Fitness Friend
Technical Documentation

Group 6

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

March 31, 2020 Software Engineering

Table of Contents

App Overview

Fitness Friend is an app in production designed to currently work for Android devices. The foundation of this app is made up of its three key user features: Calendar Sync, Calorie Tracking, and personalized Music Curation. Calendar Sync is a feature unique to Fitness Friend in that it syncs the user's Google Calendar via a Google Calendar API, and sends notifications to workout only at times wherein the user has free time. The user is able to set further restrictions on these notifications via the app's extensive yet easy-to-use user interface. Calorie Tracking is a feature that takes inputs from the user of food eaten daily and exercises completed, and, using mathematical algorithms, a food Calorie API, and the user's personal health information, aims to return a clear and easy-to-read page summarizing calories burned and consumed. The user inputs can range from type of food to the UPC barcode of the food consumed and the exercises are inputted in plain English, and the app returns Calorie statistics. The final feature, the Music Curator, is designed to provide the user with a Workout Playlist where the BPMs of the songs and timing match a user-inputted desired workout and user-inputted genre/artist preferences. Developing this feature required a very involved UI to receive inputted exercise and music information from the user, and using a separate algorithm along with a Spotify API and SDK in order to develop a playlist that would follow the user's music taste and workout pace. The following documentation provides a more in depth breakdown of the underlying UI and algorithms of each core feature of the app.

Design/UX

Fitness Friend's User Experience (UX) design was largely focused on ease of usability and a logical user experience based on ideal user scenarios which were based on our app's goal. The app's subdivision of features allowed for the base design of a login page followed by a home page with three pathways leading to the three features, and a fourth for user settings. This offers a user with any intention streamlined access to the feature they desire, i.e. the user does not have to endure an unwanted feature to reach a feature they want to use. User Experience design also influenced the design of pages for individual features. For the Calendar Sync in particular, every setting to adjust notification times and frequencies are displayed in a easily-readable single page, and can be edited on the same page. For the Calorie Tracker, the user is able to search for food using their keyboard and the user can see a picture of the food for reference to confirm, and alternatively the option to scan the UPC barcode to receive Caloric information vastly reduces work required of the user, and expedites food input. Similarly with exercise input for the Calorie Tracker, workout input is simple and on a single page. Calories for both food and exercise are calculated using an algorithm away from the user's sight using information from user settings so that there is a simple, streamlined result with no effort. The Music Curator currently uses several different pathways for exercise input due to the extreme detail needed to have detailed exercise information. Despite the exercise input being spread across different pages, the input still follows a logical and visually easy format in that a comprehensive list of exercises is not shown immediately. Instead, the user is shown three key types of exercises: Aerobic, Muscle Training, and Yoga, and a screen press leads them to a page with more detailed exercises for their chosen type, followed by a prompt asking for timing and reps. Preferred music genre is asked before the user is able to confirm that the app can proceed with generating the playlist. Similarly to the other features, the algorithm which matches the inputted exercises and genre occurs behind the scenes, increasing simplicity. The user is prompted to log into Spotify only the first time they log in (unless their account is deleted), and then once the app generates the playlist it is displayed. Each individual feature of Fitness Friend follows a specific UX design based on an experience which will fit a variety of individuals with a variety of possible interests and needs.

User Interface Information

For the User Interface, we wanted to keep the layout clean and simple. As with the overall User Experience (UX), we wanted to make sure that the user found navigating between the pages straightforward. Our app is divided into 3 main features that the user can customize based on what their needs are. The Calendar feature uses the user's schedule,which is obtained through Google Calendar, to find a suitable time for their workouts to take place. By using Google Calendar, the user can more efficiently interact with the app as they will already be familiar with the environment. The Calorie Tracker feature lets the user input their own calorie intake goals as they see appropriate and can then keep track of how they are doing. From the Calorie Homescreen, the user can navigate to either the Food Input page or the Exercise Input page. The UI is command-line for the Food Input page while the UI on the Exercise Input page is menu driven. The Music Curator feature allows users to create their own playlists for their workouts to make the experience enjoyable. This can conveniently be done by syncing their account to Spotify or choosing to make their own. The playlists can be assigned with the chosen workouts. We made it easy for the user to navigate between the features and ensured that the user wouldn't be distracted by any unnecessary material on the screen. Overall, our goal is to make the app easy to use and make the user feel more motivated to continue their fitness journey by having the app adjust to their needs/goals.

Algorithm Implementation

Fitness Friend has many algorithms that are made in conjunction with an API (discussed further in section: API/SDK integration). For its Music Curator feature, however, there is an algorithm still in remedial development which will serve as an intermediary algorithm connecting the user input to the Spotify packages. This algorithm will convert data about the user-inputted workouts and genres to stats for songs for the application to select from Spotify's extensive collection. Because careful song selection based on the pace and time length of each exercise, and the user's desired genres and artists is a vital part of the Music Curator, this algorithm is very complex with many conditional cases and, in addition, requires much ongoing research pertaining to BPM in songs in association with working out. For that reason, it is still in production, and once its base version is made it will require much testing and editing. As the makers of Fitness Friend we are immensely aware that components of our app attempt to cater to the user's overall fitness experience in ways different from normal music or normal fitness apps, and thus many algorithms and revisions are needed. In the case of the music curator, there are no commercially available apps which attempt to match songs directly to a user's exercise, and for that reason, the algorithm development starts from a novel point of reference, and must reach a high place of accuracy. Thus, the algorithm is still in development.

Navigation & Integration

Initially, the different screens were created separately to allow for all of the screens to be worked on easily by their corresponding groups. Once all of the screens are created and all of their features are working as expected, the screens will be mapped together. The ability to switch between pages was implemented by importing createStackNavigator from react-navigation. This allows us to create a "RootStack" that will be used to organize and label the various pages. To make the code easier to read and edit, there will be one main javascript file that will be used to initialize the different pages and what they will be named as well as the initial page. The separate pages will be imported from a screens file that will hold the various codes created by each group. By importing the files instead of compiling all of the codes for each page into one file, we are able to minimize the chance of compilation errors as well as the complexity of our code.

Once the user opens the app, they will be taken to our login/sign up page where they will be able to log into their pre-existing account or create an account if they do not already have one. The user will be prompted with this screen only if they have not yet logged into their account or if they logged out of their account via the settings page. If the user is currently signed into an account, the first screen they will see once they open the app will be the homepage where they will be able to navigate to the different screens. Once the RootStack has been created, the buttons on each page will be set up so that each button will direct the user to the correct page and the "Back" button will navigate back to the previous screen.

Mathematical Models

In order to accurately calculate the amount of calories burned by the user during a certain exercise, we used the user's inputted weight and gender to personalize these calculations as much as possible. The user will input this information when they initially create an account and can edit their selections at a later time in Settings if they choose. The types of exercise were broken down into three categories based on our source[1]--moderate, strenuous, and very strenuous:

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

To calculate the amount of calories burned, the code takes the user's weight and divides it by 140 if the user is female and 175 if the user is male. It then multiplies that by both the number of minutes the user inputted and the calories burned per minute.

<u>Equations</u>

Male user:

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

Female user:

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

API Integration

There are several APIs in our application. Several APIs are already present and then there are a few that are still currently being worked in and are in the testing phases. Please find an expanded list - with descriptions below - below.

Database API

This is currently being integrated with the Login Page User Interface. The API itself is still being chosen between Firebase and MySQL. This database will be able to store the users' information to allow them to successfully login with their email and password on any device. It will be a realtime database that will store the user's account information as well as their unique device ID's and information. In addition to storing the user's account credentials, this database will also be a way

[1] https://www.onhealth.com/content/1/calories_burned_during_fitness

to retrieve the user's unique token ID needed in order to be able to send push notifications to the user's device. The database will not be used for storage of certain user preferences such as preferences specifically pertaining to any feature such as the calendar or calorie tracker. Those preferences will be stored by the app itself as variables within the code that retrieve and store the user's input from the buttons and dropdown menus on each corresponding page.

Edamam Food Input API

When the user searches for a food item, we use the Edamam API system to return the calorie information about the food. Edamam provides an open source database with over 700K food items, and we use their Food Database Look-up API to request information from it. There's two options for our food search: food name, and UPC code. If the user decides to search with the food name, we send a request to the Edamam database which returns the most relevant results. At the moment, our program only uses the top result, but we plan to allow the user to decide which result they want to use. The same process applies if the user types in a UPC code. We also plan to allow the user to scan barcodes as well to obtain the UPC code.

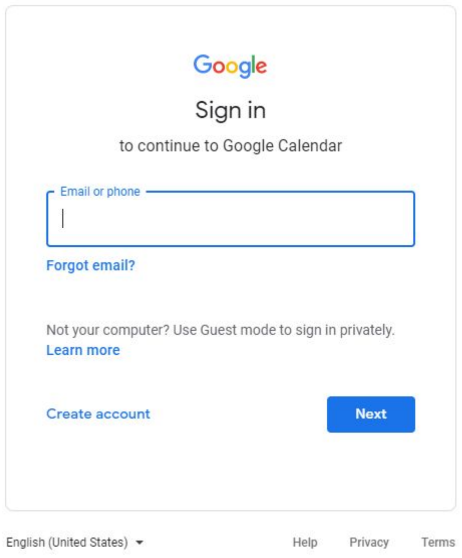
Along with determining the amount of calories in food, we will use the Edamam API to help recommend food options for the user. The API has an option to filter by number of calories, diet, health, and allergies. When we are able to implement a database, we will have this information from the user when we integrate our recommendation feature to ensure they receive relevant results.

Google Authentication API

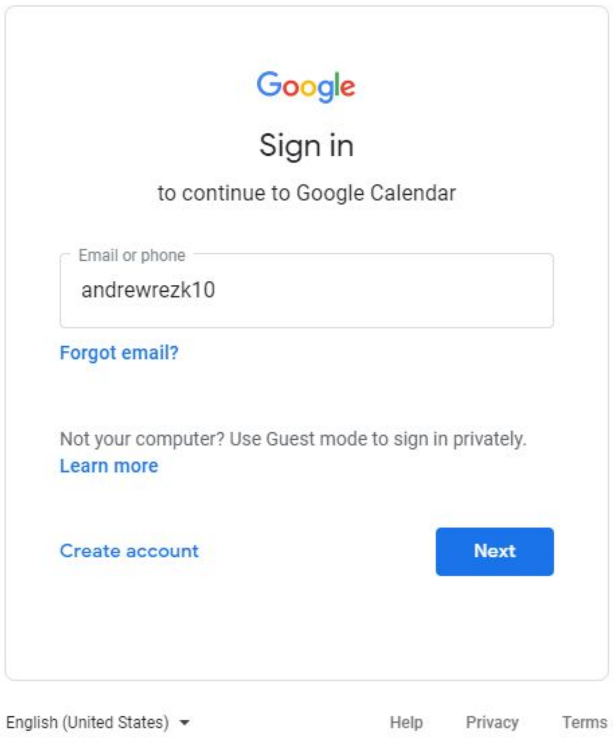
The Google Authentication API allows for the user to successfully authenticate themselves as a Google User. This API is still being integrated into the Calendar User Interface. It will allow for the user to login, with information that may already possibly be stored on their phone, and then take the user directly back to the application. Currently, the Google Client ID remains in the code and the actual code for the API is being worked on. Google typically works with Node.js but there are ways to allow for the user to be logged in into Google without doing so. This remains a work in progress. As per the Google Authentication integration, the below steps/directions are what the user is going to do to allow the app to have a valid token to pull the calendar info from google servers.

<u>To sign in to Google Calendar (still in progress, was used to integrate calendar info for the demo):</u>

1. To sync your Google Calendar, sign into the Google login prompt:

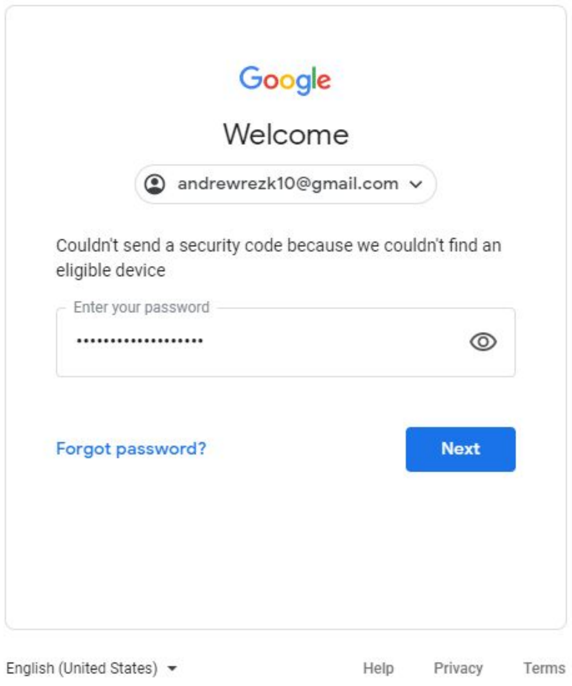


2. Enter your email or phone in prompt (as per example):



Email or phone: andrewrezk10

3. Enter your password in prompt (as per example):


Google

Welcome

andrewrezk10@gmail.com

Couldn't send a security code because we couldn't find an eligible device

Enter your password

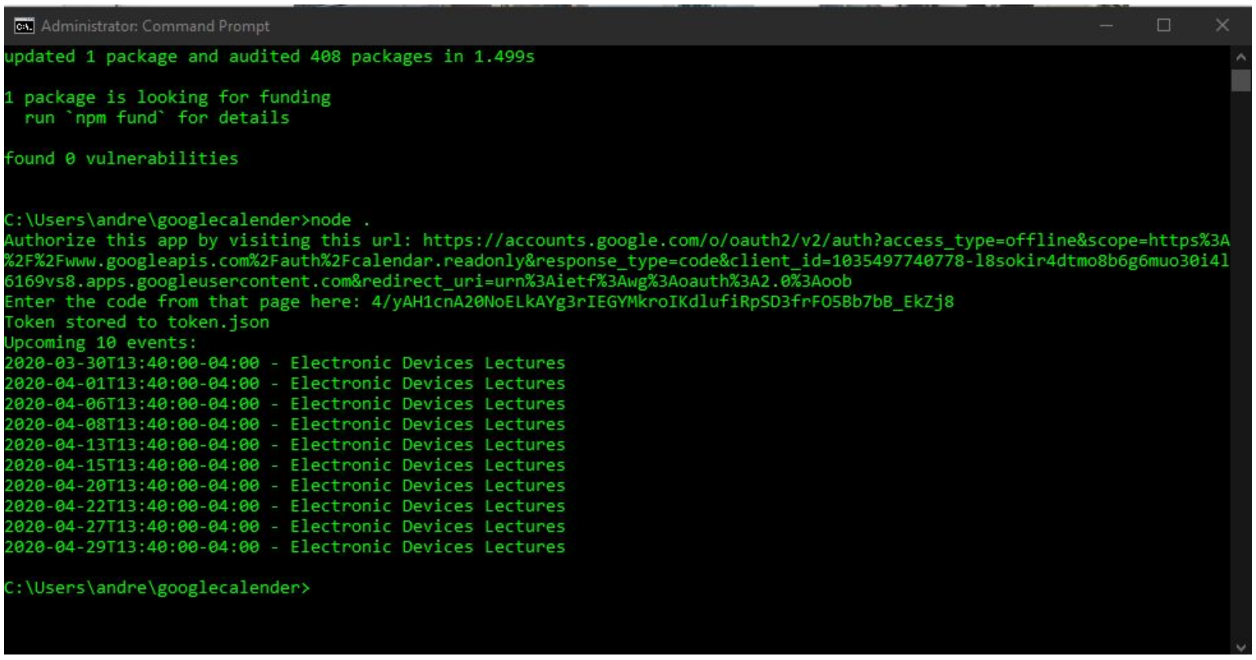
••••••••••••••••••••

4. (Still in the works) The app should send you a notification saying "Google Calendar Synced"
5. (Still in the works) The app should have all the time slots that you are free that week, and display them in the Calender part of the app.

Google Calendar API

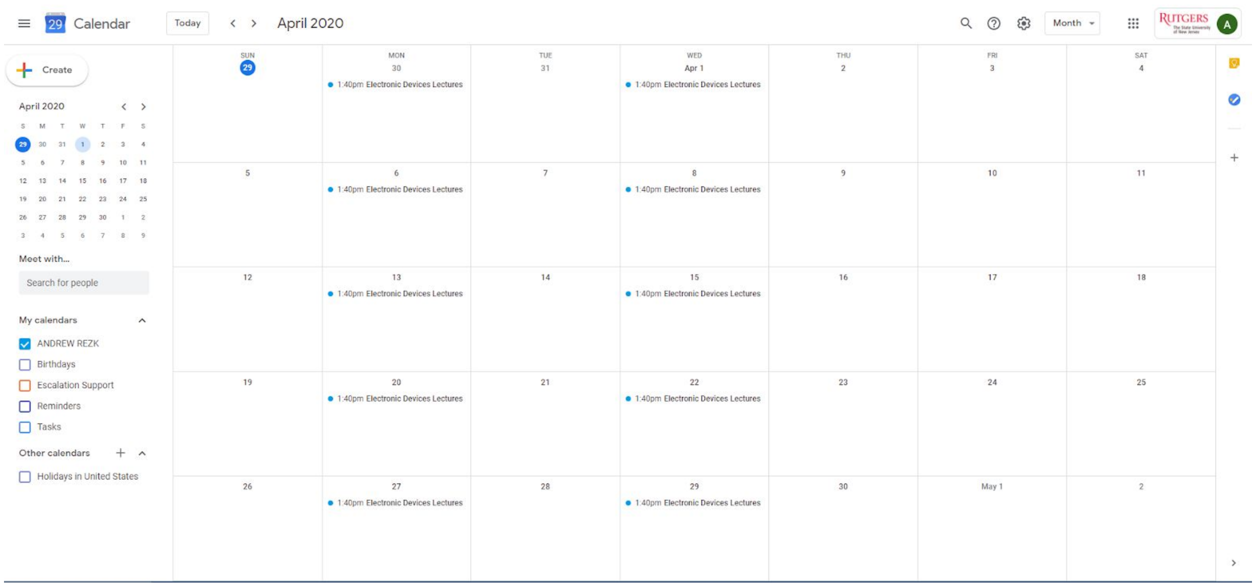
The Google Calendar API is currently being worked on upon a local server. Google Calendar's API requires Node.js so it cannot be integrated on Snack Expo. The group is currently working on the remainder of the user interface. From here, we will find a different form of combining our API code with the remainder of our application. How it works is that we take the token from the authorized login from google and then pull the information from google servers. Once the users login is authenticated the code pulls the events from the user's calendar. It then displays that, and it is currently set up to display 10 events. However, the code is still in progress, the end goal is for the app to display free time block in between the users events.

Here running the code, you see that my example user's 10 next events were repeated lectures on different days and times:

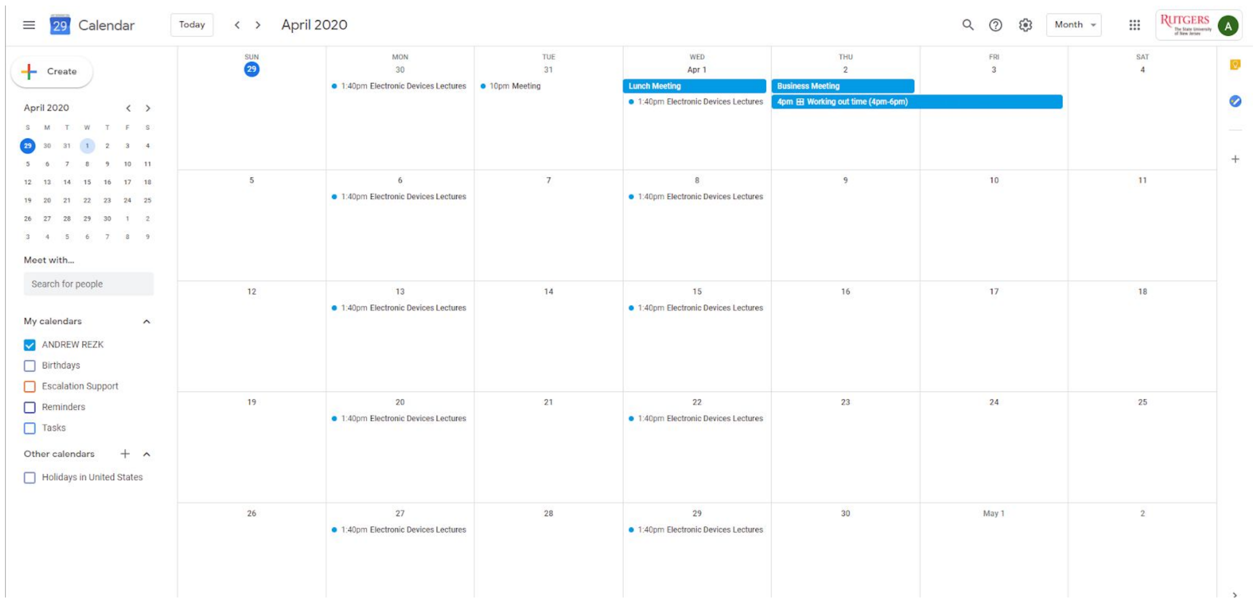


```
Administrator: Command Prompt
updated 1 package and audited 408 packages in 1.499s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities


C:\Users\andre\googlecalender>node .
Authorize this app by visiting this url: https://accounts.google.com/o/oauth2/v2/auth?access_type=offline&scope=https%3A
%2F%2Fwww.googleapis.com%2Fauth%2Fcalendar.readonly&response_type=code&client_id=1035497740778-l8sokir4dtmo8b6g6muo30i4l
6169vs8.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob
Enter the code from that page here: 4/yAH1cnA20NoELkAYg3rIEGYMkroIKdlufiRpSD3frFO5Bb7bB_EkZj8
Token stored to token.json
Upcoming 10 events:
2020-03-30T13:40:00-04:00 - Electronic Devices Lectures
2020-04-01T13:40:00-04:00 - Electronic Devices Lectures
2020-04-06T13:40:00-04:00 - Electronic Devices Lectures
2020-04-08T13:40:00-04:00 - Electronic Devices Lectures
2020-04-13T13:40:00-04:00 - Electronic Devices Lectures
2020-04-15T13:40:00-04:00 - Electronic Devices Lectures
2020-04-20T13:40:00-04:00 - Electronic Devices Lectures
2020-04-22T13:40:00-04:00 - Electronic Devices Lectures
2020-04-27T13:40:00-04:00 - Electronic Devices Lectures
2020-04-29T13:40:00-04:00 - Electronic Devices Lectures

C:\Users\andre\googlecalender>
```

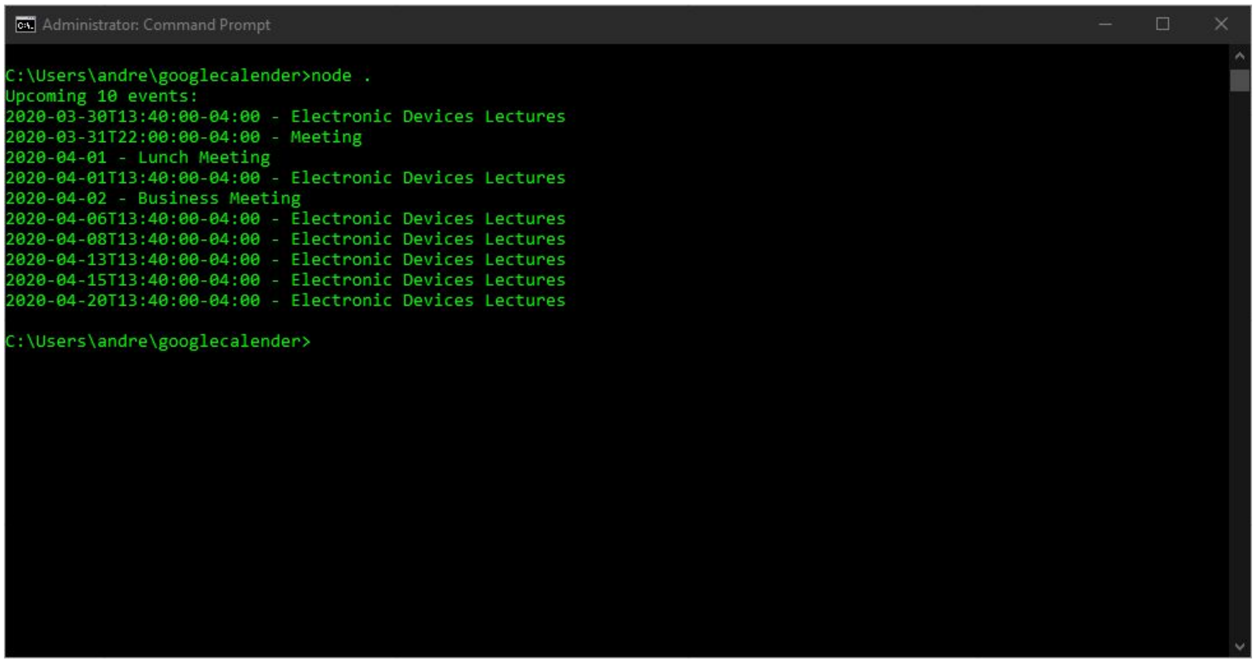
And here is the view of the Google Calendar itself:



After tweaking the calendar by deleting and adding events we get this view of the calender:



And by running the code, you see that the next 10 events change to the following list:



```
C:\Users\andre\googlecalender>node .
Upcoming 10 events:
2020-03-30T13:40:00-04:00 - Electronic Devices Lectures
2020-03-31T22:00:00-04:00 - Meeting
2020-04-01 - Lunch Meeting
2020-04-01T13:40:00-04:00 - Electronic Devices Lectures
2020-04-02 - Business Meeting
2020-04-06T13:40:00-04:00 - Electronic Devices Lectures
2020-04-08T13:40:00-04:00 - Electronic Devices Lectures
2020-04-13T13:40:00-04:00 - Electronic Devices Lectures
2020-04-15T13:40:00-04:00 - Electronic Devices Lectures
2020-04-20T13:40:00-04:00 - Electronic Devices Lectures

C:\Users\andre\googlecalender>
```

The goal of the integration however is instead of displaying events, we change the output to the blocks available that day. So by the 10 results it should list out the blocks of time free of events to workout. So it would then result into a notification being set to the user about the free time from the app (push notification).

The code we will be using in the API integration is in the folder Google Calendar Sync.

Spotify API

The Spotify API works by connecting a snack expo URI to the Spotify developers dashboard. A client ID is created in the dashboard and certification and authentication packages are installed and registered on the dashboard. The certification ensures authentication and enables a secure connection. The SHA certificate package has been generated. This ensures communication between our system and the user. Two fingerprints are created, one for development and one for production and this has been done due to security reasons. Then an app remote SDK is added to the code under the mainactivity or function part of the app. Secondly, the Spotify app remote release model is imported through the Spotify SDK. Then the dependencies are imported and the credentials are set up. Lastly, the app remote SDK is used to connect to our application and user authorization is requested.

The rest of the integration is still being processed. Once the connection between our remote app and the Spotify SDK has been established then we will work on creating playlists based on BPM and music recommendations. The user will also have a choice to sync their Spotify account to our app and get personalized suggestions based on the workout they select and BPM's required for those kinds of workouts and lastly their artist and genre of choice.

The code we will be using in the API integration:

```
package com.yourdomain.yourapp;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

import android.util.Log;

import com.spotify.android.appremote.api.ConnectionParams;
import com.spotify.android.appremote.api.Connector;
import com.spotify.android.appremote.api.SpotifyAppRemote;

import com.spotify.protocol.client.Subscription;
import com.spotify.protocol.types.PlayerState;
import com.spotify.protocol.types.Track;

public class MainActivity extends AppCompatActivity {

    private static final String CLIENT_ID = "your_client_id";
    private static final String REDIRECT_URI =
"com.yourdomain.yourapp://callback";
    private SpotifyAppRemote mSpotifyAppRemote;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
}

    @Override
    protected void onStart() {
        super.onStart();
        ConnectionParams connectionParams =
                new ConnectionParams.Builder(CLIENT_ID)
                        .setRedirectUri(REDIRECT_URI)
                        .showAuthView(true)
                        .build();

        SpotifyAppRemote.connect(this, connectionParams,
                new Connector.ConnectionListener() {

                    public void onConnected(SpotifyAppRemote
spotifyAppRemote) {
                        mSpotifyAppRemote = spotifyAppRemote;
                        Log.d("MainActivity", "Connected!
Yay!");

                        // Now you can start interacting with
App Remote
                        connected();
                    }

                    public void onFailure(Throwable throwable) {
                        Log.e("MyActivity",
throwable.getMessage(), throwable);

                        // Something went wrong when attempting
to connect! Handle errors here
                    }
                });
    }

    @Override
    protected void onStop() {
        super.onStop();
        SpotifyAppRemote.disconnect(mSpotifyAppRemote);
    }

    private void connected() {
        // Play a playlist

mSpotifyAppRemote.getPlayerApi().play("spotify:playlist:37i9dQZF
1DX2sUQwD7tbmL");

        // Subscribe to PlayerState
        mSpotifyAppRemote.getPlayerApi()
                .subscribeToPlayerState()
                .setEventCallback(playerState -> {
                    final Track track = playerState.track;
                    if (track != null) {
                        Log.d("MainActivity", track.name + " by
" + track.artist.name);
                    }
                });
    }
}
```

User/Legal Rights

In the case that our app becomes available on the mobile application market, we will have to be very knowledgeable about mobile user laws and protections as well as have access to an experienced lawyer able to advise us in creating user privacy and terms of use agreements. While our app is still in development, we still must keep a log of the potential privacy and terms of use violations that our app frame and databases create. It is not possible for a lawyer with little to no computer experience to uncover every potential privacy violation our app creates. Fitness Friend has several databases with user information that must be thoroughly protected from cyber attacks and, if sold to advertisers, has terms that must be explained transparently to users. Fitness Friend stores user's emails and passwords which must be thoroughly protected, probably with the exception of email addresses used by the Fitness Friend team to confirm user email. The Calendar Sync option connects with the user's Google Account, which on the market means the Fitness Friend app must communicate with the Google API about permissions needed which Google communicates to the user. Fitness Friend would need these permissions to access the user's calendar without encountering legal dilemmas thereafter. In the case of the Calorie Tracker option along with settings, Fitness Friend requires that the user allows them to save their personal health information such as height, weight, and age, for the purpose of the Calorie Tracker algorithm. The Calorie Tracker must also make the user aware in Terms that their food inputs are generated into Calories using an API. This must be clearly communicated to the user. For the Music Curator, the user must be made clear that their exercises and genre preferences will be used in the Music Curator algorithm. The app must also communicate with the user that their Spotify account will be accessed through the Spotify API so that the app can access the user's account and display the generated playlists for the user. In this way, Google and Spotify permissions with Fitness Friend are similar. If Fitness Friend becomes public to the Android and/or Apple markets, various app permissions due to API and SDK use as well as permissions to use the user's information in the context of the app's features and algorithms require the app to create privacy and terms of use agreements.

Technical Troubleshooting

To provide the best support for our users, we will use the following steps as a guide to fix their technical issues:

1. Identify the problem.
    a. Pinpoint where the issue lies whether that be in one of the main features or in one of the general pages.
    b. Check if the problem is a software issue.

2. Establish a theory of probable cause.
    a. Follow the actions the user took that caused them to arrive at this problem.
    b. Come up with 1 or more possible causes based on the information provided by the user.
3. Test the theory to determine cause.
    a. Check if any of the possible causes was in fact causing the problem.
    b. If none, then go back to the previous step.
4. Establish a plan of action to resolve the problem and implement the solution.
5. Verify full system functionality.
6. Document findings, actions, and outcomes.
    a. Write in detail what the issue was and steps taken to resolve it.
    b. Mark any changes made to the system.