
Dynamic
Rhythm
A Spotify Playlist
Player and Visualizer
Chilling Nostalgic Hostages

Goals

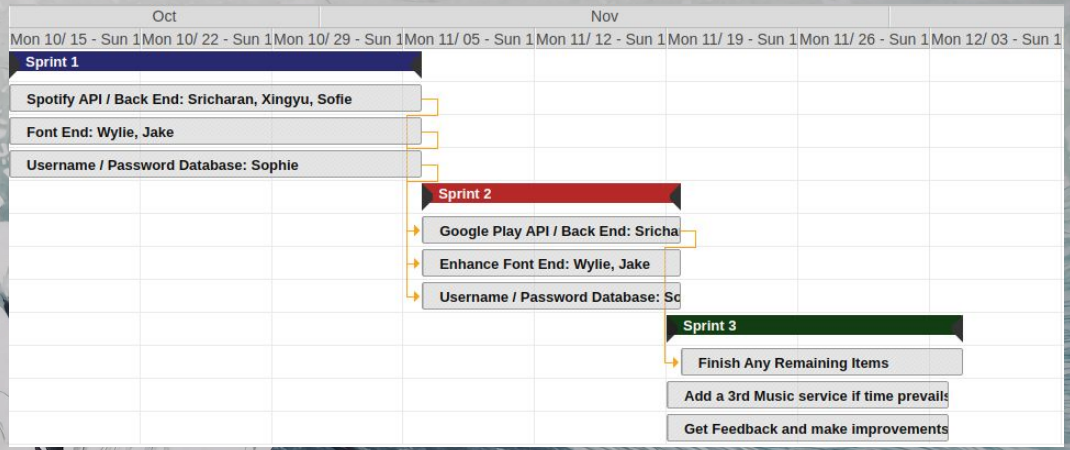
- Access your Spotify account information
  - User name
  - Access tokens
  - Personal playlist information
    - Song information in each playlist
      - Image, track title, etc...
- Incorporate a music visualizer
  - WebGL Visualizer
- Use NodeJS and HTML to create a webpage that will incorporate the goals


Project
Management

Development Tools

- Trello: 1/5: made, but unused
- Slack 5/5: communication
- GitHub 5/5: version control

Project Management Methods

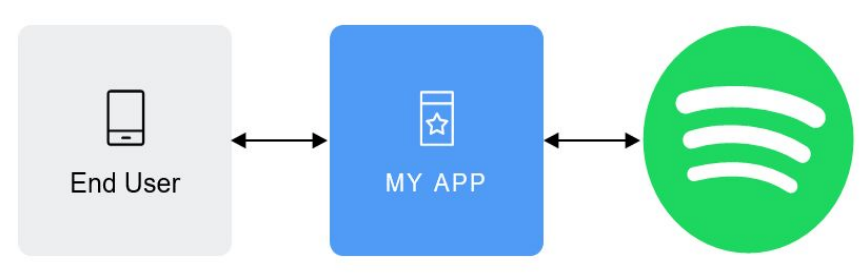


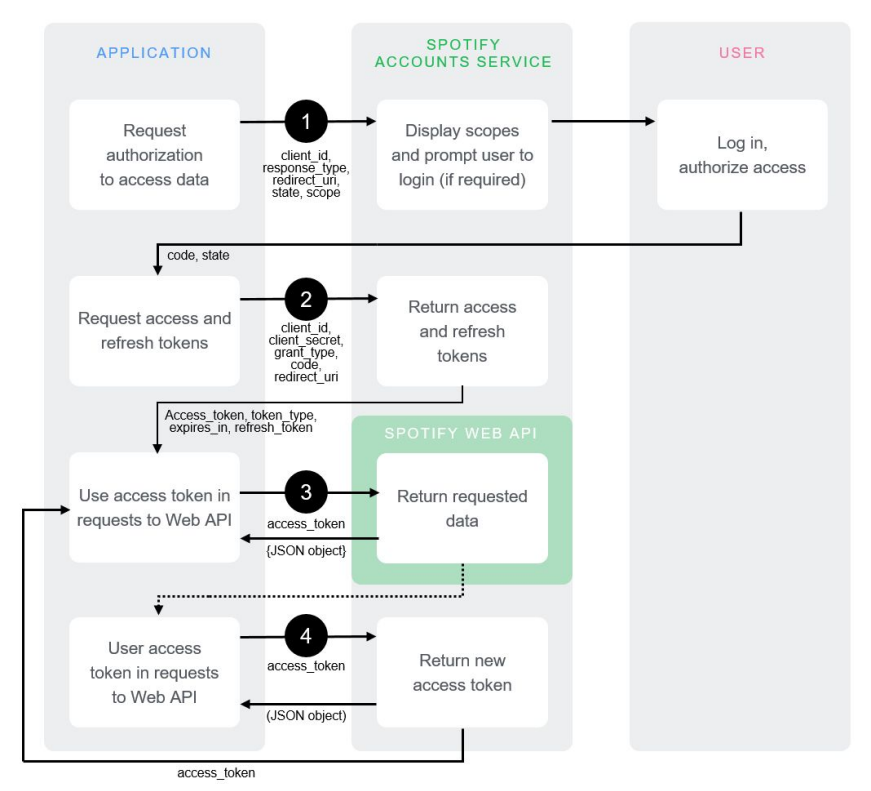
- Scrum 3/5
- Agile 2/5


Spotify API

Authorization Flows

- Security is of the utmost importance for using any API that access personal information
- We have used the following:
  - streaming user-modify-playback-state
  - user-read-private user-read-email
  - playlist-read-private
  - user-read-currently-playing





Authorization Examples

```
app.get("/login", function (req, res) {

    var state = generateRandomString(16);
    res.cookie(stateKey, state);

    // your application requests authorization
    var scope = ['streaming',
        'user-modify-playback-state',
        'user-read-private',
        'user-read-email',
        'playlist-read',
        'playlist-read-private',
        'playlist-modify',
        'playlist-modify-private',
    ]
    res.redirect('https://accounts.spotify.com/authorize?' +
        querystring.stringify({
            response_type: 'code',
            client_id: client_id,
            scope: scope,
            redirect_uri: redirect_uri,
            state: state
        }));
});
```

```
var express = require('express');
// Express web server framework
var request = require('request'); // "Request" library
var cors = require('cors');
var querystring = require('querystring');
var cookieParser = require('cookie-parser');
```

- The frameworks that were used (above) allow us to authenticate through Spotify
- Once you log in, we set a cookie so you can stay logged in
- Authorization flow was a challenge to understand and implement effectively

Fetching Playlists

- In order to utilize the visualizer we first needed to get access to the users Spotify music.
- Utilizing the access token we had received previously we were successfully able to obtain all of the users playlists.
- We implemented a function to change the player to the playlist when the user clicks on its title, and to render the list of tracks on the right side

```
let _token = access_token;
var playlistInfo;
$.ajax({
    type: "GET",
    url: "https://api.spotify.com/v1/me/playlists?limit=50",
    headers: {
        'Authorization': 'Bearer ' + access_token
    },
    beforeSend: function (xhr) {
        xhr.setRequestHeader('Authorization', 'Bearer ' + access_token);
    },
    success: function (data) {
        playlistInfo = data["items"];
        for (i = 0; i < playlistInfo.length; i++) {
            var p = document.getElementById("user_playlists");
            var playlist_name = playlistInfo[i].name;
            var playlist_uri = playlistInfo[i].uri;

            var new_button = "<button id = '" + playlist_uri +
"'class = 'btn btn-default' ondblclick= 'changePlaylist(this)' >" +
playlist_name + "</button>";
            $("#user_playlists").append(new_button);
        }
    }
});
```

```
function changePlaylist(elem) {
    console.log(elem);
    uri = elem.id;
    dataString = '{"context_uri":"' + uri + '","position_ms":0)';
    $.ajax({
        url: "https://api.spotify.com/v1/me/player/play?device_id=" + device
,
        type: "PUT",
        data: dataString,
        beforeSend: function (xhr) {
            xhr.setRequestHeader('Authorization', 'Bearer ' + access_token);
        },
        success: function (data) {
            console.log(data)
        }
    });
    id = uri.split(':');
    id = id[id.length - 1];
    console.log(id, 'ID');
    myurl = 'https://api.spotify.com/v1/playlists/' + id + '/tracks';
    console.log(myurl);
    $.ajax({
        type: "GET",
        url: myurl,
        headers: {
            'Authorization': 'Bearer ' + access_token
        },
        success: (result) => {
            console.log(result);
            var tracks = result["items"];
            console.log(tracks);
            current_tracks.innerHTML = "TRACKS";
            for (i = 0; i < tracks.length; i++) {
                var p = document.getElementById("current_tracks");
                var t_name = tracks[i].track.name;
                var t_uri = tracks[i].track.uri;
                console.log(t_name + " : " + t_uri);
                var new_button = "<button id = '" + t_uri +
"'class = 'btn btn-default'>" + t_name + "</button>";
                $("#current_tracks").append(new_button);
            }
        }
    });
}
```


Visualizer

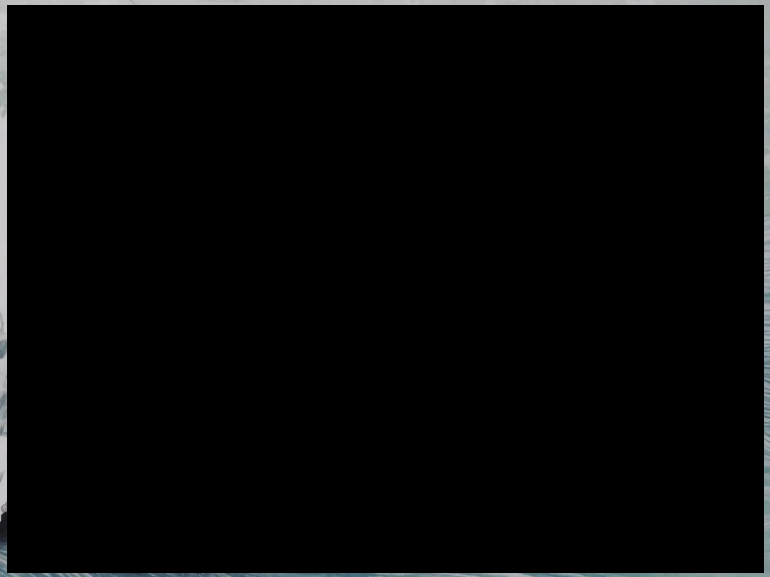
Predesigned Visualizer

- As interesting as it could have been, making a visualizer itself is its own *project*.
- Made by possan:
  - A visualizer that already uses currently playing tracks from Spotify
  - glMatrix, a nodeJS matrix library, and WebGL is used

- Topics covered in that file include:
  - Orthogonality
  - Transposes
  - Rotations
  - Linear Transformations
  - Scaling
  - Translations
  - Inverses
  - Inner Products

Predesigned Visualizer

- The visualizer takes an image and brakes it into many small triangles that float around and rotate in a virtual 3D space.
- The image breaks apart and rejoins together throughout the song, based on the beats which spotify provides through the track analysis




Database

PostgreSQL

- We did not need the database to be very complex
  - (in fact we don't *need* one at all to make the site function) 2/5
- Database is very simple, it holds:
  - User Id
  - User email
  - User name
  - User country
  - URI

```
var pgp = require('pg-promise')();

const dbConfig = {
    host: 'localhost',
    port: 5433,
    database: 'DynamicRhythm',
    user: 'postgres',
    password: 'Password'
};

var db = pgp(dbConfig);

module.exports = db;
```

```
create table
if not exists users
(
    id varchar
(40) NOT NULL PRIMARY KEY,
    email varchar
(40) NOT NULL,
    name varchar
(40) NOT NULL,
    country varchar
(20),
    URI varchar
(40)
);
```

```
function addNewUser(user) {

    db.any('SELECT count(*) from users where id = $1', [user.id])
        .then(data => {
            console.log(data),
                console.log(data[0].count);
            if (data[0].count >= 1) {
                console.log('User already in database')
            } else { //add user to database if they aren't already in there
                var query =
                    'INSERT INTO users(id, email, name, country, uri) VALUES($1, $2, $3, $4, $5)'
                db.one(query, [user.id, user.email, user.display_name, user.country, user.uri])
                    .then(data => {
                        console.log(data.id); // print new user id;
                    })
                    .catch(function (error) {
                        console.log('woopsie!');
                    })
            }
        })
        .catch(function (error) {
            console.log('error')
        })

}
```

```
[dynamic_rhythm=# \d users
                    Table "public.users"
 Column  |          Type          | Collation | Nullable | Default
---------+------------------------+-----------+----------+---------
 id      | character varying(40) |           | not null |
 email   | character varying(40) |           | not null |
 name    | character varying(40) |           | not null |
 country | character varying(20) |           |          |
 uri     | character varying(40) |           |          |
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)

dynamic_rhythm=#
```


Music Player
UI

Music Player - Web Playback SDK

- Creates a new player inside the browser
- Then, we can use API to control it
- We can play, pause, skip forward or backward, change the song
- Connected functions to perform requests with buttons on the page

Seek/Previous

- Uses a 'POST' request to the API

```
<div class="btn prev" onclick="sendCommand('POST', 'previous')"><i class=
"fa fa-step-backward" aria-hidden="true"></i></div>
<div class="btn play" id="playbtn"><i class="fa fa-play" aria-hidden="true"></i></div>
<div class="btn play"><i class="fa fa-pause" aria-hidden="true"></i></div>
<div class="btn next" onclick="sendCommand('POST', 'next')"><i class="fa fa-step-forward"
 aria-hidden="true"></i></div>
```

Play / Pause:

- Can use a 'PUT' request to the API to pause or play
- Also, Web Playback SDK has player.togglePlay() function

```
document.getElementById("playbtn").addEventListener("click", function () {
    //console.log(this);
    player.togglePlay();
});
```


Testing

Stress Testing and Normal Testing


```
{
    "country": "US",
    "display_name": "kawmaster",
    "email": "[redacted]",
    "external_urls": {
        "spotify": "https://open.spotify.com/user/kawmaster"
    },
    "followers": {
        "href": null,
        "total": 2
    },
    "href": "https://api.spotify.com/v1/users/kawmaster",
    "id": "kawmaster",
    "images": [{
        "height": null,
        "url":
"https://profile-images.scdn.co/images/userprofile/default/74198f751fb1ee6bf67a78f6f5f55cfb41b066fb",
        "width": null
    }],
    "product": "premium",
    "type": "user",
    "uri": "spotify:user:kawmaster"
}
```


- Stress Testing:
  - Sri's (only) playlist of 300+ songs
  - Spotify API can only get a max of 200 songs / playlist
- Norma Testing:
  - Looked at the JSON console output,
  - And made sure it matches our personal user information


Live Example
Thanks for listening!