Musilinky

A service that provides links to other music services at the push of a button

Project Description

- In 2023, music is everywhere and is an essential element of culture all around the world. Sites like Spotify, Apple Music, and YouTube have revolutionized what the meaning of shared music is and will continue to define and shape the future of music as we know it. Thus, our vision for this project was to create a website that allows anyone to search for custom music (similar to these common applications) and generate links to these songs for other sites. We believe this website will continue to spread the ripple effect of communal music and we are so excited to share our project with you.

Contributors

- Silas Khan
- Tyler Chung
- Eric Gosnell
- Alex Burch
- Cade Williams
- Patrick Fleming

Tools that we used (methodologies)

- Agile Methodology
  - **Purpose:** To complete the required tasks in the project in an organized fashion via user stories, epics, and team meetings.
  - **Rating:** ☆☆☆☆
  - **Reason:** The Agile methodology used throughout the project allowed us to manage the required tasks for a project in a clear and concise manner.
- Peer Programming
  - **Purpose:** To complete code for the project by working with one another in pairs.
  - **Rating:** ☆☆☆☆
  - **Reason:** This methodology allowed us to effectively resolve conflicts and doubts during in-person and remote meetings.

AGILE
1 Requirements
2 Design
3 Development
4 Testing
5 Deployment
6 Review

Tools that we used

- Project Tracker - GitHub Project Board
  - **Purpose:** Track the status of user stories and epics throughout the course of the project
  - **Rating:** ☆☆☆
  - **Reason:** The GitHub project board allowed us to organize the required tasks for the project in a systematic way.
- VCS Repository - GitHub
  - **Purpose:** Collaborate with others throughout the project
  - **Rating:** ☆☆☆
  - **Reason:** GitHub provides an effective way to work with others on software projects, but the nuances can be difficult for beginners.
- Database - PostgreSQL
  - **Purpose:** Stores user, song, and playlist information for various features in the project.
  - **Rating:** ☆☆☆☆
  - **Reason:** The PostgreSQL database provides an effective way to manage data. The errors are understandable as well.

Tools that we used

- IDE - Visual Studio Code
  - **Purpose:** The primary interactive development environment for creating the project codebase.
  - **Rating:** ☆☆☆☆☆
  - **Reason:** This IDE offers great integrated support for GitHub. It allows developers to resolve merge conflicts and access common Git commands through the UI.
- UI Tools - EJS and CSS
  - **Purpose:** Display visual information and styling to the pages that the users interact with
  - **Rating:** ☆☆☆
  - **Reason:** EJS syntax is intuitive and the CSS styling looks cool and is not too bad to manage. However, EJS errors are often hard to interpret.
- Application Server - NodeJS
  - **Purpose:** Houses the API endpoints that process the client requests (such as search and MUSALINK) on the server
  - **Rating:** ☆☆☆
  - **Reason:** Effective tool for implementing an application server, but can become extremely tedious to manage when more endpoints or code is added.

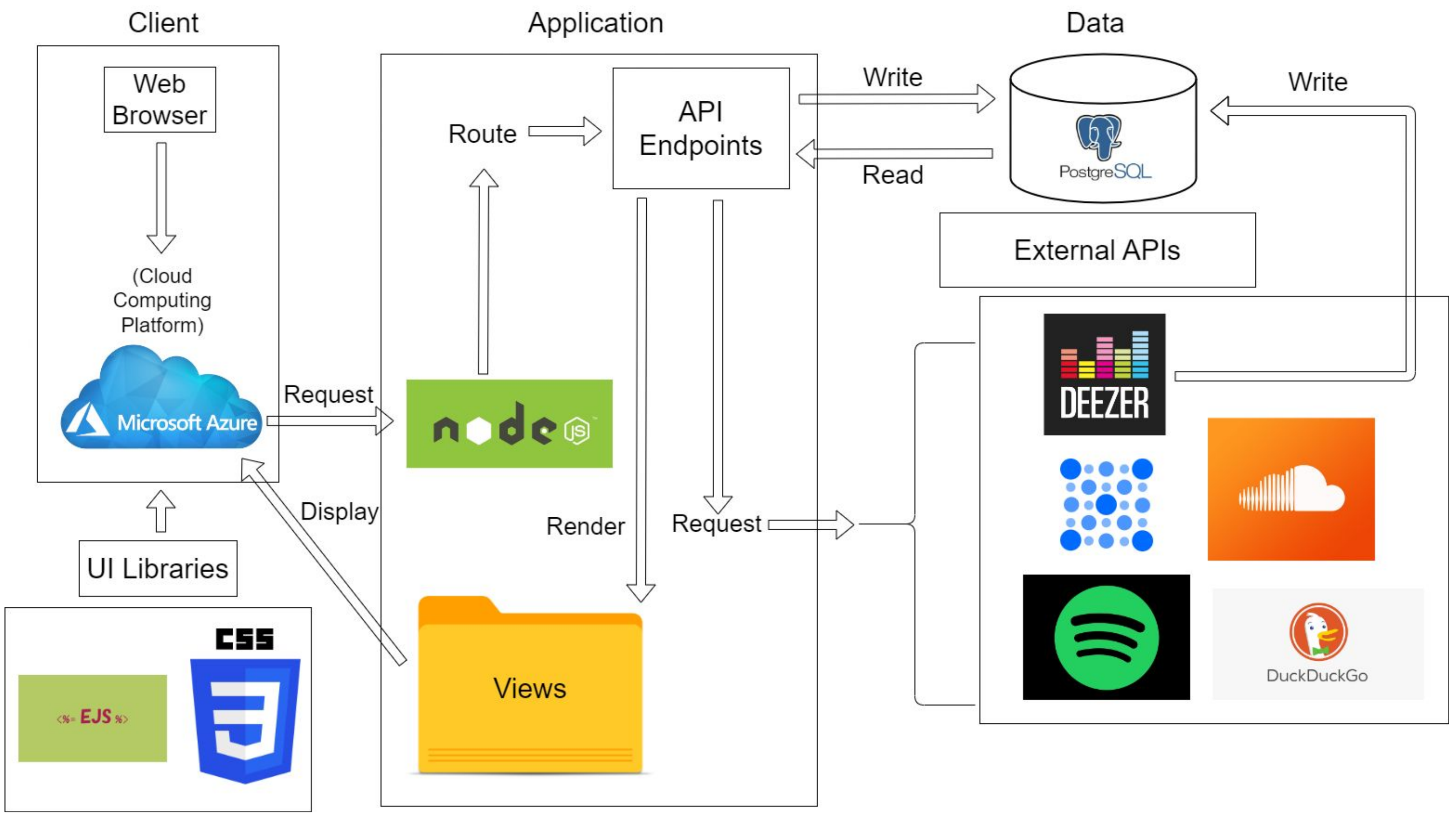
Tools that we used

- Deployment Environment - Microsoft Azure
  - **Purpose:** This is the cloud computing platform that hosts our deployed application.
  - **Rating:** ☆☆☆
  - **Reason:** Useful tool for deploying an app on the cloud, but the process of setting it up can be tedious.
- Testing Tool - Mocha and Chai
  - **Purpose:** Allowed us to write unit test cases for the API endpoints in the project
  - **Rating:** ☆☆☆
  - **Reason:** Both of these libraries are useful for testing and provide useful feedback, but the syntax requires a bit of a learning curve.

Tools that we used

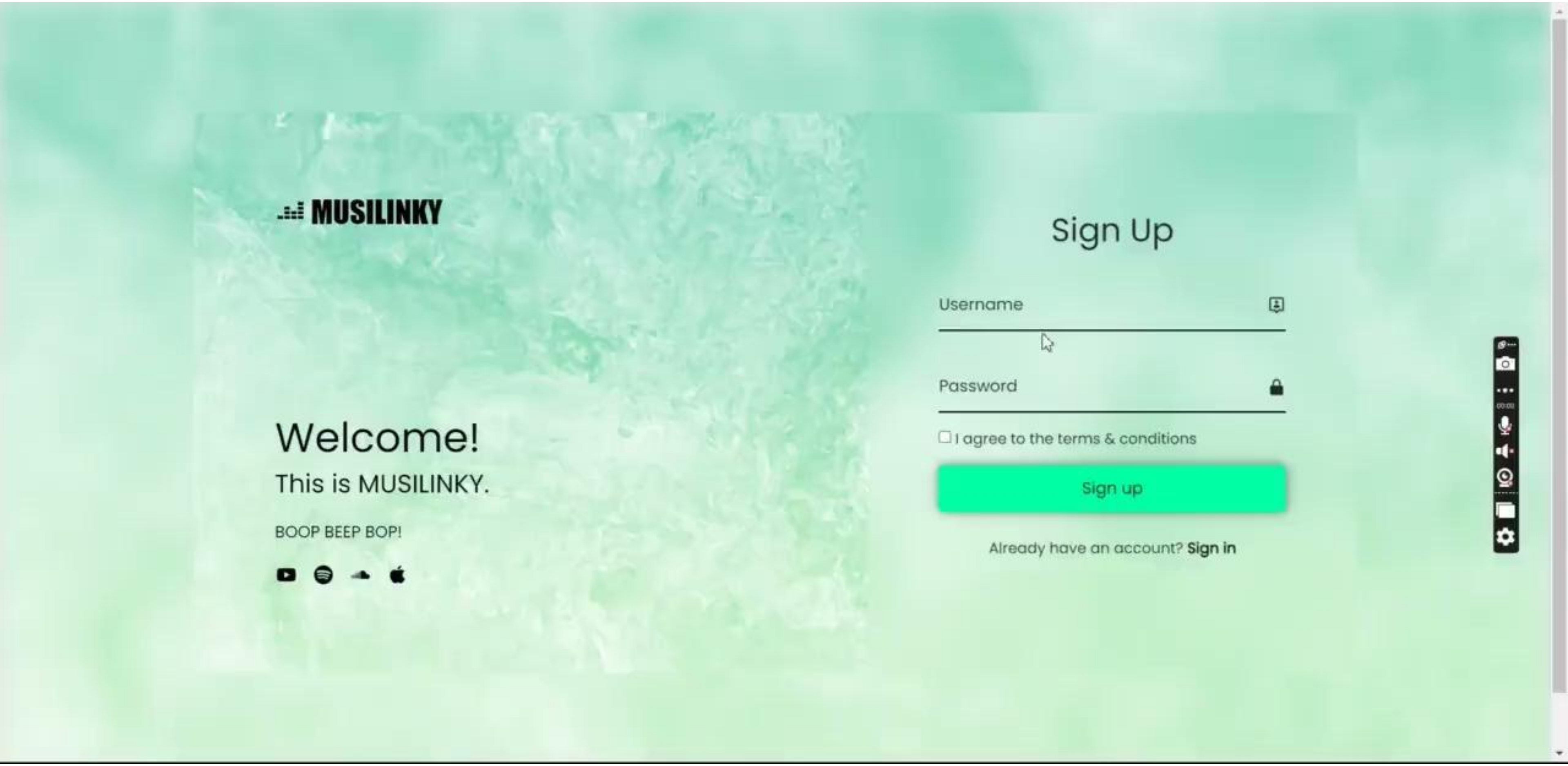
- External API - MusicAPI
  - **Purpose:** Track the user-generated links: Spotify, Apple Music, YouTube
  - **Rating:** ☆☆☆
  - **Reason:** This API call contained our axios calls to garner links for our music sites. Unique tool that allowed us to complete our unique website functionality.
- External API - Deezer
  - **Purpose**: Provide a music database to store the music we called in our search query
  - **Rating**: ☆☆☆☆☆
  - **Reason:** The Deezer Music API was a simple to follow API call that we made that gave us a song database to call from. Great because it's free as well.
- External API - DuckDuckGo
  - **Purpose:** Used to search the web for song links on other platforms
  - **Rating:** ☆☆☆☆
  - **Reason:** This external API reliably returns data and the response data is easy to parse.
- External API - Soundcloud Downloader
  - **Purpose:** Used to retrieve song data from Soundcloud links
  - **Rating:** ☆☆☆☆
  - **Reason:** This external API provides great functionality when the user inputs a Soundcloud link. It's free as well!

Architecture Diagram



Challenges

- Playlist functionality
  - Determining how to use the returned Deezer results in a playlist was difficult.
  - In the end, the solution lied in temporarily storing the data from the Deezer API in a SQL table.
- Time - time is the bane of everything.
  - We likely would have been able to add more with more time.
- Backend vs Frontend
  - With any software development project, there are bound to be inevitable conflicts between these two sides.
- External APIs
  - Determining which ones to use and how much to make use of them can be difficult.
    - For example, we had to drop support of Amazon Music because the MusicAPI did not return any data for these links.
    - We also needed to change the external API for Soundcloud because the original one did not return the right data.



MUSILINKY

Welcome!

This is MUSILINKY.

BOOP BEEP BOP!

Sign Up

Username

Password

I agree to the terms & conditions

Sign up

Already have an account? **Sign in**