Fitness Friend

Group 6

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

March 1, 2020 Software Engineering

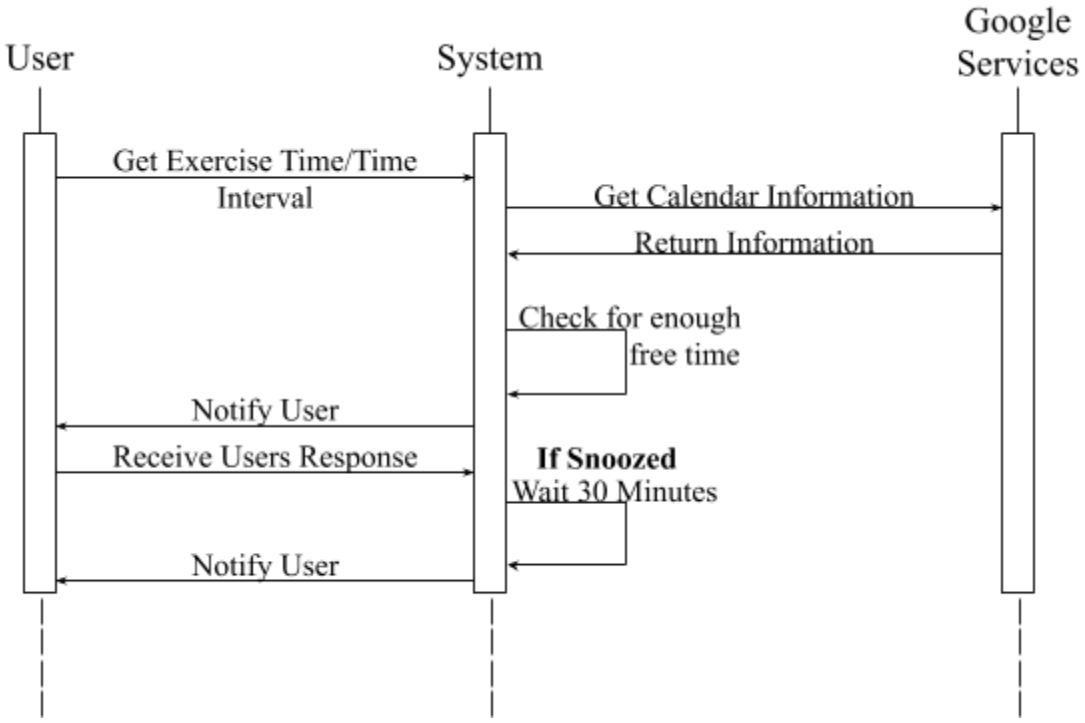
Table of Contents

Individual Breakdown

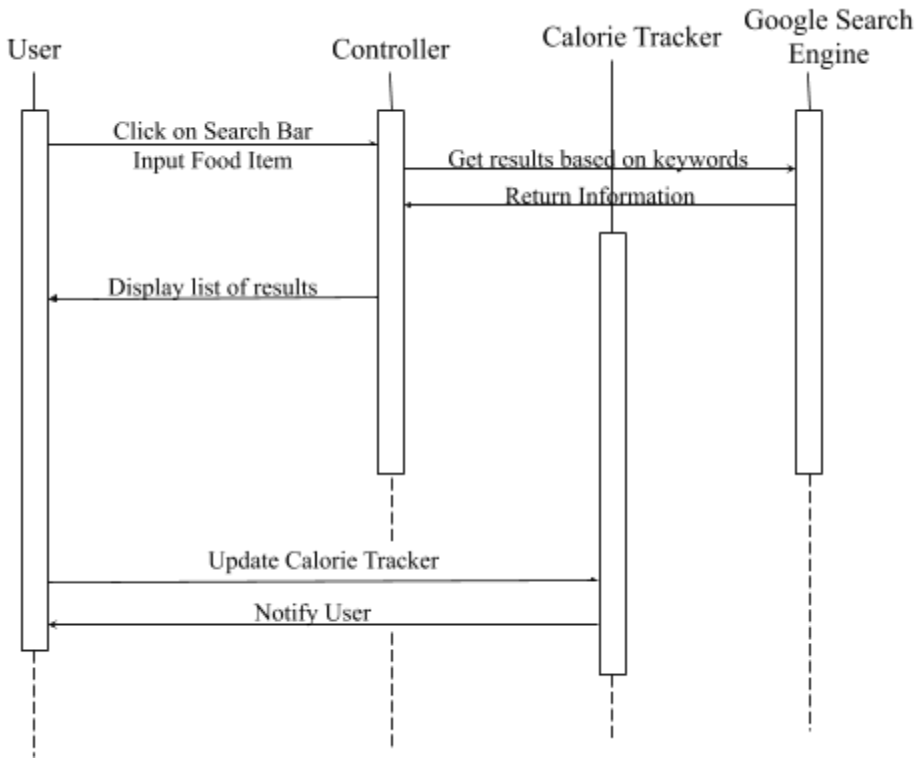
		Amber Haynes	Tiyon King	Jenna Krause	Mya Odrick	Devvrat Patel	Andrew Rezk	Maria Rios	Shivani Sunil	Hedaya Walter
Interaction Diagrams				33%	33%			33%		
Classes & Specs	Class Diag & Descr.			100%						
	Signatures			33%	33%				33%	
Sys Arch & Design	Architectural Styles	50%	50%							
	Identifying Subsystems	50%			50%					
	Mapping Hardware		100%							
	Persistent Data Storage		50%		50%					
	Network Protocol		100%							
	Global Control Flow		100%							
	Hardware Requirements		50%						50%	
Algo. & Data Struct.		20%	20%	20%	20%				20%	
User Interface Design & Implementation		20%		20%		20%		20%	20%	
Design of Tests		14.2%	14.2%	14.2%		14.2%	14.2%	14.2%	14.2%	
Project Management &	Meeting Attendance	16.6%	16.6%	16.6%	16.6%		16.6%	16.6%		
	Merging the Contributions from		50%	50%						

Plan of Work	Individual Team Members									
	Project Coordination & Progress Report	16.6%	16.6%	16.6%			16.6%	16.6%	16.6%	
	Plan of Work		25%	25%		25%			25%	
	Breakdown of Responsibilities		20%	20%	20%	20%			20%	

Interaction Diagrams

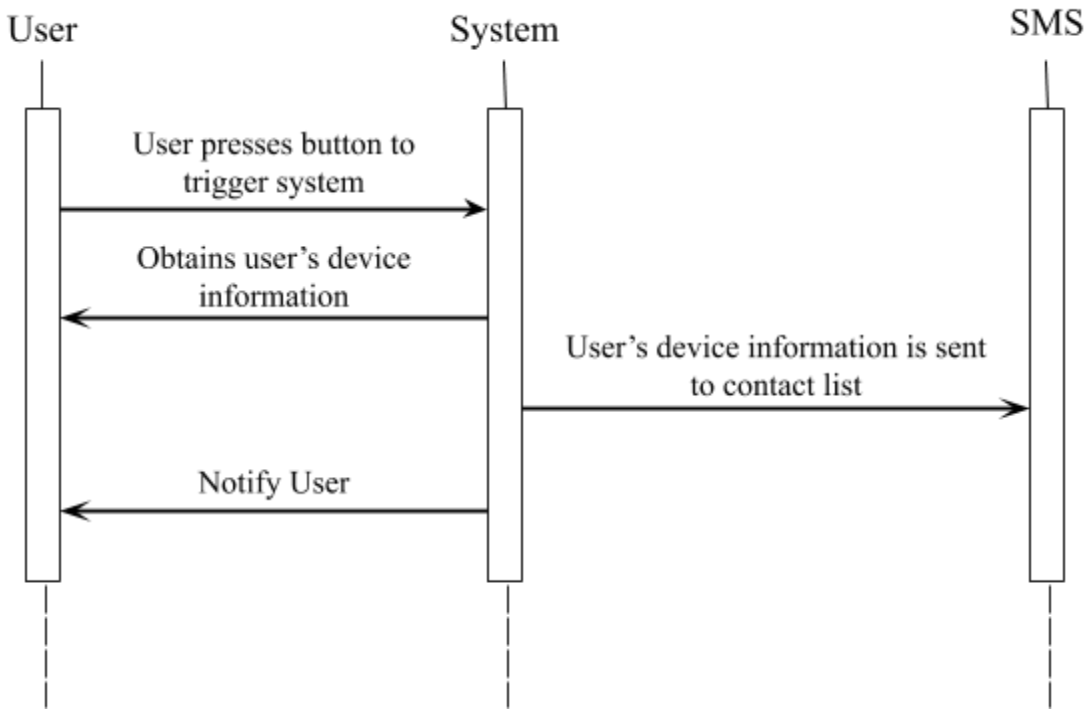


Description of Use Case #5:
This sequence is used to notify the “User” of free time that is available to be used for exercise. Once the “User” inputs the required amount of free time necessary to work out and the interval of time in which free time notifications will be enabled, the “System” will send a request to “Google Services” for the user’s calendar information. Then, the “System” will check for free time which meets the two inputted requirements. This will be done to update and see where the system creates a block of free space. After that, if there is enough free time within the free time notification interval, the device will send a notification to the “User” informing them of this availability. The “System” will then wait for a response from the “User” and then act accordingly. In this case, the Interface Segregation Principle is used, being able to understand each class in isolation.

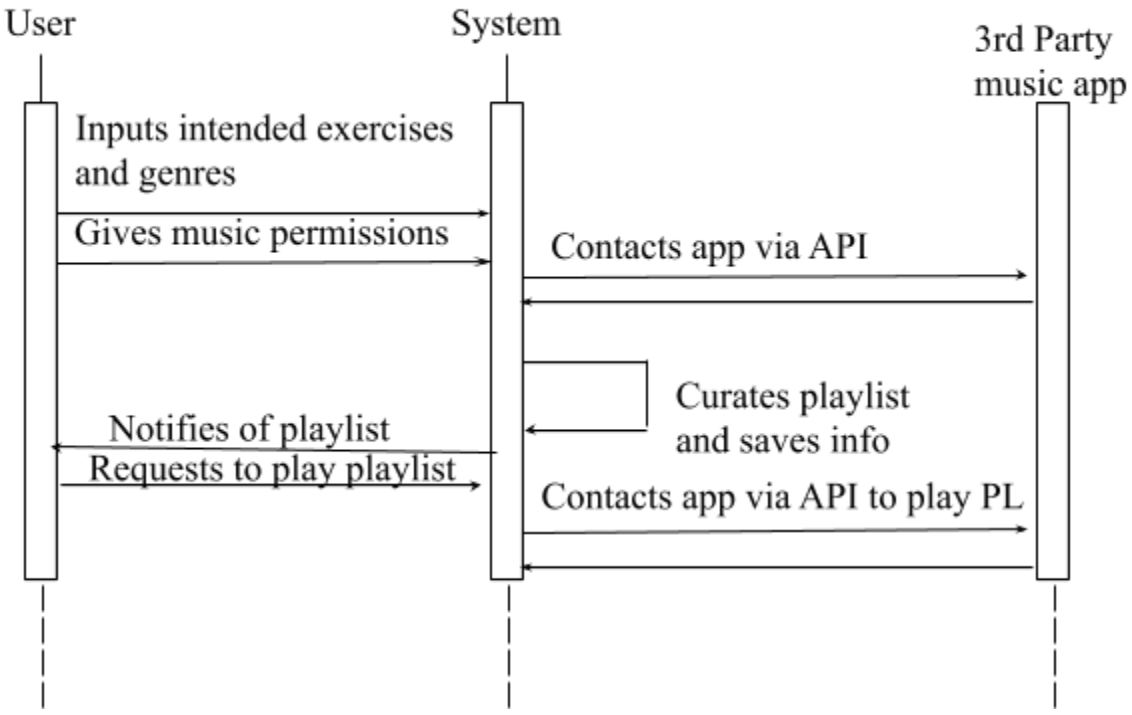


Description of Use Case #7:

This use case shows the sequence of the search function within the calorie tracker. In this case, we use the Single Responsibility Principle (SRP). The search engine is responsible for relaying 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. We also use 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. These independent functions are combined to create success for this use case.



Description of Use Case #10:
The process starts with the "User" pressing a button to alert the "System" that the "User" is in some type of danger. The "System" grabs the user's device information, the most important being the current location, and sends this information to the contacts listed on the "User's" premade emergency contact list. The "System" notifies the "User" that this information has successfully sent. Ideally, the "User" will have their heartbeat monitored during this time. Here, we can use Liskov substitution principle (LSP) to create an algorithm that would work on any future interface that we wish to implement. This would make it easier to modify the feature depending on the needs of the User and make any necessary improvements.

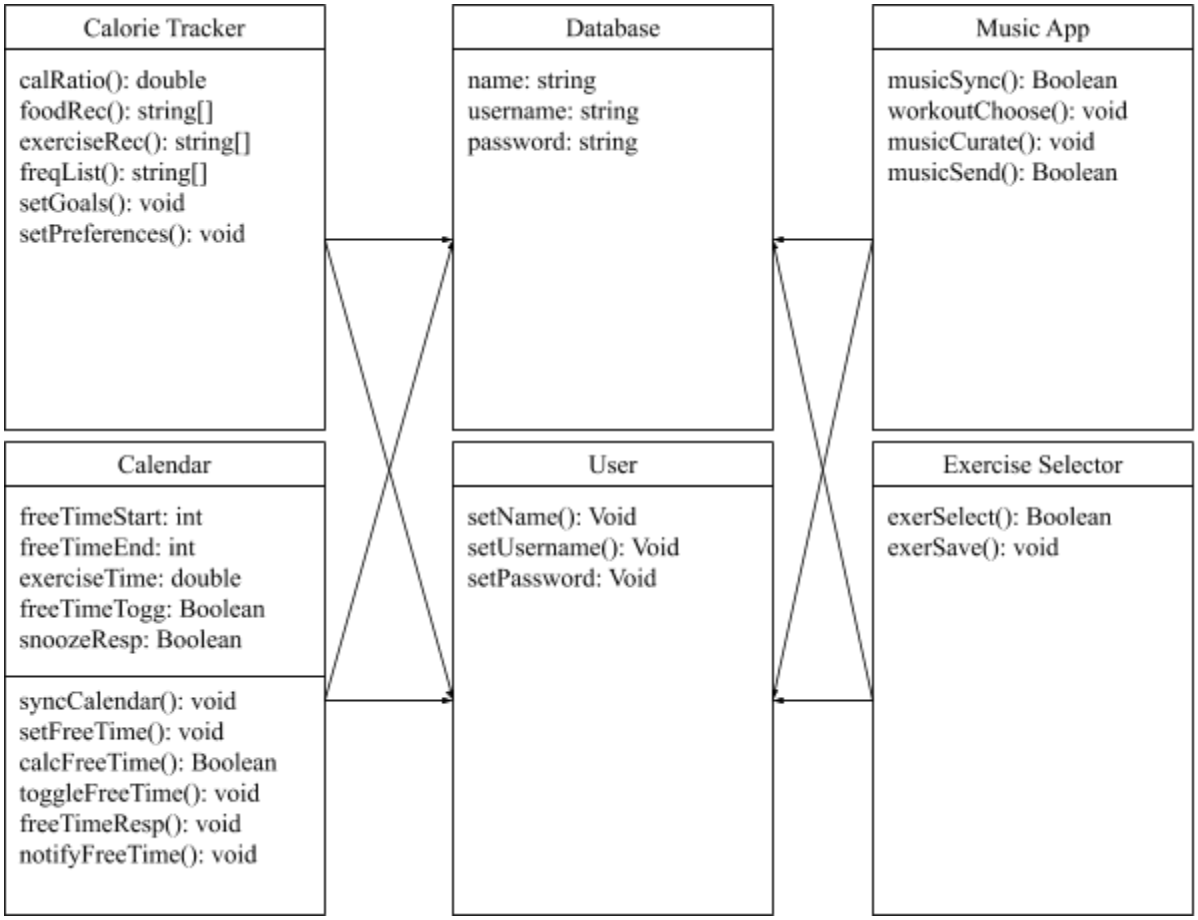


Description of Use Case #13:

This use case shows the process of the creation of the curated music playlists. Once the "User" inputs the exercises they will be performing, music genres they wish to listen to, 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 needed information, the "System" is able to create a curated playlist that meets the "Users" inputted needs. Once the music playlist has been created, the "System" will notify the "User" indicating that their custom playlist is complete and is ready to be played. If the "User" indicated that they wish to play the songs, the "System" will contact the "3rd Party Music App" to retrieve the music requested in order to play the song. In this case, the Interface Segregation Principle is used, being able to understand each class in isolation.

Class Diagram and Interface Specification

Class Diagram



Data Types and Operation Signatures

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.

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 daily set goal

freqList(): string[ ]

Returns user's favorite items, based on a manual selection or based on most frequent exercises and meals

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).

snoozeResp: Boolean

Boolean variable that is toggled "True" if the user snoozed the response and will be toggled "False" if a snooze was not pressed.

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 either sync their personal account to spotify 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

musicSend(): Boolean

Sends the selected playlist to the user

Exercise Selector:

exerSelect(): boolean

Returns the amount of calories from selected exercise and adds exercise to user's information

exerSave(): void

Adds the selected exercise to the user's frequent exercises list

Traceability Matrix

Domain	Calendar	Calorie Tracker	Music App	Exercise Selector

Concepts				
Calendar	This domain concept mapped directly to its class.			
Calorie Tracker		This domain concept mapped directly to its class.		
Workout Notifications				These domain concepts were reduced to one class.
Workout Selector				
Music Playlist			This domain concept mapped directly to its class.	

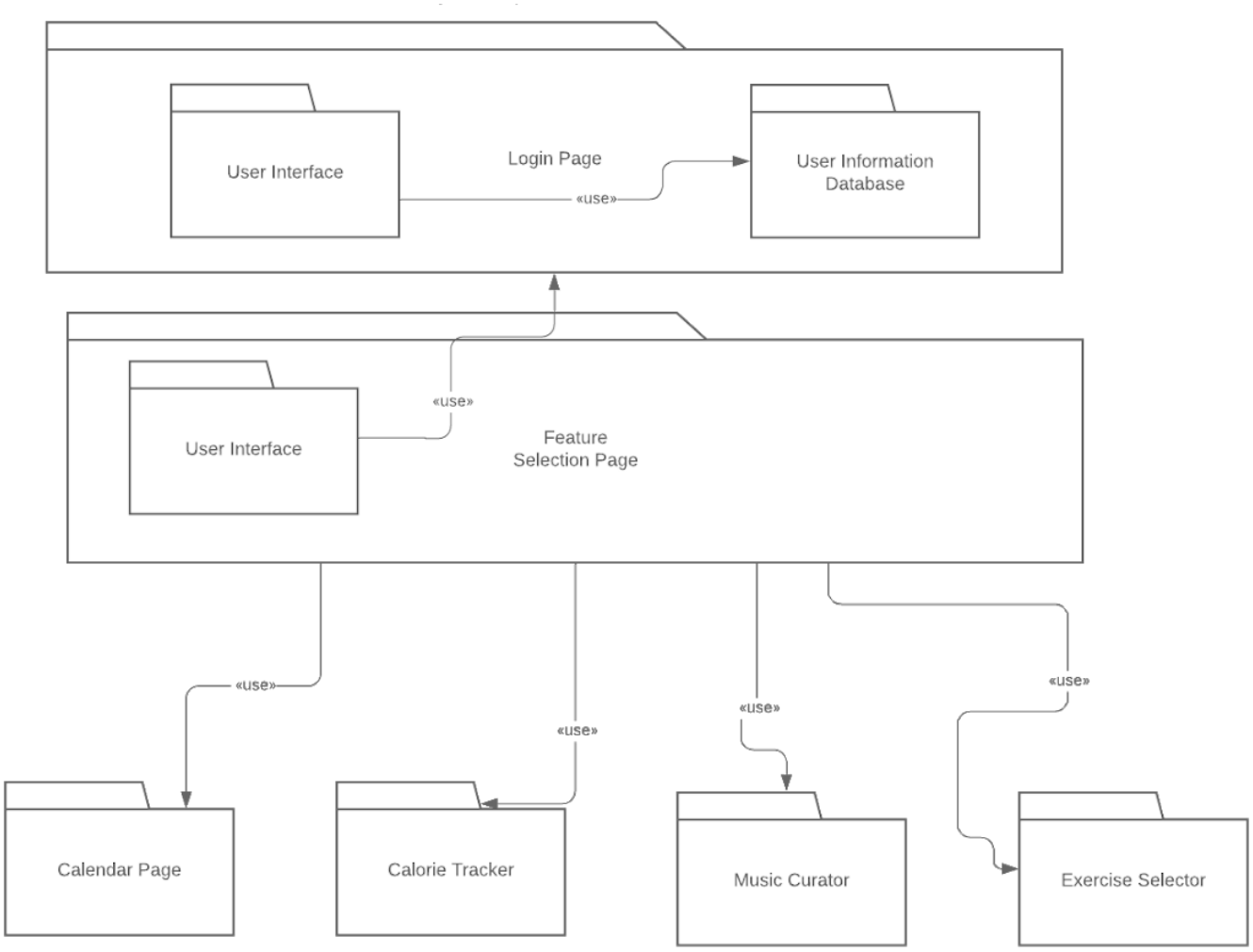
System Architecture and System Design

Architecture Styles

The architecture style of our system is largely component-based because the app is broken down into several logical subsections with mostly distinct functions. These subsections are our user options--i.e. the Calorie Tracker, Music Curator, etc. Instead of being dependent on one another, each subgroup has operations and attributes unique from other subgroups. This decreases the likelihood that bugs in one area will affect the entire code. In addition, since this particular type of app offers many services, separating the services streamlines the user experience. In effect, a user can select an option and only provide information needed for that subsection.

In addition, our system architecture is event-driven. Many sections of the app rely on user input and data. For example, the workout notification part relies on data from the user's Google Calendar, such that a change to the calendar changes the notifications. Consider also that the Calorie Tracker and Music Curator parts of the app rely on user inputted workout data and the user's heart rate. In this way, the app is very sensitive to user action.

Identifying Subsystems



Mapping Subsystems to Hardware

Our system does not need to run on multiple computers.

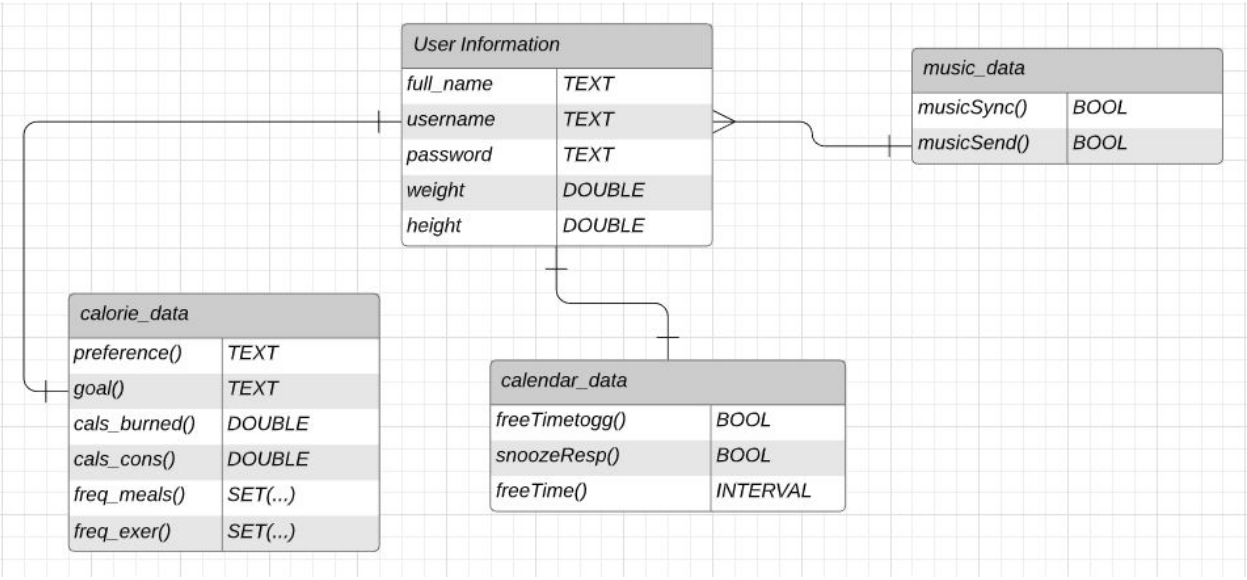
Persistent Data Storage

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.

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

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:

Feature	Requirements
Calendar Sync	Watch screen display: ● size 34 mm or higher Android Display Server communication: Android→ watch Server communication access → Google calendar Google Calendar account information sent via API integration (Google Calendar app is needed) Network connection with minimum network bandwidth
Calorie Tracker	Watch screen display ● size 34 mm or higher Android Display Calorie SQL database Network connection with minimum network bandwidth
Music Curator	Watch screen display ● size 34 mm or higher Android Display Server communication→ Spotify Network connection with minimum network bandwidth (if premium option not selected)
Exercise Selector	Watch screen display ● size 34 mm or higher Android Display

Algorithms and Data Structures

Algorithms

The calendar feature does use an algorithm. We use 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.

The calorie tracker feature does not use any complex algorithms.
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.

Data Structures

The calendar feature does not use any complex data structures.
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.
The calorie tracker feature stores the users most frequent exercises and meals using linked lists. Since the user can add data dynamically, we will use a linked list because the size of the list is unknown at compile time. However, the performance of linked lists is a drawback, but the flexibility is more important for our feature.

User Interface Design and Implementation

Calendar Sync

Only one modification has been made to the initial screen mock-up for the calendar page. 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.

Calorie Tracker;

For the Calorie Tracker section, we decided to create 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:

The modification that we decided to make was to include Spotify as a shell within our app and not make use of the Spotify app on the user's phone. This way the connection can be established through our application and the curator can work as it was intended to. The Spotify app integrated with our application will have features that the default app does not have. These features will be added by means of the android SDK which is a package included in the developer API provided by Spotify. For example, Spotify typically does not send push notifications to users with regards to the playlist, but the Spotify within our application, fitness friend, will notify the user each time a new playlist is generated, the reason being to increase the overall motivation aspect leading to a healthier lifestyle.

The Exercise Selector portion of the Music Curator app was modified to include separation of common workouts in subsections followed by division of those subsections into subsections, along with time and pace options. I.e. The subsections include cardio, muscle training, and yoga. The sections within those subsections are as follows: i.e. for cardio, when clicked the subsections include jogging, walking, elliptical, running, jumping jacks, etc, with additional options to include the number of reps or length. The end of exercise selection for one exercise gives the user the option to add another or finish.

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:

- There is enough free time during the free-time interval specified.
- There is no free time available during the entire day.
- There is enough free time but it is **not** during the free-time interval specified.
- There are numerous free time slots during the free-time interval specified.
- 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 consuming a meal and can head to the adding calories screen and search for the foods they ate. The calories corresponding to each food will be added to the overall calories.

3. The user checks their 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 as the first demo. For this we will be using the developer API- Android SDK via the following steps:

- First an application is registered with the spotify dashboard which provides us with a client ID as well as URI. So that the app relaunches every time the user logs in.
- The Android SDK is then used to create a link between the application server and the Spotify database.
- If successful, the user should be able to log into spotify via the application we have created. In turn, giving our API access to their account.

In addition to the Spotify API connection, the matching component of the music curator which matches user-intended exercises to a song matching pace and style preference will have to be extensively tested for accuracy using test cases. An ideal scenario for the use of this algorithm is:

1. The user, John, is prompted to enter exercises. Using the exercise selector subsection of the music curator, John enters that he wants to do 15 minutes of medium paces jogging, followed by 20 pushups, 100 sit-ups, and 10 minutes of walking to cool down.
2. The curator returns a Spotify playlist with 15 minutes of music with a high BPM of around 150, then lower BPM music fit for anaerobic exercise for about 3 minutes (estimated using the algorithm), then 10 minutes of calm BPM music around 50 BPM for John's cool down.
3. If John enters this same workout again when reusing the music curator, step 2 should be carried out with a slightly different list of songs (via randomization).

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.

Project Management and Plan of Work

Merging the Contributions from Individual Team Members

The largest issue that was faced was the lack of knowledge in creating an app. In addition to the general creation, many of us were not familiar with the languages, such as Javascript, needed to do so. After much research and outside advice, we decided to use snack.expo to create our app's interface. Fortunately for us, snack.expo provides a lot of well-explained documentation as well as examples that have allowed us to be successful in the creation of our app.

In addition to this issue, the group also faced a smaller issue midway through the first half of the semester. During the initial creation of our group, four subgroups were created; three groups of two and one group of three. After multiple reports and team meetings, it was determined that the best direction for our project to follow would be to split one of the subgroups containing two people into the two other subgroups also containing two people. The subgroup that was initially working on the SOS feature was the group that was split up due to their feature having already been implemented on the Apple watch.

A final issue that our group is dealing with a new change in operation. Our weekly meetings were removed due to the immediate evacuation of Rutgers. Though we didn't have a Project Manager, we did agree on a time - as a group - where there was at least one representative per subgroup. This meeting allowed for each group to get a look into what the other groups' statuses are. Currently, each subgroup has been communicating utilizing their preferred method [video conferencing, group messaging, etc.]. We plan to overcome this issue by utilizing When2Meet to find a time where the entire group can work on the project together. We have found that working on the project simultaneously allows questions to be answered regarding shared interfaces and ensures that each group

member is being held responsible for providing some contribution to the overall project.

Project Coordination and Progress Report

What use cases have been implemented?

For the music curator, we are in process of finishing up linking the app with Spotify; as this is one of our integral/base features, upon which we will code/add some advanced features using SDKs. For the exercise selector, adding the massive list of possible exercises is in the process of being implemented.

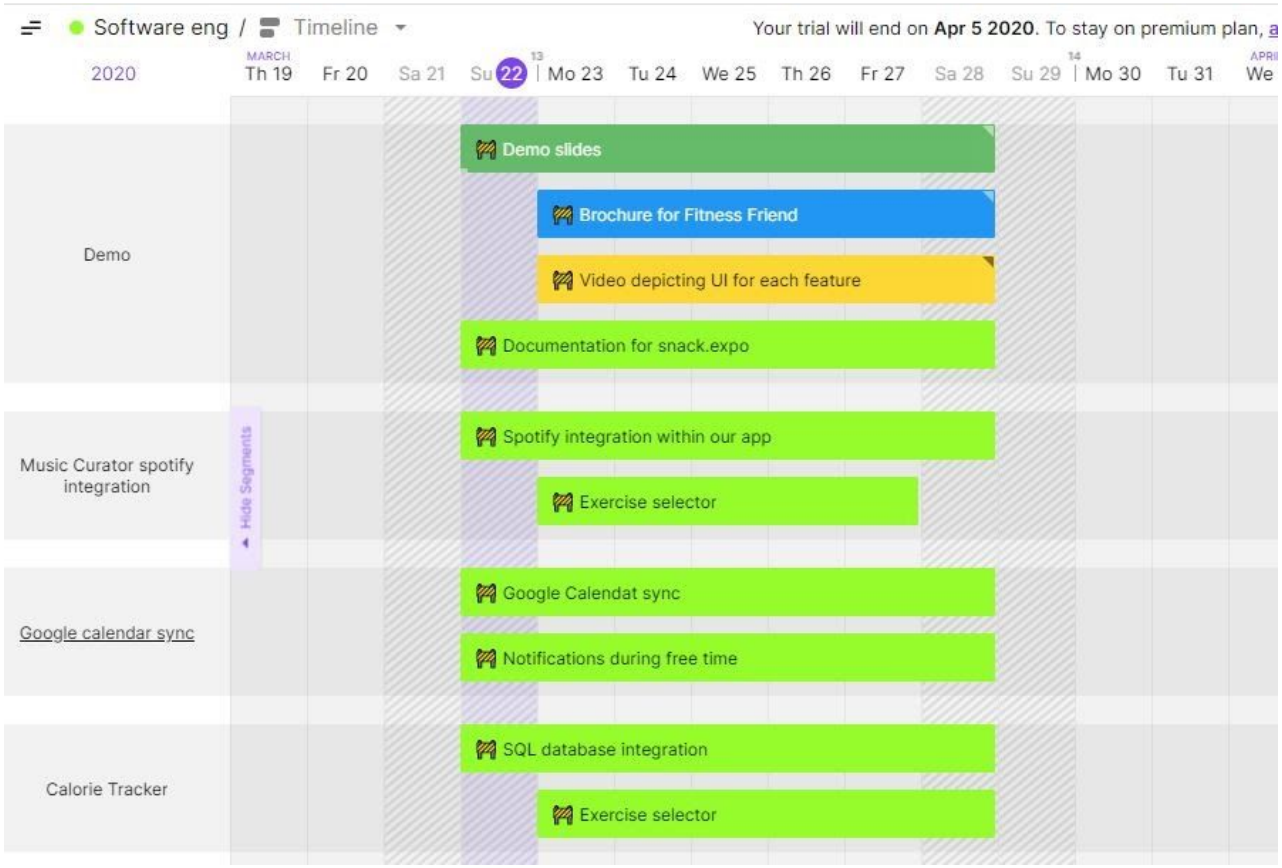
What is already functional, what is currently being tackled?

The calendar sync group has been successful at the creation of their three pages with working buttons and dropdown menu options. Upon the creation of an account database that will be able to store the user's information and preferences, the pages will be fully completed. The database that will be used to create user's accounts, as well as pull information (for preferences and settings and to log in), is still currently being tackled by the group as a whole.

For the music curator, the connection with Spotify is nearing completion, whereas the algorithm linking exercises to playlist curation, another core feature, is still being immensely researched and drawn out. This involves immense research into exercise pace and how exercise pace varies depending on age and health level. This algorithm needs to be very accurate in order to be considered successful. Else, the music curator will not be very customisable. Research is also being done on BPM in songs in relation to exercise and being able to reach a target heart rate.

The calorie tracker has made the interface of the screens uniform and are working on putting the database together. From there, we will connect the database to the backend of the app. The numbers returned from the database will be used in our calculation algorithm. The calculation algorithm will allow for the overall calories to be displayed on the screen for the user's convenience.

Plan of Work



Breakdown of Responsibilities

Subgroup 1: Tiyon King, Jenna Krause, Andrew Rezk

Tiyon King: Google Calendar API Integration

- Adding events
- Snoozing exercise notifications
- Storing the user's response
- Toggling free time notifications

Jenna Krause: Interface and Push Notifications

- Calendar, User Settings, Login Pages
- Notification integration of all app's features together
- Setting free time interval and workout time
- Notifying free time

Andrew Rezk: Google Calendar API Integration

- Calculating free time

Creating test Google Calendar accounts
Syncing Google Calendar

Subgroup 2: Mya Odrick, Maria Rios, Hedaya Walter

Calorie Tracker
HomeScreen Interface
Mya Odrick
Food Input Interface
Calculating amount of calories from food
Maria Rios
Calorie Tracker Main Interface
Formulating the ratio of calories in and out
Hedaya Walter
Exercise Input Interface
Calculating amount of calories from exercises

Subgroup 3: Amber Haynes, Shivani Sunil, Devvrat Patel

Music Curator
Shivani Sunil: Spotify API integration
Exercise Matching
Amber Haynes: Exercise Selector and Matching
Spotify Integration

Integration will be coordinated by the developers of the app which include all the members from each subgroup, with each subgroup representing a feature of the fitness friend. Each sub aspect of the feature will be tested by the individual it was assigned to. This will ensure that each of the subfeatures of each main feature work accordingly. In other words, unit testing as well as the documentation for it will be done individually and the pieces will then be put together to create a fully functional and integrable feature.

After the unit testing has been successfully completed, each subgroup will select one member who will coordinate the integration testing of the feature, as a whole. This will ensure clarity among each group and also contribute to the overall efficiency. The fact that we are using snack.expo will make it easier to come up with test drivers and test stubs. However, the integration testing will also be done by the testers.br

References

1. "Software Engineering book", Ivan Marsic
   http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf
2. “SQL Data Types for MySQL, SQL Server, and MS Access”
   https://www.w3schools.com/sql/sql_datatypes.asp
3. Spotify Android SDK
   Android SDK
4. Snack expo app development environment
   Push Notifications