.
- **Functional Requirements Specification**
  We have modified ‘Stakeholders,’ as well as ‘Actors and Goals,’ to reflect who will be interacting with it. The format of the ‘Actors and Goals’ section was switched into a tabular format to make it easier to read. We created additional use cases for creating an account, logging in, and logging out as well as system sequence diagrams.
- **Domain Analysis**
  We have removed the SOS feature to reflect the merging of the sub-groups. We have also added new cases to the traceability matrix to reflect the new added use cases.
- **Interaction Diagrams**
  We have replaced Use Case #7, the Google Search Engine with the Edamam API, because the resources allocated with the Edamam API better align with our system. We have also created new use cases for creating an account and logging in. We have revised the Interaction Diagrams to incorporate the Publisher-Subscriber pattern.
- **Class Diagram and Interface Specification**
  We have adjusted the format of ‘Data Types and Operation Signatures’ into a table as well as updated the class diagram to reflect the removal of the ‘Exercise Selector’.
- **System Architecture and System Diagram**
  We have removed the ‘Exercise Selector’ sub-folder as one of the features on the ‘Feature Selection Page’ and replaced it with the ‘Settings’ page to match our updated UI.
- **Algorithms and Data Structures**
  We implemented the Edamam API for the calorie tracker feature.
- **User Interface Design and Implementation**
  We have added explanations and descriptions for the select page, login page, sign-up page, homepage, and settings page.

Customer Statement of Requirements

As a consumer, it becomes very difficult to find time throughout the day to work out. Whether working or in school, incorporating workouts into an already busy schedule becomes stressful and discouraging. In addition to finding time, it also becomes hard to stay motivated to stick to exercising during the time I've devoted to it. Using the calendar and reminder features on the device, the overwhelming and discouraging aspect of finding the times to dedicate to exercise are taken out of the equation. This relief combined with the additional feature of motivational reminders for the time I had already dedicated to working out took away major factors that previously prevented me from sticking to the plan.

For someone who uses their Google calendar to keep track of events and their schedule, being able to easily incorporate my already created calendar into a new device makes the experience seamless and much less stressful. The only factors that I will have to worry about are selecting the range of time in which the reminder feature will look for free time day-to-day and the amount of time needed in order to exercise, making the set-up experience quick and painless. Picking the amount of free time needed in order to exercise will give me the flexibility for whether I work out at home or if I go to the gym to work out. Once the range of times and amount of time needed to exercise are set, I will receive motivational reminders or notifications informing me of when I have enough free time to incorporate a workout into my schedule.

These features will allow me to be better motivated in order to implement workouts into my schedule on a greater basis. As most know, constant notifications can become quite annoying, usually prompting me to turn the reminders off completely. When the reminders occur, I will have the option to snooze it - like an alarm - for a later reminder, check-in and say that I am working out, indicate that I am at an event that was not inputted, or indicate that I do not wish to workout that day. The ability to snooze or dismiss the reminders and notifications with a wide range of options will lower the chance that I will disable the feature completely.

As a user, I need a way to keep track of my intake of calories and keep track of how many calories I've burned. I want to be able to see my progress and check how much I should be eating based on the exercises I complete. I can input my details into the app at the beginning of my fitness journey such as my current weight, height, age. I can let the app know my ultimate goal and the app can suggest different foods I can eat based on other users that have similar details and have had success. The calorie tracker feature will incorporate a search bar to make it easier for me to search up the foods that I am eating

and will give me the number of calories each food contains. The calorie tracker feature can then use this information to calculate how many total calories I've consumed.

In addition, it will be difficult for me to know which exercises are better suited for me and how effective it will be. I will rely on the app to give me different exercise suggestions based on those that have worked for other users with similar profiles. I can choose which ones to follow and put that information into the app. For example, if I want to gain muscle, I will receive food recommendations that will help me achieve this. Likewise, if I want to lose fat, I will be given the appropriate recommendations. These recommendations will save me time and make it easier for me to make the right choices.

Because I have a busy schedule and can be forgetful at times, I will depend on the app to send me a reminder at the start of the day to let me know what type of exercises I should complete. I can adjust the frequency of the reminders so I could choose to have the app remind me at the meal times that I input into the system or just have it remind me at the start of the day. At the end of each day, the app will let me know the progress I've made that day. I will receive a notification of what I've burned off and what I have consumed. Seeing the progress I've made will motivate me to keep working hard towards my goals

This app provides various playlists for each emotion I feel. It took my preference then created playlists based on songs I'd typically like. For instance, it created a playlist solely based on different types of workouts I have. For a hard, intensive day, the app created playlists for me that were a list of songs that are generally fast paced and upbeat. For a slower day of workout, the app created songs that were not as fast paced but generally motivational nonetheless. The variance in the music displayed the various levels of intensity in my workout. This app took into account of my emotions during physical activity and incorporated my specific music tastes and provided me with an enhanced playlist of music.

Glossary of Terms

Calorie Deficit is when the amount of calories consumed is less than the amount needed to maintain current body weight. If the user's goal is to lose weight, we recommend a caloric deficit diet and workout through low-calorie meals and calorie-burning exercises.

Calorie Surplus is when the amount of calories consumed is greater than the amount needed to maintain current body weight. If the user's goal is to gain weight or muscle, we will recommend a caloric surplus diet and workout through healthy high-calorie food items and muscle gaining exercises.

The **Favorites List** is a list where the user can add specific meals and songs, allowing them to access those items quicker. Upon clicking on one of their favorite items, they will receive information like the number of calories in the meal, or the best exercises to perform while listening to their songs.

Freetime is a block of time where the user doesn't have anything planned on their Google Calendar. The user manually enters the interval for which the app should be looking for free time and the app will suggest the user to exercise during that time.

Notifications are sent to the user - outside of the app - to communicate when they have the free time to exercise. They are also sent to remind them to input their daily calorie intake and incorporate certain foods and exercises to meet their goals. These notifications are able to be turned on or off, manually, through the app or the cellular device.

Syncing the app with Google means that the user authenticates the Fitness Friend app to pull information directly from its calendar to see when there is free time, as well as input exercise times. This is done by the user logging into Google through the app.

Tempo refers to the speed or pace of the music. The app looks at the user's desired workout and recommends music based on tempo and the user's preferred music genre.

System Requirements

Enumerated Functional Requirements

Table #1: Enumerated Functional Requirements

ID	Priority Weight	Requirement
REQ-1	3	As an authorized user, I will be able to take pictures of my food to input calorie intake.
REQ-2	2	As an authorized user, I will be able to enter my fitness goals.
REQ-3	5	The system will suggest meals and exercise based on the goals of the user.
REQ-4	7	The system will use the user's heart rate to calculate the number of calories burned during a workout.
REQ-5	3	The system will display the number of calories burned and the number of calories consumed.
REQ-6	7	As an authorized user, I will be able to search for my meals through the search bar to input my calorie intake.
REQ-7	3	As an authorized user, I will be able to save frequent or favorite meals/snacks for a faster and easier experience.
REQ-8	6	As an authorized user, I will be able to sync my Google calendar with the application.
REQ-9	3	As an authorized user, I will be able to snooze workout reminders for a later time.
REQ-10	5	As an authorized user, I will be able to toggle workout reminders on or off.
REQ-11	3	As an authorized user, I will be able to 'check-in' with the app to inform that I am sticking to my scheduled workouts.
REQ-12	9	The application will suggest times the user should workout, based on the user's free time.
REQ-13	9	As an authorized user, I will be able to pick a type of workout I would like to do and the app should provide me with playlists with BPMs suited for the workout
REQ-14	8	My playlist options should also incorporate the artists I prefer
REQ-15	9	I should be able to set a workout goal in terms of calories or steps and upon achievement of those goals, I should be able to activate a free membership reward.

REQ-16	3	I should be able to save the playlists I prefer and shuffle the same next time.
REQ-17	4	I should be able to rely on the app to generate new playlists for me, as well.
REQ-18	2	The music tempo should be correlated to my heartbeat while working out

Enumerated Non-Functional Requirements

Table #2: Enumerated Functional Requirements

ID	Priority Weight	Requirement
REQ-19	7	The system should work without maintenance for a long period of time.
REQ-20	7	The system should be easy to fix if it ever breaks.
REQ-21	5	The system should allow multiple users at the same time and it should not crash.
REQ-22	6	The system should not lag to the user's operations.
REQ-23	4	The system should have high security such that only the authorized user is allowed access to their data.
REQ-24	5	The system should remain functioning in the event of an update.

User Interface Requirements

Table #3: User Interface Requirements

Pre -Content Presentation/Welcome page REQ-25: Displays the login/sign up page of the app, allowing the user to create a personalized profile meant for easy navigation of the app's features
Content Presentation-Features Display REQ-26: Displays the 4 features of the app and provides the user to choose any one at a given instant
Easy Navigation REQ-27: Same as REQ 26, provides the user with a simple, concise and clear display
Consistent UI Elements

REQ-28: The app as a whole and each of the sub features follow a consistent theme and design with respect to color coordination as well as menu displays.
Feedback Mechanism REQ 29: The menu allows the user to switch based on their responses. If the user wants to sync their google calendar to the app, the next step will be to login to their gmail account.
Default Settings REQ 30: This app comes with a default interface consistent throughout irrespective of Android versions or screen sizes.
User Centric Approach REQ-31: The app focuses on maximizing user satisfaction by providing services and products.

Functional Requirements Specification

Stakeholders

Identify anyone and everyone who has an interest in this system (users, managers, sponsors, etc.). Stakeholders should be humans or human organizations.

- Users
- Nutritionists
- Music Services
- Google Calendar
- Firebase Database

Actors and Goals

Table #4: Actors and Goals

Actor Name	Type	Role	Goal (if applicable)
Users	Initiating	Implement this technology into their lives	To maintain or achieve their fitness goals
Google Calendar	Participating	Provide access to previously created Google calendars	
Nutritionists	Participating	Gives input on what foods and exercises to recommend to users	

Music Services	Participating	Provide access to music streaming services	
Firebase Database	Participating	Provide a real-time database that will be used to store user information	

Use Cases

Table #5: Use Cases

Actor	Actor's Goal	Functional Requirements	Use-Case Name
System	Remind users during pre-set period if they have enough free time to exercise	Provides users with time throughout the day that they can implement a workout	Free Time Notifications (UC#1)
User	Users will be able to set the minimal amount of free time needed to exercise as well as the time interval in which free time reminders are enabled.	Allows the user to customize the free time interval and amount of time required for a work out.	Set Free Time (UC#2)
System	If the user indicated that they exercised or if they dismissed the free time notification free time notifications will be disabled until the start period of the following day.	Turn off free time or work out notifications until the free time interval of the next day.	Dismiss Reminders (UC#3)
User	Users will be given the option to enable or disable free time notifications.	Toggle free time notifications ON/OFF.	Toggle Notifications (UC#4)
User	Users will be able to acknowledge, snooze, or accept the work out reminders.	Encourages the user to stick to their goal of exercising.	Acknowledge/ Snooze Reminders (UC#5)
System	Remind the user to exercise, again, after a given period	Waits 30 minutes before checking if the amount of free	

	of time.	time is enough to implement a work out.	
User	Users will be able to sync their Google Calendars to the calendar on their device.	Allows users to seamlessly add their already created calendars to their device.	Calendar Sync (UC#6)
User	Users will be able to use a search bar to look up the calories in their food.	Lets users easily add the calories eaten into the system.	Calorie Search (UC #7)
User	Users will select what exercises they have completed for the day.	System will receive that information and know how many calories were burned based on pre-assigned values.	Exercise Selector (UC #8)
User	Users will be able to select a future workout based on the number of calories they will potentially burn	System will determine the number of calories burned based on the user's past exercise experiences	
System	Keeps track of the calories consumed and burned.	Notifies users of the calories lost or gained at the end of each day.	Calorie Notifier (UC #9)
User	Users will be able to create accounts with their first name, last name, email, and password of choice.	System will store this information in the database.	Create Account (UC #10)
User	Users will be able to interact with recommendations suggested by nutritionists	The system will have a database of healthy recommendations suggested by professionals.	Professional Recommendations (UC #11)
User	Users will be able to save frequent/favorite meals for a quicker and easier meal-tracking experience.	App will display the list of favorite meals under the meal search bar.	Favorite Meals (UC #12)
User	Users will be able to select a certain type of workout and the app will generate BPM based playlists based on the intensity of the workout	App will display the selected playlists and the user will have the option to save them for future workouts	App generated BPM playlists (UC #13)

System and User	On the 1st of every month, the user will be prompted to enter a workout goal, either in terms of calories or steps.	The app will then reward the user, in terms of subscriptions of one month	Workout rewards (UC #14)
User	Users will be able to integrate spotify/apple music from their phone	The app will also generate playlists based on the artists the user likes/prefers and will allow the user to save them	User preferred playlists (UC #15)
System	The User will receive daily revised playlists from the app, to keep them motivated as listening to same music overtime gets monotonous	The app will generate new playlist notifications for the user, and perhaps in conjunction with the workout reminders to increase the motivation factor.	Playlist notifications (UC #16)
System	The user will experience a change in music tempo in conjunction with their heart rate.	The app will change songs/ music tempo by detecting the user's heart rate, high BPM songs will play	Heart rate and BPM sync (UC #17)
User	Users will be able to log into their accounts and use Fitness Friend.	System will log the user in through email authentication.	Log In (UC# 18)
User	Users will be able to log out of their account.	System will log the user out.	Log Out (UC# 19)

System Sequence Diagrams

Diagram #1: Acknowledge/Snooze Reminders

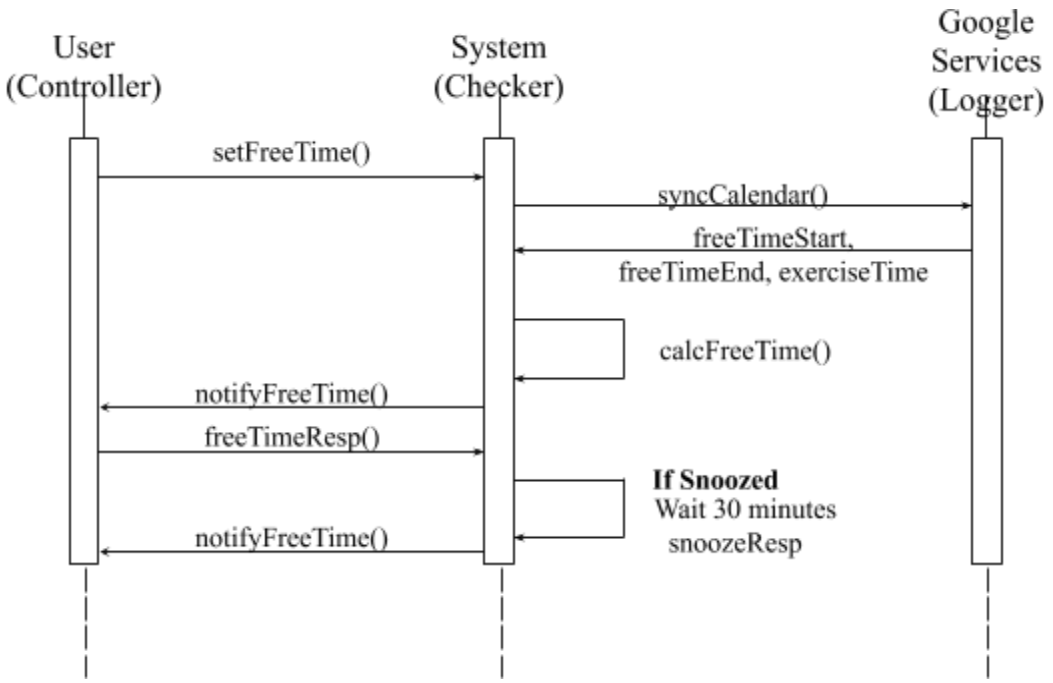


Diagram #2: Calorie Search

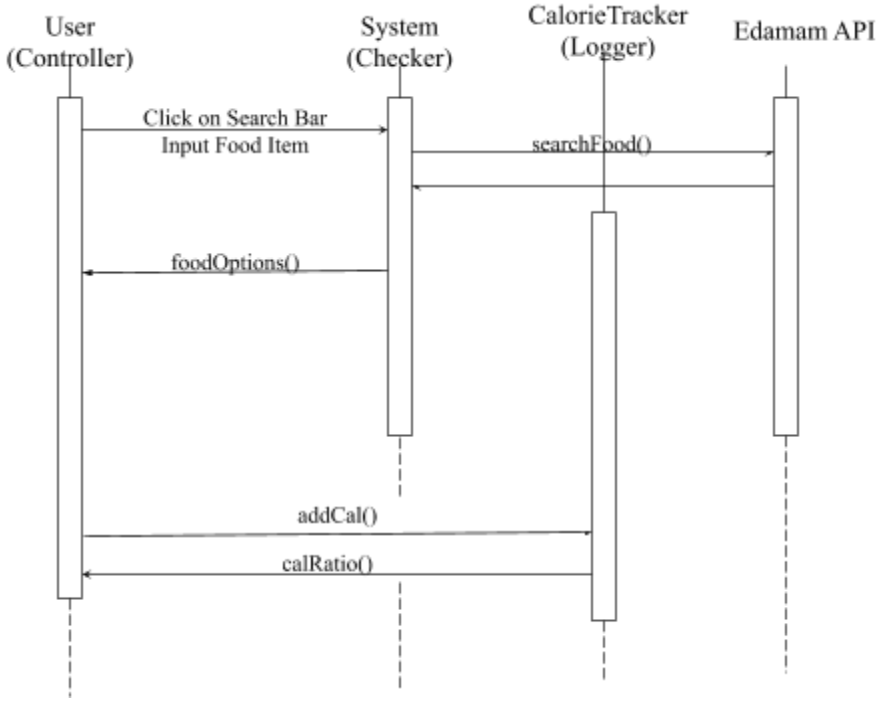


Diagram #3: App Generated BPM Playlists

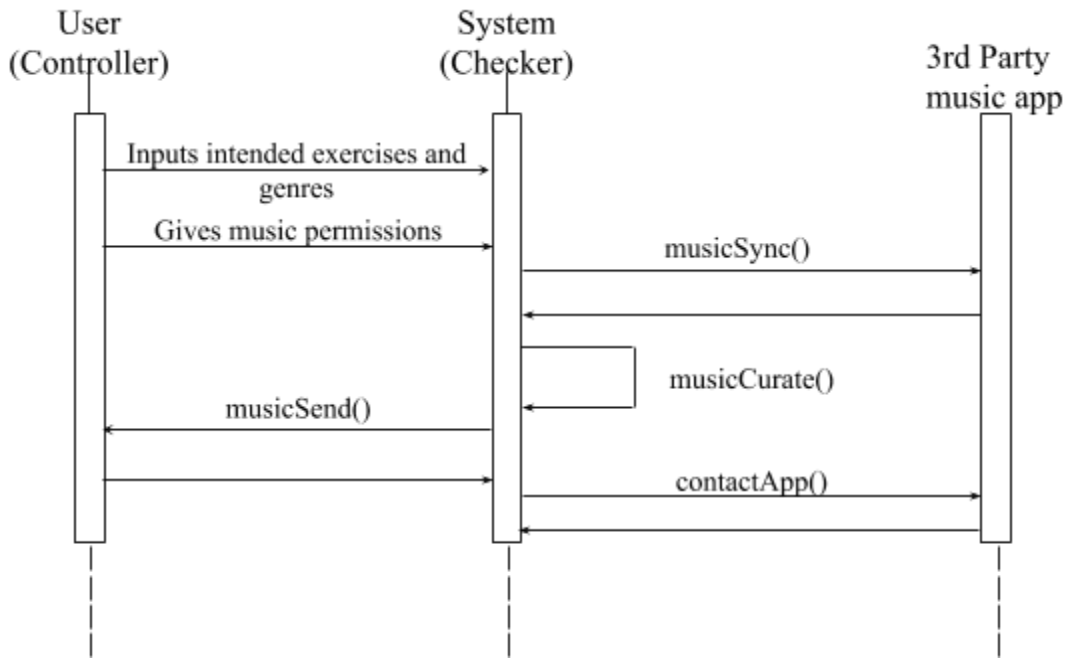


Diagram #4: User Login

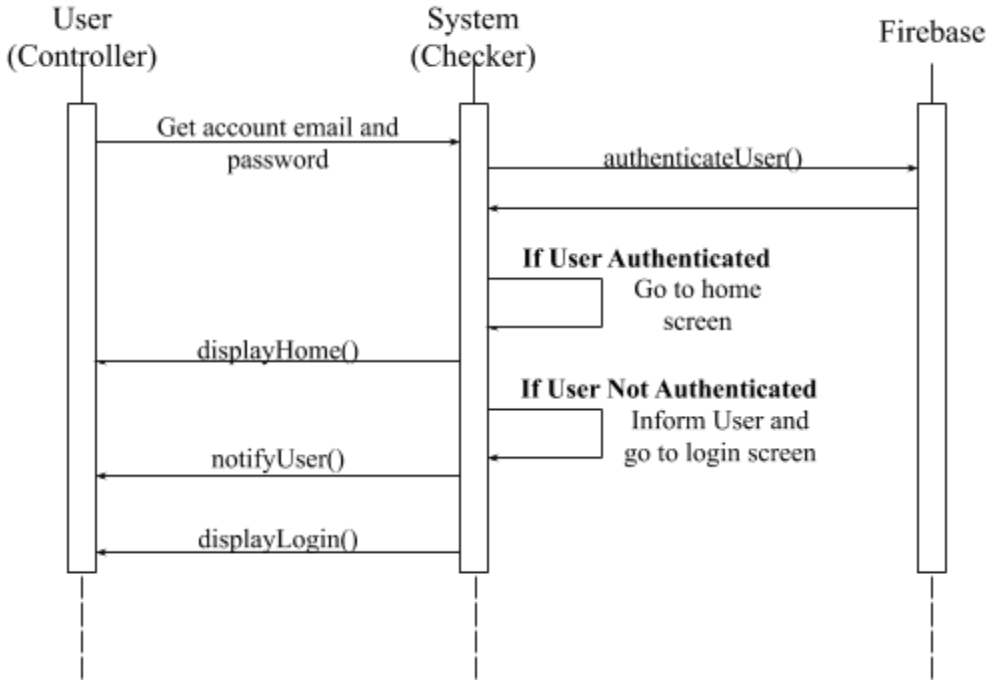
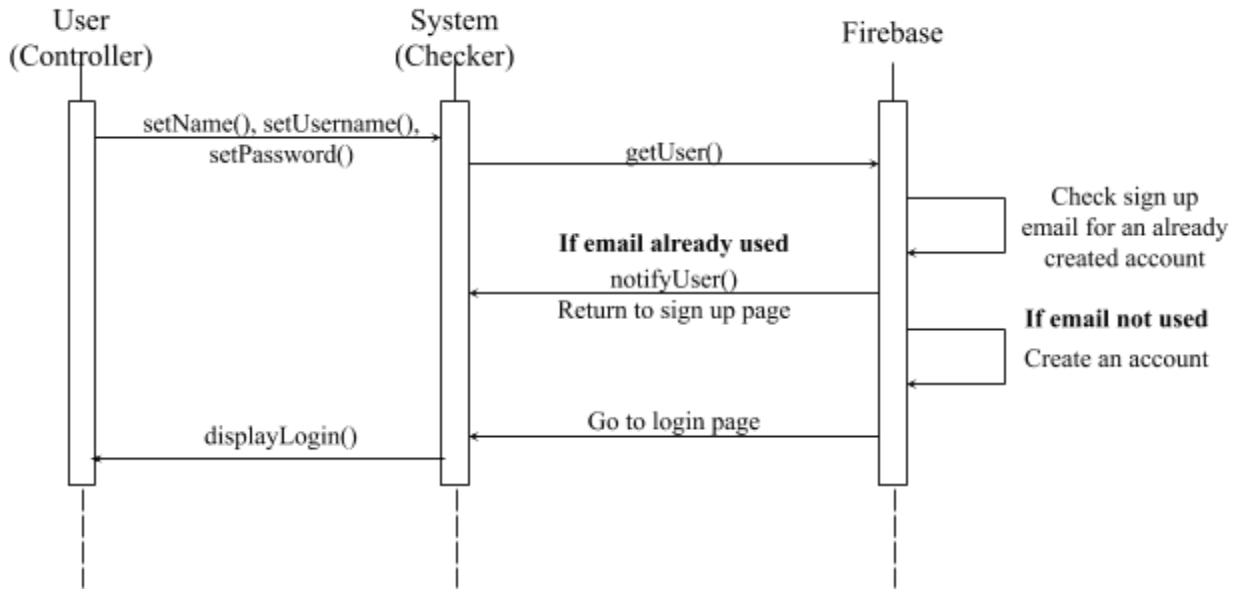


Diagram #5: Create an Account



Effort Estimation

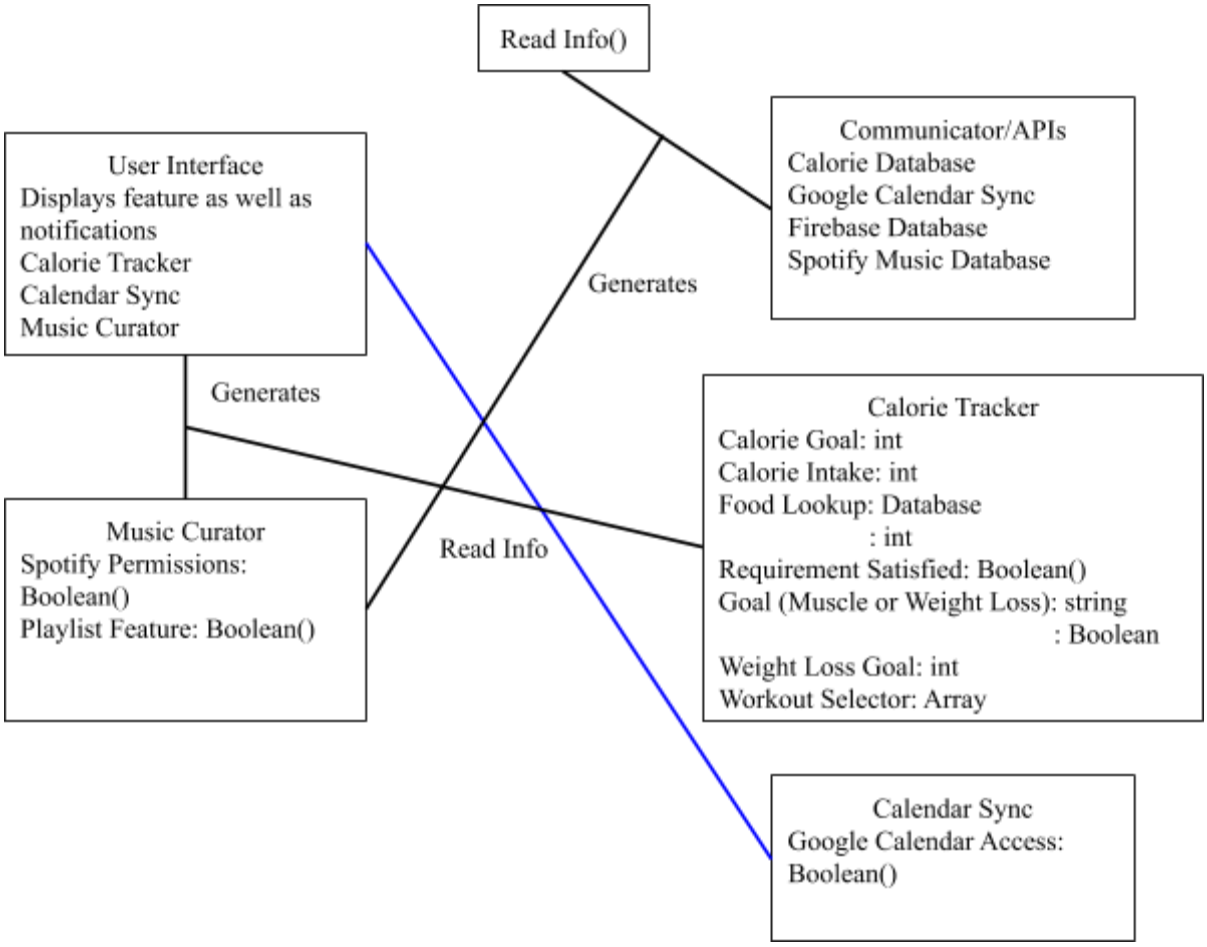
UCP = User Case Points
PF = Productivity Factor

Assume PF = 28 hours per use case point. Previously in Report 1, we found that the UUCP = 192, TCF = 0.935,and ECF = 0.995 for the overall project. Then, we found UCP = UUCP x TCF X ECF = 192* 0.935*0.995 = **178.62.**

Duration = UCP x PF = 178.62 x 28 = **5,001.36 hours**

Domain Analysis

Domain Model



Below are our domain models from the first report.

Diagram #6: Second Report Domain Model

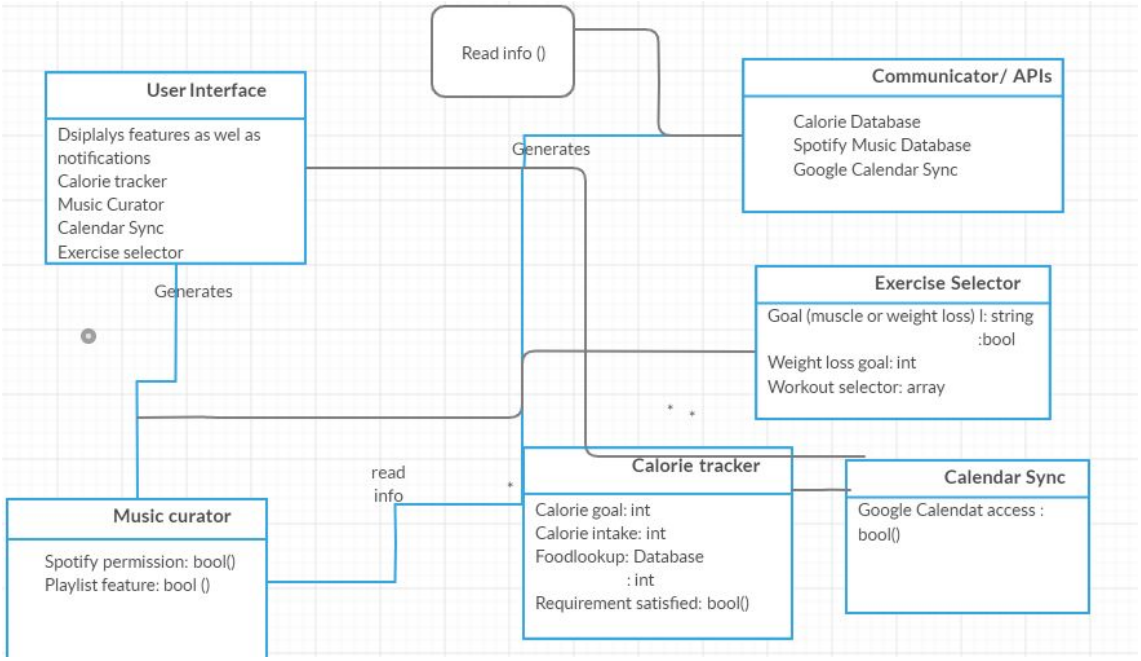
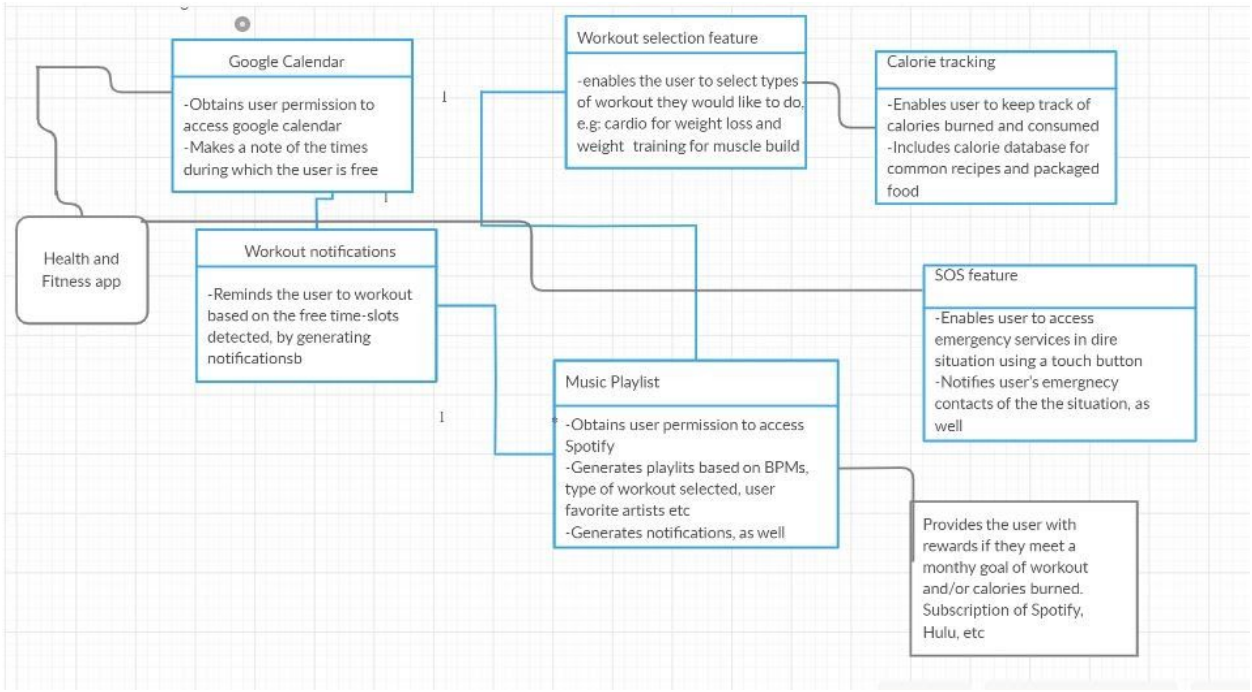


Diagram #7: First Report Domain Model



Concept Definitions

Table #6: Concept Definitions

Responsibility	Concept
R1: Set the users free time interval and workout time	Calendar
R2: Calculate and keep track of users free time	Calendar
R3: Send notifications to work out during enough free time	Calendar
R4: Snooze/Dismiss/Acknowledge free time notifications	Calendar
R5: Enable/Disable free time reminders	Calendar
R6: Gather data from Google Calendar	Calendar
R7: Calculate user's calorie intake/outtake ratio	Calorie Tracker
R8: Display the amount of calories consumed/burned	Calorie Tracker
R9: Recommend exercises/ meals to meet goal	Calorie Tracker
R10: Compile list of most frequent exercises/meals	Calorie Tracker
R11: Prompts user for 3rd-party music subscription login and permissions	Music Playlist
R12: Accesses user-intended workout when user prompts creation of playlist	Music Playlist
R13: Matches songs to exercise intensity and BPM to compile into playlist	Music Playlist
R14: Saves playlist under unique name for later use	Music Playlist

Association Definitions

Table #7: Association Definitions

Concept Pair	Association Description	Association Name
Calendar ↔ SyncCalendar	Calendar calls SyncCalendar to sync the users existing Google Calendar	Get Request Send Response
Calendar ↔ SetFreeTime	Calendar calls SetFreeTime when user sets free time interval and time needed	Set Request
Calendar ↔ CalcFreeTime	Calendar calls CalcFreeTime to check if there is enough free time to exercise	Get Request Send Response

Calendar ↔ ToggleFreeTime	Calendar calls ToggleFreeTime to enable/disable free time notifications	Set Request
Calendar ↔ FreeTimeResp	Calendar calls FreeTimeResp to store user's response to free time notification	Get Request Send Response
Calendar ↔ NotifyFreeTime	Calendar sends a notification to the user when there is free time to exercise	Send Response
Calorie Tracker ↔ CalRatio	Calorie Tracker invokes CalRatio to get ratio of calorie intake to outtake	Send Result
Calorie Tracker ↔ FoodRec	FoodRec uses Calorie Tracker, and user preferences, to recommend meals that will help the user meet their goals	Get Request Send Response
Calorie Tracker ↔ ExerciseRec	ExerciseRec uses Calorie Tracker to recommend exercises that helps user burn enough calories to meet their goals	Get Request Send Response
Calorie Tracker ↔ FreqList	FreqList compiles a list of the user's most frequent exercises and meals	Get Request Send Response
CalDisp ↔ CalRatio	CalDisp uses information from CalRatio to display the user's calorie consumption and calories burned	Send Result
Music Playlist ↔ MusicSync	MusicSync prompts user to input their 3rd party music subscription info and give the Spotify app permissions	Get Request Set Request Send Result
Music Playlist ↔ WorkChoose ↔ ExerSave	WorkChoose displays list of saved workouts and common genres for user to choose for playlist curation, and saves in temporary secondary memory	Get Request Set Request
Music Playlist ↔ MusicCurate	MusicCurate matches songs based on workout chosen and genre preferences	Get Request Send Result
Music Playlist ↔ MusicSend	MusicStore stores playlist and notifies the user of playlist creation	Send Result

Attribute Definitions

Table #8: Attribute Definitions

Responsibility	**Attribute**	**Concept**

R1: Set the users free time interval and workout time	SetFreeTime	Calendar
R2: Calculate and keep track of users free time	CalcFreeTime	Calendar
R3: Send notifications to work out during enough free time	NotifyFreeTime	Calendar
R4: Snooze/Dismiss/Acknowledge free time notifications	FreeTimeResp	Calendar
R5: Enable/Disable free time reminders	ToggleFreeTime	Calendar
R6: Gather data from Google Calendar	SyncCalendar	Calendar
R7: Calculate user's calorie intake/outtake ratio	CalRatio	Calorie Tracker
R9: Recommend exercises/ meals to meet goal	CalDisp	Calorie Tracker
R9: Recommend exercises/ meals to meet goal	FoodRec & ExerciseRec	Calorie Tracker
R10: Compile most frequent exercise/meal list	FreqList	Calorie Tracker
R11: Display list of common exercises for selection and additional options for each exercise post-selection	ExerSelect	Workout Selector
R12: Compiles and saves workouts under user-chosen name	ExerSave	Workout Selector
R13: Prompts for 3rd-party subscription login and permissions	MusicSync	Music Playlist
R14: Accesses workout when user prompts creation of playlist	WorkChoose	Music Playlist
R15: Matches songs to exercise intensity to compile to playlist	MusicCurate	Music Playlist
R16: Saves playlist under unique name for later use	MusicSend	Music Playlist

Traceability Matrix

Table #9: General Traceability Matrix

Domain Model	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6	UC 7	UC 8	UC 9	UC 10	UC 11	UC 12	UC 13	UC 14	UC 15	UC 16	UC 17	UC 18	UC 19
Calendar Sync	x	x	x	x	x	x													
Calorie Tracker							x		x		x	x		x					
Music Playlist								x					x		x	x	x		

System Operation Contracts

Contract CO1:

Operation:	Exercise Reminder/Indicator
Cross Reference:	User Case UC- 5 : Acknowledge/Snooze Reminders
Pre-Conditions:	User has inputted the time intervals in which they have free time
Post-Conditions:	If acknowledged, reminders are stopped until the next free time interval. If snoozed, another reminder is sent after 30 minutes - if there's still enough time available for exercise.

Contract CO2:

Operation:	Search Calories Consumed
Cross Reference:	User Case UC - 7: Calorie Search
Pre-Conditions:	User finished a meal and wants to know how many calories to input
Post-Conditions:	Calorie tracker records the number of calories taken in.

Contract CO3:

Operation:	Create Personalized Music Playlist
Cross Reference:	User Case UC- 13: App-Generated BPM Playlists
Pre-Conditions:	User requests playlist creation and selects saved workout and genre
Post-Conditions:	Music Playlist feature curates playlist, saves it, and notifies the user

Mathematical Models

Table #10: Calories Burned Calculator Reference

Activity Level	Exercise Type	Calories Burned Per Hour	
		Male	Female
Moderate	Walking (3.5 mph)	460	370
	Cycling (5.5 mph)	460	370
	Dancing	460	370
Strenuous	Swimming	730	580
Very Strenuous	Running (7 min/mile)	920	740

Calories Burned Calculator Based on User Input (Cal):

Male User:

$$\text{Calories burned} = \frac{weight}{175} * \frac{Calories\ Burned\ Per\ Hour}{60} * duration\ of\ exercise\ in\ minutes$$

Female User:

$$\text{Calories burned} = \frac{weight}{140} * \frac{Calories\ Burned\ Per\ Hour}{60} * duration\ of\ exercise\ in\ minutes$$

Calories Burned Calculator Based on User Heart Rate (kCal):

Male User:

$$\text{Calories burned} = [(age * 0.2017) + (weight * 2.205 * 0.1988) + (heart\ beats\ per\ minute\ * 0.6309) - 55.0969] * \frac{duration\ in\ minutes}{4.184}$$

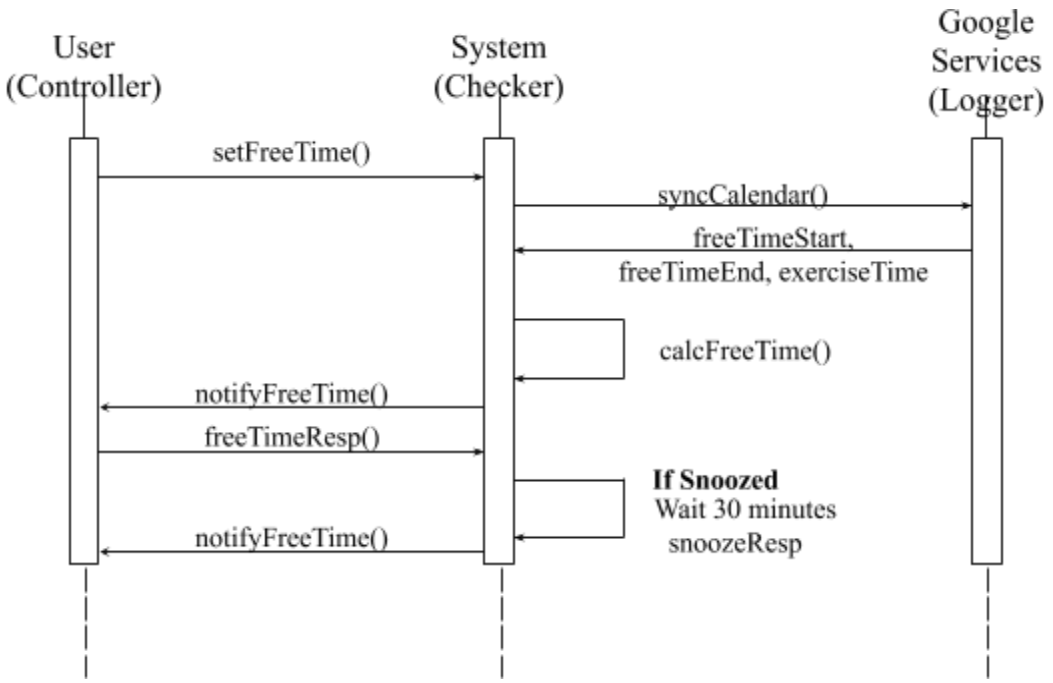
Female User:

$$\text{Calories burned} = [(age * 0.074) + (weight * 2.205 * 0.1263) + (heart\ beats\ per\ minute\ * 0.4472) - 20.4022] * \frac{duration\ in\ minutes}{4.184}$$

Interaction Diagrams

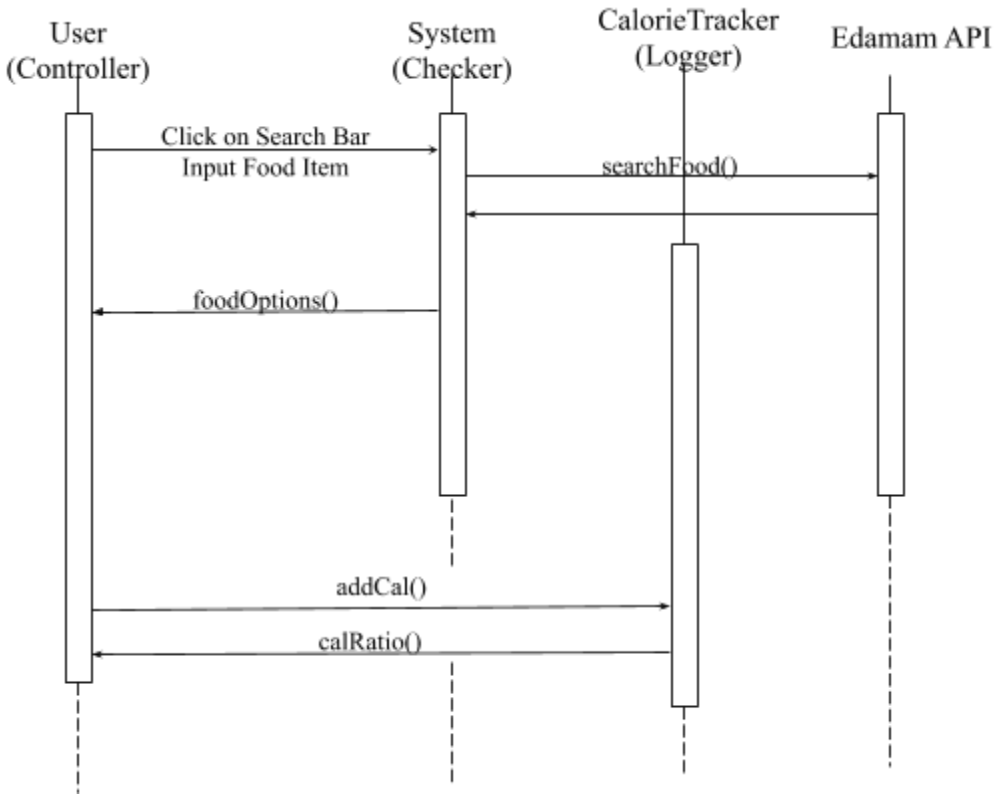
Diagrams

Diagram #8: Acknowledge/Snooze Reminders



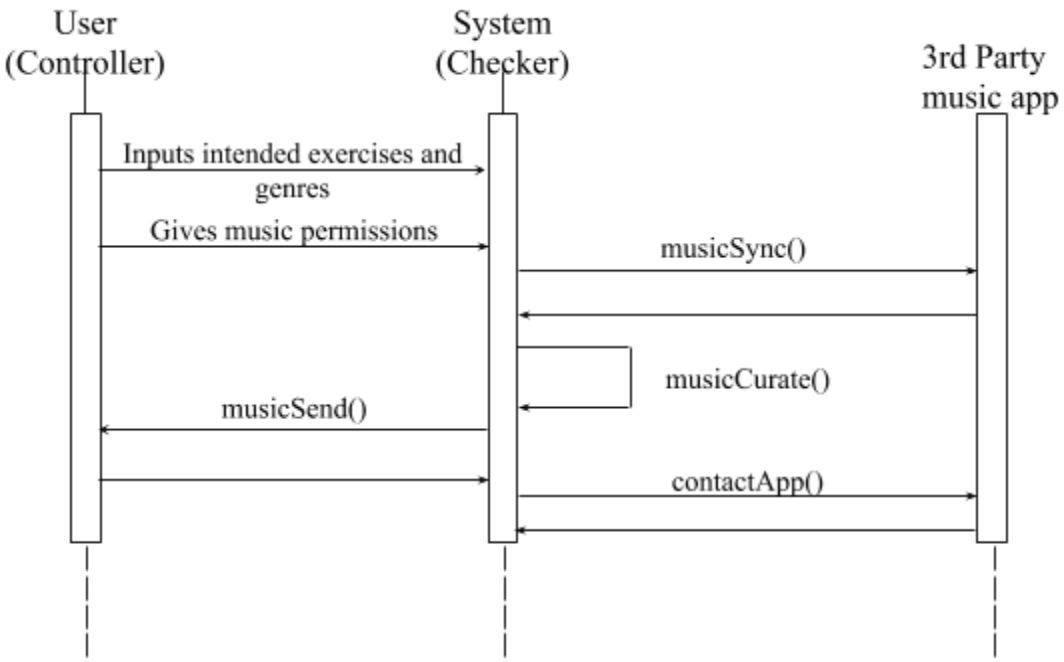
Description of Use Case #5:
This sequence is used to notify the "user" of free time available to exercise. Once the "user" inputs the required amount of free time necessary to work out and the interval of time in which free time notifications will be enabled, the "system" sends a request to "Google Services" for the user's calendar information. Then, the "system" checks for free time that meets the two requirements. If there is enough free time within the free time notification interval, the device sends a notification to the "user" informing them of this availability. The "system" waits for a response from the "user" and acts accordingly. The Interface Segregation Principle is used to understand each class in isolation.

Diagram #9: Calorie Search



Description of Use Case #7:
This use case shows the sequence of the search function within the Calorie Tracker. It uses the Single Responsibility Principle (SRP). The search engine relays information about the requested food, the calorie tracker is used to keep an updated calorie counter, and the controller serves as the main unit for the user's interaction. It also uses the Interface Segregation Principle (ISP), where each class can be understood in isolation. The calorie tracker requests the number of calories consumed and calculates the caloric ratio with the number of calories burned. The search engine class on its own performs the task of generating calorie information upon receiving serving size information and food names as keywords. The independent functions are combined for this use case.

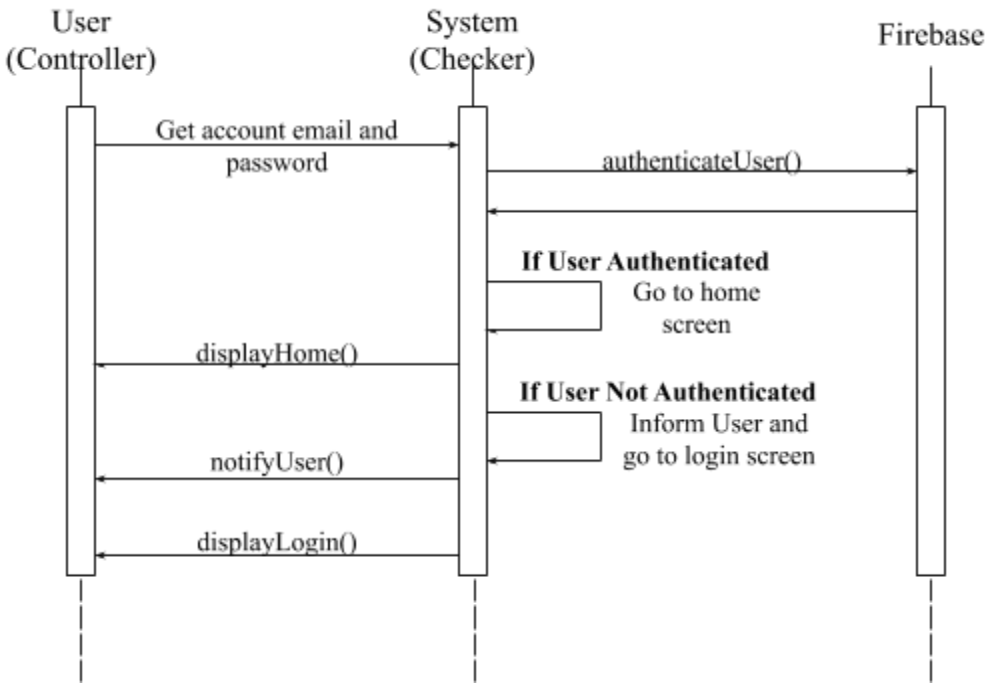
Diagram #10: App Generated BPM Playlists



Description of Use Case #13:

This use case shows the process of the creation of the curated music playlists. Once the "user" inputs the exercises, preferred music genres, and allows the app to have access to its music permissions, the "system" is able to contact the "3rd Party Music App". After the "3rd Party Music App" responds with the information, the "system" creates a curated playlist that meets the "user's" inputted needs. Once the music playlist has been created, the "system" notifies the "user" indicating that their custom playlist is complete to be played. If the "user" indicated they wish to play songs, the "system" contacts the "3rd Party Music App" to retrieve requested music in order to play the song. In this case, the Interface Segregation Principle is used, being able to understand each class in isolation.

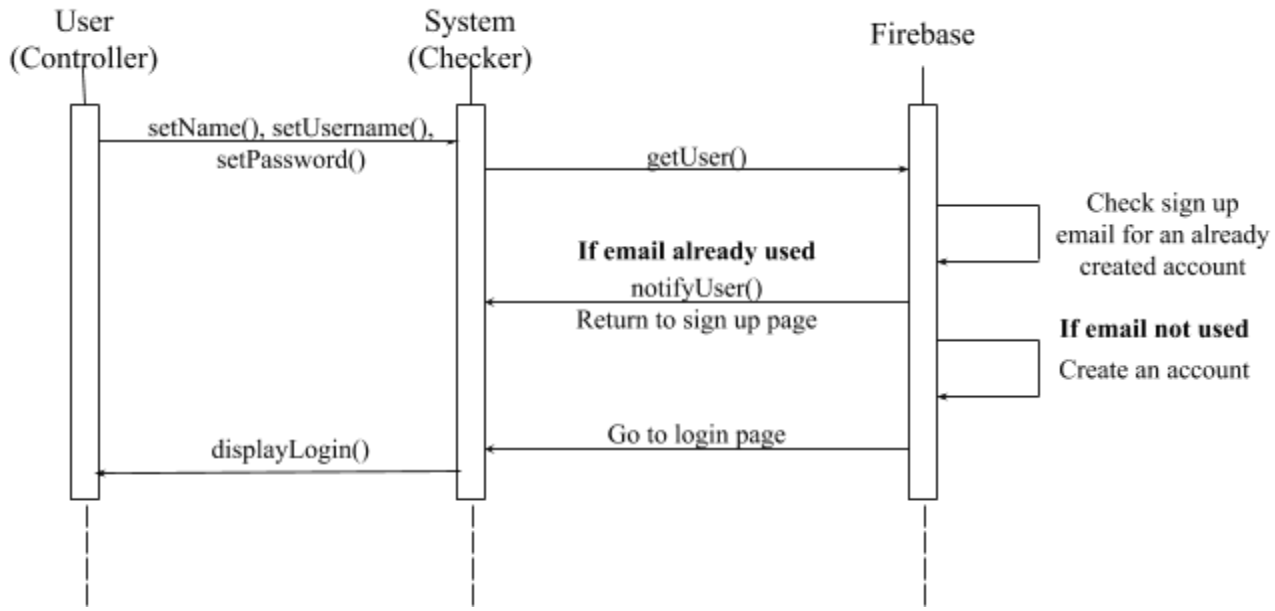
Diagram #11: User Login



Description of Use Case #18:

This use case demonstrates the process of logging into their created account. Once the "user" inputs the email and password associated with their account and presses the "login" button, the "system" contacts the real-time database hosted through "Firebase." "Firebase" checks to see if the email and password match the corresponding account using their email authentication feature. "Firebase" returns whether or not the "user" is authenticated to the "system." If the "user" is authenticated, the "system" signs "user" in and displays the home screen. If the "user" is not authenticated, the "system" notifies the "user" that the email and password combination do not match any account found and returns to the login screen for the "user" to try a new email/password combination.

Diagram #12: Create an Account



Description of Use Case #10:
This use case demonstrates the process of creating an account. The “user” inputs the first and last name, email, and password they wish to be associated with their account and presses the “sign-up” button. Once the button is pushed, the “system” contacts the real-time database hosted through “Firebase.” “Firebase” checks to see if the email used matches any already created account. “Firebase” then returns whether or not the “user” is already in the database to the “system.” If the email corresponds to an already created account, the “system” notifies the “user” and returns to the sign-up screen. The “user” can sign-up with a different email or navigate to the login screen. If the email used does not correspond to an already created account, an account will be created with the inputted first and last name, email, and password. Once the account has been created, the “system” goes to the login screen for the “user” to login with their newly created account.

Design Patterns

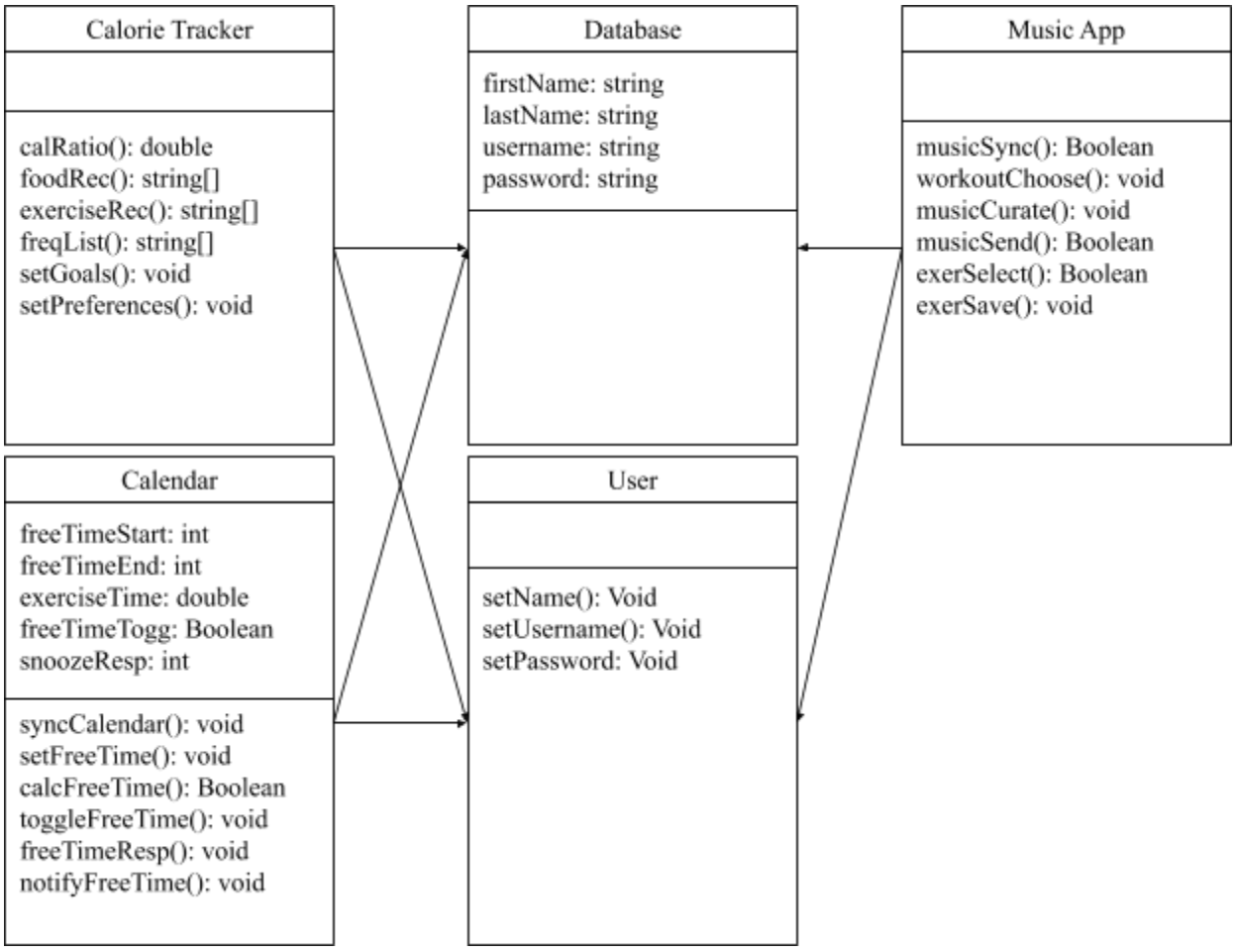
We chose the Publisher-Subscriber design pattern because of its indirect communication system. In this case, the subscriber could be the back end of our application while the publisher could be any third-party applications we work with. One of the advantages of using this design pattern is that it makes it easy to separate the responsibilities of the objects. The subscriber and publisher only have to be concerned with their own tasks.

Class Diagram and Interface Specification

Class Diagram

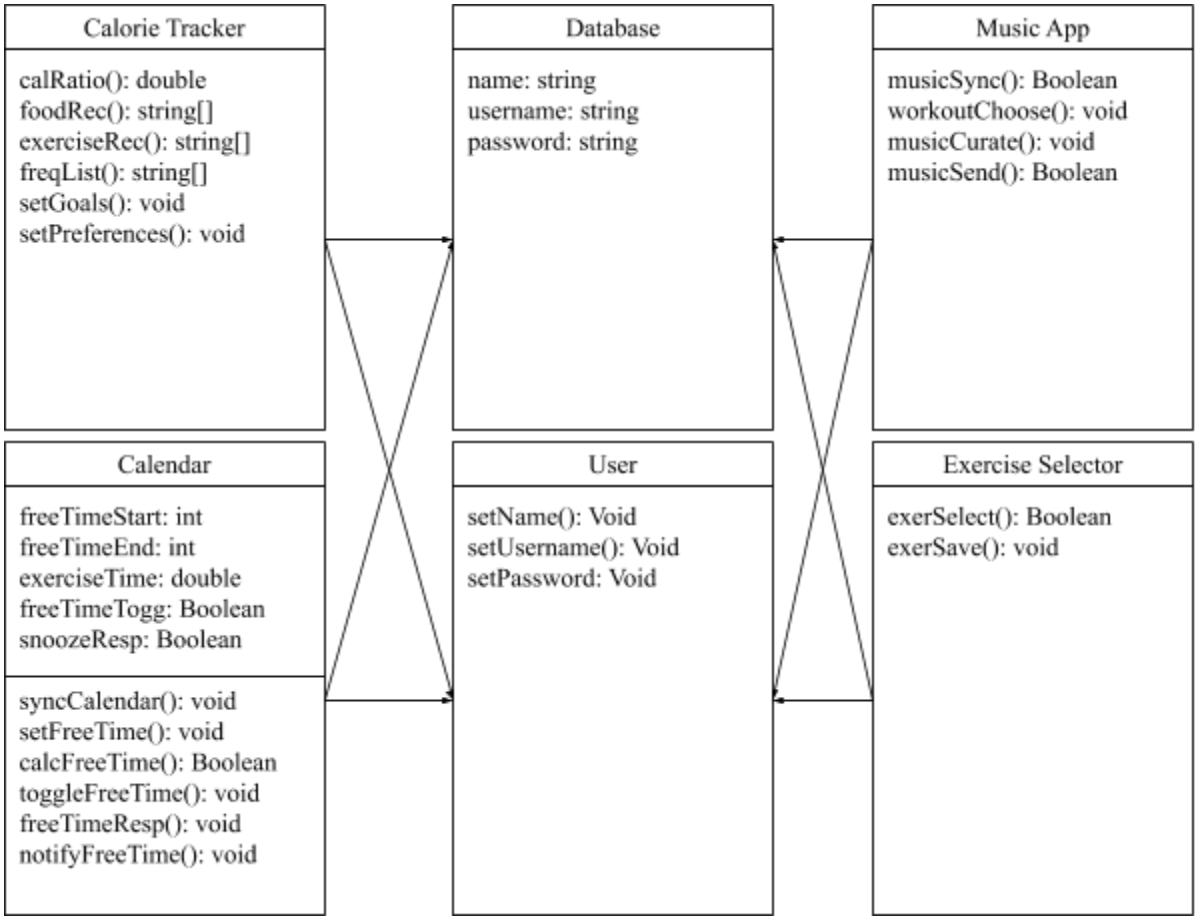
Below is our current class diagram.

Diagram #13: Updated Class Diagram



Below is our class diagram from our first report.

Diagram #14: Previous Class Diagram



Data Types and Operation Signatures

Table #11: Data Types and Operation Signatures

Class	Data Name	Data Type	Description
Database	name	string	String variable that stores the user's name for the given account
	username	string	String variable that stores the user's username for the given account
	password	string	String variable that stores the user's password for the given account
User	setName()	void	Sets the name for the given account

	setUsername()	void	Sets the username for the given account
	setPassword()	void	Sets the password for the given account
	getUser()	void	Retrieves user information
	notifyUser()	void	Notifies user of any changes
Calorie Tracker	calRatio()	double	Returns the ratio between the number of calories consumed and burned
	foodRec()	string[]	Returns a string array of food recommendations based on user goals and preferences
	exerciseRec()	string[]	Returns a string array of recommended exercises that would help the user meet the daily set goal
	freqList()	string[]	Returns user's favorite items, based on a manual selection or based on most frequent exercises and meals
	searchFood()	void	Searches for food item in the database.
	foodOptions()	string[]	Returns possible food items based on which foods match the keywords.
	calCons()	double	Returns calories consumed.
	addCal()	double	Adds calories consumed to the tracker.
	setGoals()	void	Sets the user's workout and calorie goals
	setPreferences()	void	Sets the user's diet preferences
Calendar	freeTimeStart	int	Integer corresponding to the time, during each day, in which free time notifications will be enabled
	freeTimeEnd	int	Integer corresponding to the time, during each day, in which free time notifications will be disabled
	exerciseTime	double	Double variable corresponding to the amount of free time needed for the user to implement a work out

	freeTimeTogg	boolean	Boolean variable corresponding to the status of free time notifications (ON/OFF)
	notifyTime	int	Integer value, that the user sets, that determines the amount of time between when “snooze” is chosen and when the next free time notification will be sent out (given there is enough free time)
	syncCalendar()	void	Sync the events from the user’s existing Google calendar with the calendar on the device
	setFreeTime()	void	Sets the amount of time needed for the user to implement a work out, and the time interval in which free time notifications will be enabled each day
	calcFreeTime()	boolean	Calculate if there is enough free time within the user specified time interval for the user to use to exercise
	toggleFreeTime()	void	Enable or disable free time reminders
	freeTimeResp()	void	Stores the user's response to a free time notification
	notifyFreeTime()	void	Sends a notification to the user, during the specified free time notification interval, that there is enough free time to work out
Music App	musicSync()	boolean	Gives the user an option to sync their personal account or create a new one and use a playlist generated by default
	workoutChoose()	void	Allows the user to select which workout the user wants to complete
	musicCurate()	void	Syncs the workoutChoose() function with the playlist generate
	contactApp()	void	Contacts app via API to play playlist
	musicSend()	boolean	Sends the selected playlist to the user

Traceability Matrix

Table #12: Class Traceability Matrix

Domain Concepts	Calendar	Calorie Tracker	Music App
Calendar	This domain concept mapped directly to its class.		
Calorie Tracker		This domain concept mapped directly to its class.	
Workout Notifications	This domain concept mapped directly to its class.		
Workout Selector		This domain concept mapped directly to its class.	This domain concept mapped directly to its class.
Music Playlist			This domain concept mapped directly to its class.

Design Patterns

From the interaction diagrams, we can see there is a creational design type pattern. For our database to be accessed from multiple locations in the code, we would have to make a DBConnection class. This pattern helps the communication between Calorie Tracker, Calendar Sync, and Music Feature to the database and user. For starters, a singleton class may handle a communication string or port accessible with just one method. The creational design patterns consist of abstract factory, builder, factory method, object pool, prototype, and singleton. We also introduced a singleton class for our code since our code requires a database relation. This pattern makes the information from the user and database more accessible to the other classes. The design for the Factory Method specifies an interface that delays the entity to derived classes. The created classes in this framework are used to instantaneously extract artifacts that are unique to the application's needs. These would be used for obtaining the login information and data from the several app integrations and stores data under those users. We also see an abstract factory pattern because all the data from the user, such as the calorie tracker, all go under the class of the user, because each user contains a different type of info for all these classes.

Object Constraint Language Contract Specification

Calorie Tracker:

- Search Foods
    - Invariant: At least 1 food item is searched.
    - Pre-Conditions:
        - User wants to know the amount of calories consumed.
        - User has consumed the food item(s) being searched.
    - Post-Conditions:
        - The number of calories in food item(s) are added to calorie tracker.
        - User is sent back to the search food page.

Calendar Sync:

- Handle Reminder
    - Invariant: There is at least 1 reminder.
    - Pre-Conditions:
        - User has given app permission to read Google calendar.
        - User has entered the min. amount of free time needed to exercise.
        - User has set a time interval so free time reminders are enabled.
        - There is enough free time to exercise.
    - Post-Conditions:
        - Acknowledged: Reminders are paused until the next free time interval of the following day.
        - Snoozed: Another reminder will be sent after the user's allotted snoozed time if enough time to exercise is still available.

Music Curator:

- Playlist Creation
    - Invariant: At least 1 song will be in the playlist.
    - Pre-Conditions:
        - User gives the app permission to use personal Spotify subscription account to create a playlist.
        - User enters preferred music genres for listening.
    - Post-Conditions:
        - System creates a playlist based on the data to match the songs to intended workouts and workout time.
        - System connects to Spotify to play curated songs from a playlist.

System Architecture and System Design

Architecture Styles

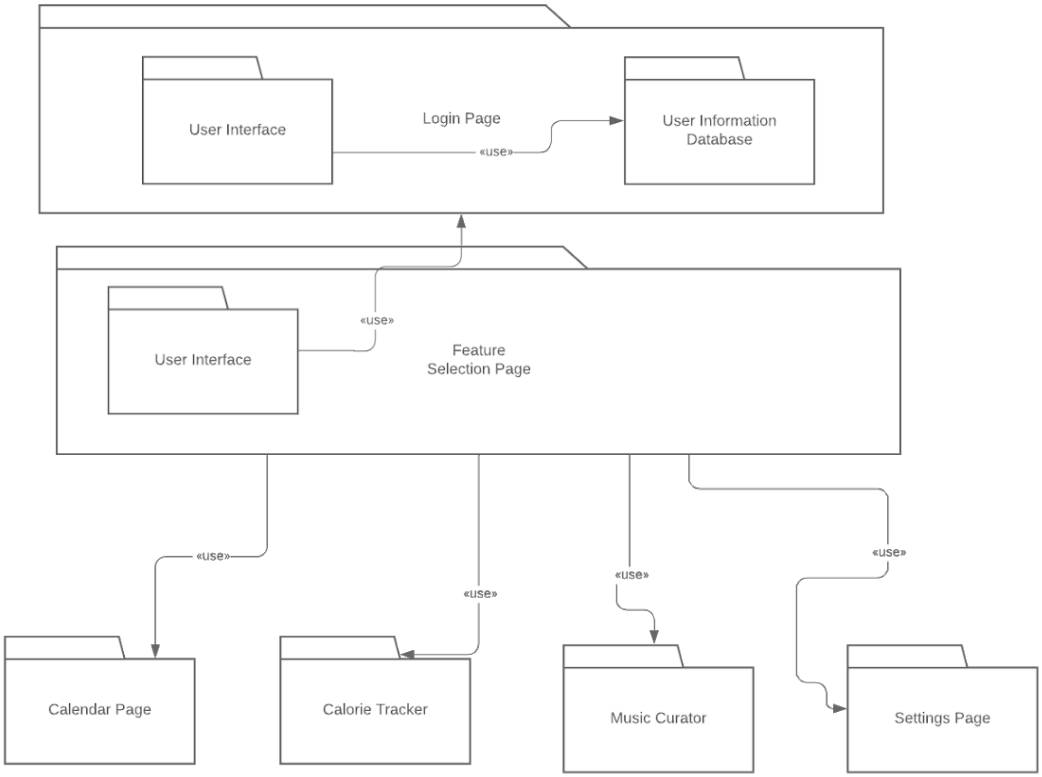
Our system has a component-based system architecture style in the context of user experience because the app is broken down into several logical subsections, or classes, with distinct functions. The main subsections are our user options - the Calendar Page, Calorie Tracker, Music Curator, and Settings page. Once a user logs in, they are able to navigate to one of these four pages at a time. The Calendar class contains member functions which all work towards the goal of sending the user notifications to work out during free time in their Google Calendar. The Calorie Tracker class contains member functions which all contribute towards the prompting, storing, and displaying of calories consumed and burned by the user. The Music Curator class contains member functions which all contribute to either prompting the user for workout/music preference information, storing that information, or the creation of unique playlists based on that information. The Settings page contains member functions focused on gathering and saving user login and name information.

The app is component-based for the purpose of clear, streamlined navigation. Instead of being dependent on one another, each aforementioned class has member functions and attributes unique from other subgroups. This decreases the likelihood that bugs in one class will affect the entire code, separating the services streamlines the user experience. A user can select an option and only provide information needed for that subsection.

In the context of subliminal operations within classes, our system architecture style can also be described as being event-driven. Many member functions of the app use or display items based on user input and data. The Calendar Page and resultant workout notifications rely on data from the user's Google Calendar, such that a change to the calendar changes the notifications. The Calorie Tracker relies on user-inputted food and workout data. The Music Curator relies on user-inputted workout and music preference data. The notifications, calorie data, and personalized playlists that the user receives from the app are all event-driven because user input is needed in order for the user experience to be personalized, a prominent theme in the app and many of our use cases.

Identifying Subsystems

Diagram #15: Identifying Subsystems



Mapping Subsystems to Hardware

Our system does not need to run on multiple computers.

Persistent Data Storage

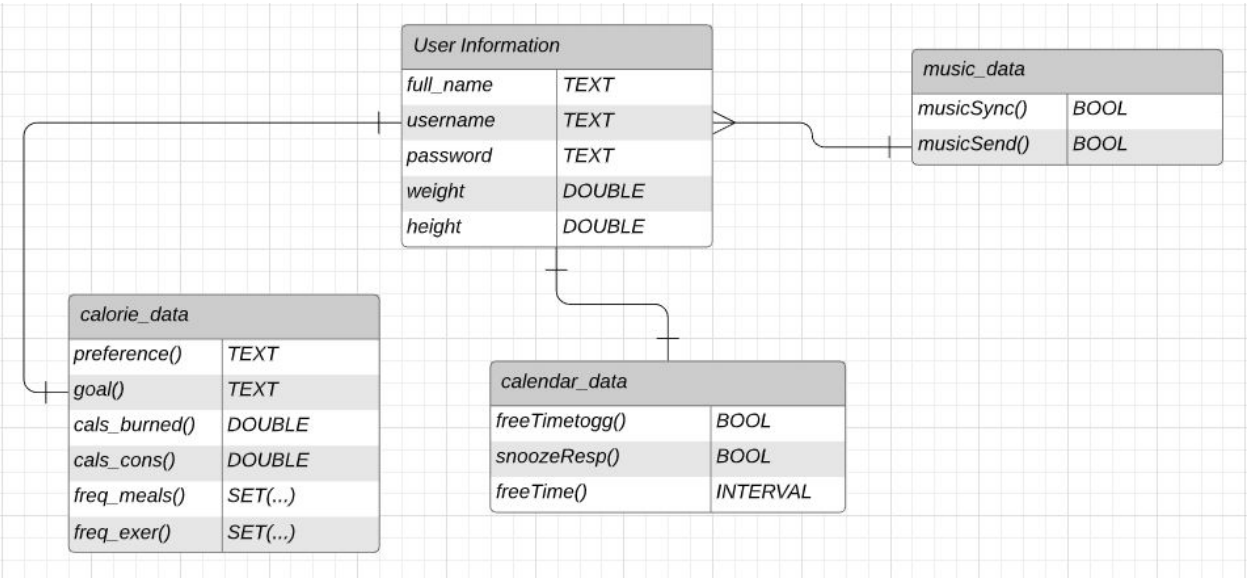
Our system relies on data that will outlive a single execution of the system, so we will use a relational database to store all of the users data.

Table #13: Persistent Data Storage Strategy

Persistent Objects	Storage Management Strategy
User information (username and password)	Relational Database
User diet preferences	Relational Database
User music preferences	Relational Database
User health goals	Relational Database
User total calorie count	Relational Database
User designated Free Time	Relational Database

Workout Reminders	Relational Database
Saved Workouts	Relational Database
Saved Meals	Relational Database

Diagram #16: Database Schema



Network Protocol

This does not apply to our group.

Global Control Flow

Execution Orderness:

Due to the different features of our project, our system is both procedure-driven and event-driven. The user must follow the same steps for the music and calorie features of the project. However, for the calendar, it is dependent on the previous events that are logged into the user's calendar already.

Time Dependency:

There are timers in our system. For the 'snooze' option, there is a timer in place to wait for 30 minutes before sending another notification to the user.

Hardware Requirements

Our system depends on numerous resources. We require server communication access to the Google Calendar. The account information is sent via API

integration. A SQL database is dependent on for recording the information about calories in food and calories that would be burned from exercise. The Spotify integration also requires server communication. There is a network connection required that has a minimum network bandwidth.

We also require a screen display from a smartphone for our application. It is optional, but for additional perks from the app, we would require a smartwatch screen by, size 34 mm or higher, to display notifications. If there is a smartwatch, we require server communication from the smartphone to the smartwatch.

Algorithms and Data Structures

The calendar feature uses the Google Calendar API in order to be able to pull data from the user's pre-existing calendar information. In addition, the app is able to create new events in the calendar. This could allow for the user to keep a record of the days and time that they do work out. Utilizing the API rather than creating a separate calendar within the app allows for us to integrate our app with what the user already has on their phone. Furthermore, it allows for our app to be created without "reinventing the wheel." It also allows for the user to be provided with a seamless integration between the app and their calendar, which they previously spent time creating. This feature does not use any complex data structures.

The calorie tracker feature implements an API developed by Edamam. When the user inputs a food item name or an UPC barcode code, our system sends a GET request to the Edaman API. This request searches through the Edamam database and returns nutritional information about the requested food item. Our system parses through the information and returns the name, an image, and the number of calories in the food item to the user. If the user decides to add the food item to the calorie tracker, the number of calories will be added to the database. The calorie tracker feature stores the user's most frequent exercises and meals using linked lists. The user can add data dynamically so we will use a linked list because the size of the list is unknown at compile time. The performance of linked lists is a drawback, but the flexibility is more important for our feature.

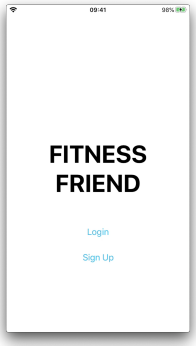
The music curator feature utilizes the developer API plug-in provided by Spotify, which will be integrated within our app/OS as a shell. The component of the music curator which matches intended workouts to songplay implements an algorithm that plays songs with BPMs matching the pace of each exercise. This algorithm can be thought of as a song locator, and it requires information about preferred genres, communication with the spotify API, as well as some randomization so the same workout can result in different

mixes. Another algorithm builds the playlist as each workout component is matched to a song. The music curator utilizes temporary memory via a linked list to build the playlist before it is saved. The music curator also uses an SQL database which is a part of the developer API provided by Spotify.

User Interface Design and Implementation

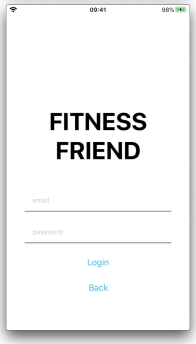
Select Page

When the user first opens the app, they will be taken to a select page where they will be able to choose whether they want to log in or sign up. If the user accidentally navigates to the sign up page but realizes they want to go to the login page, or vice versa, they will be able to easily navigate back to the select page where they can choose the correct option.



Login Page

From the login page, the user is able to input the email and password associated with their account in order to log in. If the email and password combination match an account stored in our database, they will be successfully logged in and rerouted to the homepage.



Sign-Up Page

The user is able to input the first name, last name, email, and password they wish to create an account with. If the email chosen corresponds to an already created account, the user will be notified and a new account will not be created. If the email inputted does not match an already created account, a new account will be created and stored in our database. They will be notified that their account was successfully made and will be taken back to the sign up page. From this page, they will then be able to navigate back to the select page where they will be able to sign in with their newly created account.

Homepage

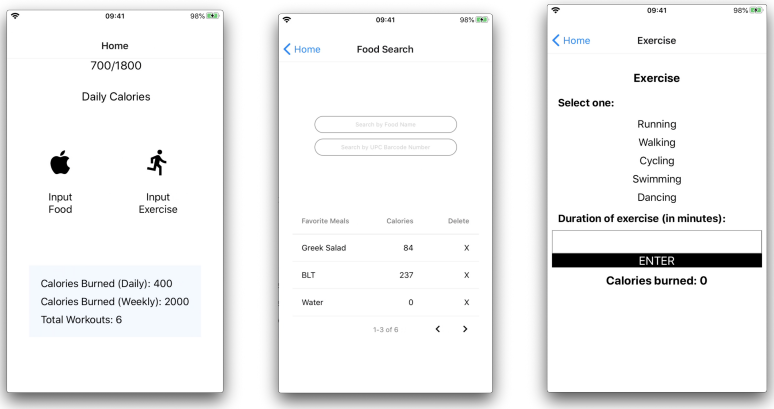
The homepage displays the name of the app displays across the top of the app. On this page, there are four item selections. The Calendar, Calorie, and Music features are all displayed to be navigated to, as well as the Settings page.

Calendar Sync

Instead of providing the user with a view of their calendar, one that they would already have access to in their phone's build in calendar app, our calendar page will consist only of the user's preferences for free time notifications. This was implemented by using the same format and layout as the settings page and login page except with the substitution of dropdown menus for the input boxes. Like before, the user will have a large variety of dropdown options to choose from in order to accomodate any user.

Calorie Tracker;

For the Calorie Tracker section, there are separate pages based on the different actions the user could take. This makes it easier for the user to navigate the app and make it clear what actions are associated with the tracker. There will be a Calorie Tracker home screen that will list the actions available and will have the overall intake of calories at the top of the screen. From there, the user could go to either the exercise selector screen or the adding calories screen. In the exercise screen, the user can choose what exercises they've done and that will be calculated. Then, the calories burned can be taken out of the Calorie Tracker. In the adding calories screen, the user can search the specific food items they've consumed and the corresponding calories will be added to the overall tracker.



Music Curator:

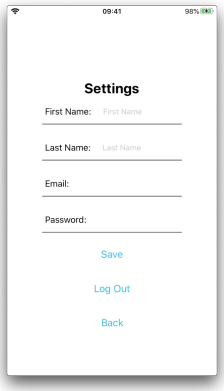
Spotify is included as a shell within our app which allows us to not make use of the Spotify app on the user's phone. The connection can be established through the application and the curator can work as intended to. The Spotify app integrated with our application has features that the default app does not have. These features are added by the Android SDK, a package included in the developer API provided by Spotify.

The Exercise Selector portion of the Music Curator app includes separation of common workouts in subsections followed by division of those subsections into subsections, along

with time and pace options. The subsections include cardio, muscle training, and yoga. The end of an exercise selection gives the user the option to add another or finish.

Settings Page

The settings page will be the place where the user will be able to update the first name, last name, email, and password associated with their account. Once the account changes are made, they will be able to use the save button to update this information in our database. From this page, they will also be able to log out of their account and be taken back to the select page where they can log back in or create a new account.



Design of Tests

Calendar Sync

For the free time notification system, the Google Calendar API will be integrated in order to pull from a user's pre-existing calendar. To test whether or not the calculation system works, we will have to create a Google account and begin operating as a user to see if we receive the necessary notifications to properly utilize this feature. This notification will notify the user if there is enough free time throughout the day, during the free time interval, to implement a workout. We will be testing the integration of the API and free time calculation system utilizing the following steps in the app:

1. The user will set their free time notification interval and the amount of free time needed for a proper workout page on the calendar page
2. The user will sync their Google Calendar to the app
3. The app will retrieve the events from the synced calendar and calculate whether enough free time is available during the pre-set free time interval
4. If the test is successful AND there is enough free time during the day, the app would output the expected free time notification

Google Calendar Scenarios:

1. There is enough free time during the free-time interval specified.
2. There is no free time available during the entire day.
3. There is enough free time but it is **not** during the free-time interval specified.
4. There are numerous free time slots during the free-time interval specified.
5. There is free time but not enough available as determined by the user.

Calorie Tracker:

For the calorie tracker, we will focus on how the calories are calculated based on the exercises selected and the foods searched. We will be testing our calculation algorithm to check the accuracy of the overall calories displayed on the screen as well as the accuracy of the calories assigned to each exercise and the food calories found based on the search. It would be difficult to get the exact numbers, but the goal is to get them as accurate as possible so the user doesn't have to do any calculations by hand. Here are some actions the user should be able to perform:

1. The user finishes a specific exercise and can go to the exercise selector screen to choose the exercise from a list. The burned calories associated with that activity can then be subtracted from the overall calories.
2. The user finishes a meal and goes to the calories screen and searches for what they ate. The calories corresponding to each item is added to the overall calories.
3. The user checks overall calories for the day to see if they have reached their goal.

Calorie Tracker Scenarios:

- The food searched is **not** found.
- The amount of calories burned exceeds the amount of calories consumed.
- The portion of food eaten does **not** correspond to the calories shown.
- The exercise completed is **not** an option on the list.

Music Curator:

For the music curator, we will integrate Spotify within our app. For this we will be using the developer API, Android SDK, via the following steps:

1. An application is registered with the Spotify dashboard which provides us with a Client ID as well as URI. The app relaunches every time the user logs in.
2. The Android SDK creates a link between the app server and Spotify database.
3. The user logs into Spotify via the app, which gives the API access to the account.

Spotify Music Curator Scenarios:

- The user is prompted to enter exercises using the exercise selector subsection of the music curator. The user is able to enter the different exercises completed.
- The curator returns a Spotify playlist of a specific length of music with a high BPM and then a lower BPM fit for high intensity workout, followed by anaerobic exercise. It is then followed by music with a calm BPM for the cool down.
- If the user enters the same workout while reusing the music curator, the playlist should be carried out with a slightly different list of songs (via randomization).

Table #14: Testing Overview

Calendar Sync	This feature will be testing the function coverage: whether or not the free time notification feature is able to pull from an existing calendar using the Google calendar API and accurately determine if there is enough free time to work out during the free time interval. This is very important because the integration of the API will help with efficiency. Majority of technology users use Google Calendar as their default. So instead of allowing the user to input their schedule twice, we will use what they have already created. This will better integrate it so the user experience is better and more comfortable.
Calorie Tracker	This feature will test the accuracy of the calculation algorithm implemented in the back end. It will test the calculation of calories for the foods consumed and for the exercises selected. Adjustments will be made as needed based on the results of the tests.
Music Curator	For this feature, we will be testing the function coverage, whether or not the user is able to log into/create their own Spotify account through the web. We will be using an Android SDK provided as a part of the Spotify developers API to establish connection between our application and the Spotify server. For this feature we are also testing the accuracy and successful randomization of our song matching algorithm given the multiple inputs about user preference.

Feature	Integration testing
Calendar Sync	The integration feature for the calendar sync involves incorporating the google calendar API and using it in the app. The calendar integration will have the users most recent events and be able to distinguish the free time the user has. To figure that out, the app would incorporate the events/tasks the user has and find blocks of time where the user is not scheduled for anything, and then notify the user for suggested times/reminders to work out. If the user has some events and has time in between them and our app recognises that free time block. Then we know it works, because the app will create a free time slot, and send the notification to the user. If the notification is sent, then the calendar sync feature works successfully.
Calorie tracker	The integration feature for the Calorie Tracker involves incorporating a calorie database to the app. The calorie database will be made beforehand and will consist of calories assigned to common food items. The user can select any of the food items in the database and the result will be fed into the calculation. The

	same approach will be used to select the exercises completed.
Music Curator	The integration feature for the curator involves connecting the dummy app to integrate spotify as a shell. This is the main testing that has to be done, as everything after the spotify app integration involves the use of the android SDK API. The algorithm will be tested by launching the app, which will then enable the user to log in to Spotify via the app. If the user succeeds, our first demo testing is complete; however, if there are failed login attempts then some debugging will have to be done. The testing will be performed by the developers as well as testers.

History of Work, Current Status, and Future Work

Throughout the past four months of working on this project, we have had ups and downs when attempting to reach milestones and deliverables. In the beginning, there was a lack of knowledge across the team of creating an app and using Javascript. We then, also, decided to take out a feature of the app and merge the group working on the feature with two of the other sub-groups. This caused a change in teamwork and expectations amongst the group's deliverables. However, through research and trial and error, about one month into the project, we found a platform that we could use to implement the features of the app. Snack Expo allowed for each individual and sub-group to work separately and still allow for other groups to view their code and UI. It did give us some difficulty in integrating all of the code with each other as often as we thought we would because we had not recognized that Snack Expo did not always show the most updated changes. Because of this, the sub-groups integrated their code in a timely manner but the overall group failed to do so until the end of the project timeline.

Key accomplishments of this project include:

- Inventing an app to combat a specific issue that is existent for our target audience
- Creating features that are not currently existent for smartwatches/smartphones
- Syncing the app with a platform that 60% of consumers have an account with

There is room for continuation on this project. One possible direction for moving this project forward is moving the app to a stand alone platform so that it does not depend as heavily on the Snack Expo library. Furthermore, this would allow for the app to be present in the Apple App Store and the Google Play Store. Finally, it would be extremely helpful to disclose information and directions on syncing the app on the users' smartwatch and implement additional 'watch only' features since the app is targeted towards smartwatch owners but all of the features currently work on phones as of now.

As of now, the only benefit to having a smartwatch is that the notifications would also appear on the watch, which is helpful if the users' phone is not nearby.

References

API Integration:

1. “Food Database API Documentation” https://developer.edamam.com/food-database-api-docs
2. “Integrating Google Sign-In” https://developers.google.com/identity/sign-in/web/sign-in
3. “Adding Push Notifications to a React-Native App Using Expo” https://medium.com/@jb_75888/adding-push-notifications-to-a-react-native-app-using-expo-2cfe0b4d74fb
4. “React Native + Firebase #6” https://www.youtube.com/watch?v=R2D6J10fhA4
5. “React Native Firebase Tutorials” https://github.com/nathvarun/React-Native-Firebase-Tutorials/blob/master/Project%20Files/1%262%20Email%20Auth/Complete/App.js https://github.com/nathvarun/React-Native-Firebase-Tutorials/tree/master/Project%20Files/6%20Push%20Notifications
6. Firebase Save Data https://firebase.google.com/docs/database/admin/save-data
7. Hide/Show Component https://aboutreact.com/example-to-hide-show-component-in-react-native/
8. Firebase Read/Write Data https://firebase.google.com/docs/database/web/read-and-write
9. Push Notifications Example https://dev.to/rishikeshvedpathak/implement-push-notification-in-react-native-using-expo-in-just-5-min-4c1m

Analytics and SQL:

1. “FitBit Health Monitoring Analytics” https://www.ece.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2014-g5-report3.pdf
2. “SQL Data Types for MySQL, SQL Server, and MS Access” https://www.w3schools.com/sql/sql_datatypes.asp

Fitness Research:

1. “Learn About the Four Different Types of Exercise” https://www.captel.com/2013/10/learn-four-different-types-exercise/
2. “Android SDK Quick Start” https://developer.spotify.com/documentation/android/quick-start/

Snack Expo:

1. “Push Notifications - Expo Documentation”

https://docs.expo.io/versions/latest/guides/push-notifications/
2. “Using Firebase”
https://docs.expo.io/versions/latest/guides/using-firebase/

Additional Resources:

1. “Software Engineering Book”
http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf
2. “Creating a watchOS App”
https://developer.apple.com/tutorials/swiftui/creating-a-watchos-app
3. “3 Music Apps That Will Create a BPM-Based Playlist for Your Workout”
https://www.mensjournal.com/health-fitness/3-music-apps-will-create-bpm-based-playlist-your-workout/
4. “Use Case Points”
https://en.wikipedia.org/wiki/Use_Case_Points
5. “Design patterns”
https://www.geeksforgeeks.org/design-patterns-set-1-introduction/?ref=lbp
https://sourcemaking.com/design_patterns

Individual Breakdown

Table #15: Individual Breakdown

			Amber Haynes	Tiyon King	Jenna Krause	Mya Odrick	Devvrat Patel	Andrew Rezk	Maria Rios	Shivani Sunil	Hedaya Walter
Summary of Changes				25%	25%	25%			25%		
Customer Statement of Responsibilities				20%	20%		20%		20%	20%	
Glossary of Terms				50%		50%					
System Requirements	Enumerated Functional Requirements						50%			50%	
	Enumerated Non-Functional Requirements						50%			50%	
	User Interface Requirements						50%			50%	
Functional Requirements Specification	Stakeholders, Actors, & Goals			14.28%	14.28%	14.28%		14.28%	14.28%	14.28%	14.28%
	Use Cases		11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%	11.1%
	System Sequence Diagrams		33.3%		33.3%				33.3%		
Effort Estimation									100%		
Domain Analysis	Domain Model	Model			50%		25%			25%	
		Con-cept Def.	20%	20%	20%	20%	20%				
		Ass.	20%	20%	20%	20%	20%				
		Attr.	25%	25%	25%	25%					
		Trace Matr.	25%	25%	25%	25%					

		Amber Haynes	Tiyon King	Jenna Krause	Mya Odrick	Devvrat Patel	Andrew Rezk	Maria Rios	Shivani Sunil	Hedaya Walter
Domain Analysis cont.	System Operation Contracts	50%						50%		
	Mathematical Model									100%
Interaction Diagrams	Diagrams	20%	20%	20%	20%			20%		
	Design Patterns							100%		
Class Diagram & Interface Specification	Class Diagram			100%						
	Data Types & Operation Signatures		25%	25%	25%			25%		
	Traceability Matrix		20%	20%	20%			20%	20%	
	Design Patterns						100%			
	OCL Contract Specification							100%		
System Architecture and System Design	Architecture Styles	100%								
	Identifying Subsystems	50%			50%					
	Mapping Subsystems to Hardware		100%							
	Persistent Data Storage		50%		50%					
	Network Protocol		100%							
	Global Control Flow		100%							
	Hardware Requirements		50%						50%	

		Amber Haynes	Tiyon King	Jenna Krause	Mya Odrick	Devvrat Patel	Andrew Rezk	Maria Rios	Shivani Sunil	Hedaya Walter
Algorithms & Data Structures		20%	20%	20%	20%				20%	
User Interface Design & Implementation		14.28%	14.28%	14.28%	14.28%	14.28%		14.28%	14.28%	
Design of Tests		12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	
History of Work, Current Status, and Future Work			100%							
References		16.6%	16.6%	16.6%		16.6%	16.6%	16.6%		
Project Management	Merging of Contributions		25%	25%	25%			25%		
	Formatting		50%	50%						
	Individual Breakdown		100%							
Breakdown Consent		AH	TK	JK	MO	DP	AR	MR	SS	HW