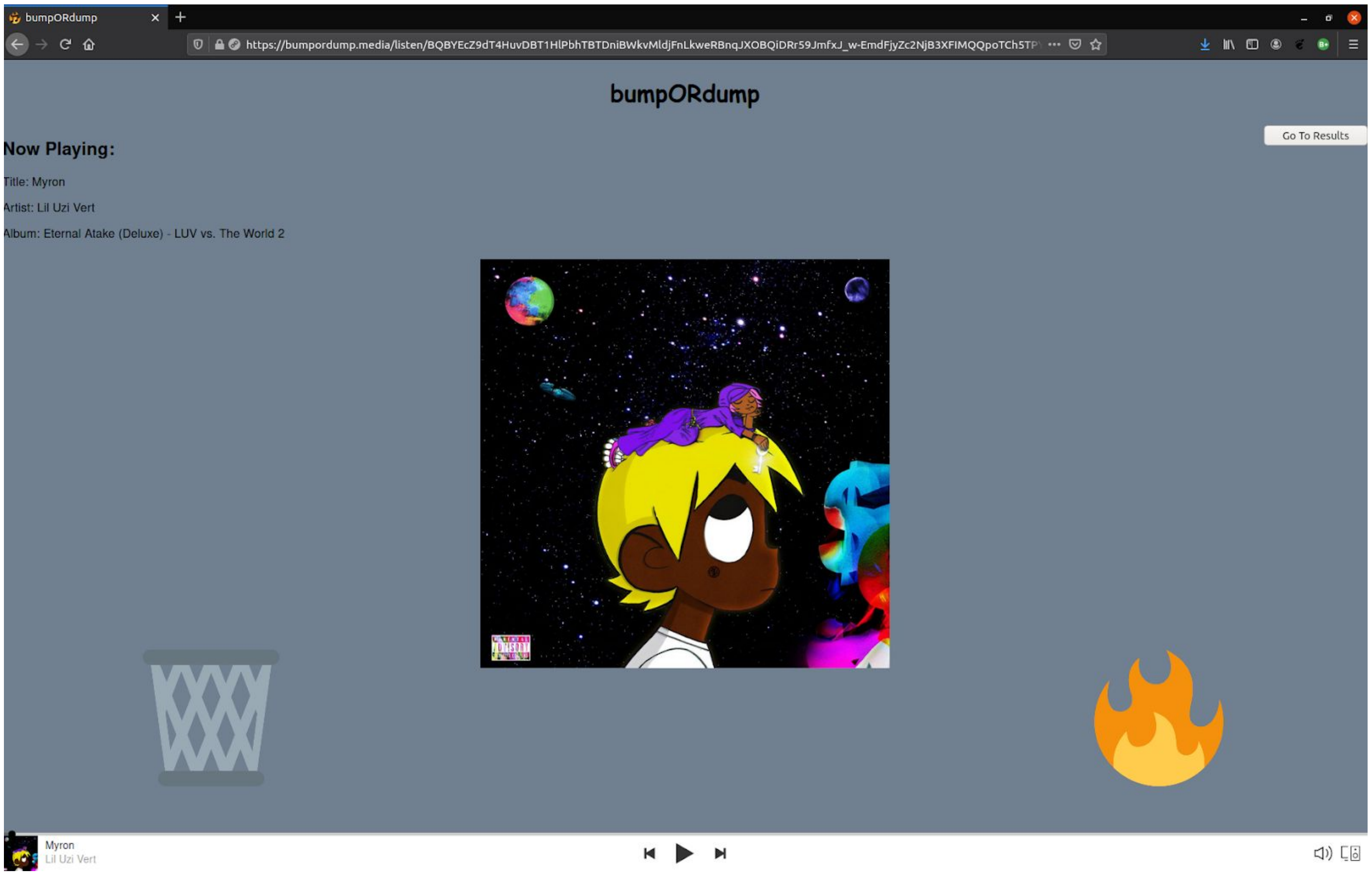
bumpORdump

Stallings, Jacob
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, United States
jacob.c.stallings@vanderbil.edu

Chudik, Quinton
Department of Electrical Engineering and Computer Science
Vanderbilt University
Chicago, United States
quinton.a.chudik@vanderbilt.edu


bumpORdump
https://bumpordump.media/listen/BQBYEcZ9dT4HuvDBT1HlPbhTBTDniBWkvMldjFnLkweRBnqJXOBQiDRr59JmfxJ_w-EmdFjyZc2NjB3XFIMQQpoTCh5TP
bumpORdump
Go To Results
Myron
Lil Uzi Vert

Now Playing:

Title: Myron

Artist: Lil Uzi Vert

Album: Eternal Atake (Deluxe) - LUV vs. The World 2

Figure 1: The listen page, it contains a like and dislike button, a player and information about each song

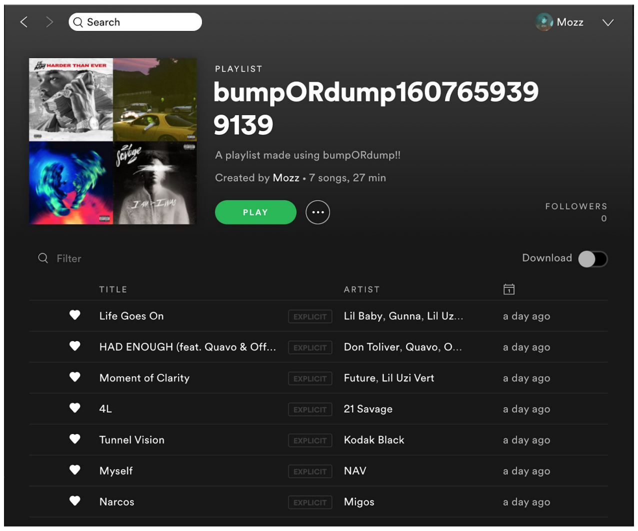
***Abstract*—bumpORdump is a web based application made for music lovers, casual listeners, and anyone who wants to find new music to enjoy. The focus of the application is on a web player that plays Spotify songs one by one, then lets the user choose to "like" or "dislike" the track. Liked tracks are saved to the user's Spotify library for later playback. The application heavily integrates Spotify, a popular music streaming service, and makes use of the platform's REST API to access the platform's proprietary algorithms to deliver recommendations to the end user. The application was built using a stack of relevant technologies such as React, Node.js, Docker, AWS, and Nginx. The result is a cloud-native web application that is hosted on a public domain, bumpORdump.media. It serves as a fun way to discover music and employs several cloud technologies taught throughout the Intro to Cloud Computing course. bumpORdump can be found at bumpordump.media and can be used by anyone that owns a Spotify premium account.**

Keywords—Spotify, React, Node.js, AWS, Web Application, Nginx

I. Introduction

The rise of music streaming in the past decade or so has vastly transformed how we discover new music. Gone are the days of browsing the record store shelves or purchasing entire albums to listen to from front to back. Today's streaming landscape puts an emphasis on quantity and availability -- users can browse thousands of songs from different artists and different genres all from their phone or computer. Often, with the entire world of music available at our fingertips, finding a new song or artist that you love can be akin to finding a needle in a haystack. Luckily, streaming services such as Spotify leverage powerful recommender systems to deliver personalized content that their users are more likely to enjoy. The inspiration for this application came primarily from the desire to leverage these algorithms via Spotify's extensive and developer-friendly application programming interface (API) to allow end users to discover new music in a fun way.

Inspiration was also drawn from the rise of dating apps such as Tinder, Bumble, etc. which allow their users to view profiles of potential matches one-by-one and then decide whether they would like to get to know that person or pass on them by swiping right or left or using a "like"/"pass" type of feature. The developers of bumpORdump aimed to extend this quick, one-by-one type of browsing to music, in the hopes that users will find a song or artist that resonates with them. Much in the way that the goal of dating apps is to find "the one", the developers want to allow users to find that special song or artist in much the same way.



bumpORdump1607659399139

A playlist made using bumpORdump!!

Created by Mozz • 7 songs, 27 min

TITLE		ARTIST	
Life Goes On	EXPLICIT	Lil Baby, Gunna, Lil Uz...	a day ago
HAD ENOUGH (feat. Quavo & Off...	EXPLICIT	Don Toliver, Quavo, O...	a day ago
Moment of Clarity	EXPLICIT	Future, Lil Uzi Vert	a day ago
4L	EXPLICIT	21 Savage	a day ago
Tunnel Vision	EXPLICIT	Kodak Black	a day ago
Myself	EXPLICIT	NAV	a day ago
Narcos	EXPLICIT	Migos	a day ago

Figure 2: An example playlist created by bumpORdump

Although there were a number of different directions this final assignment was allowed to go, we chose to develop a cloud-native web application for several reasons. First, the benefits for developing a web application using the cloud rather than hosting physical servers are numerous, including but not limited to scalability of "paying-as-you-go" and feasibility, making it possible to develop a web application without having to rent hardware. Secondly, it allowed us to focus on several concepts covered throughout the Intro to Cloud Computing course, such as containerization (using Docker), RESTful APIs such as Spotify's, and hosting with public cloud providers such as Amazon Web Services (AWS). Such workflows are becoming more and more common for developing web applications as more and more services move towards the cloud. Thus, we felt it relevant enough to pursue and hoped to leverage these technologies to bring an idea to life that we were both passionate about. More about the technologies used will be discussed in the *Implementation* section (Section 2).

The rest of this paper is organized as follows: Section 2 provides an overview of the implementation of the application and discusses technologies used and trade-offs between them. Section 3 highlights key challenges faced during development and the solutions to those challenges. Section 4 describes further improvements that could be made to the application and that will likely be added in the near future, and Section 5 concludes.

II. Design/Architecture/Implementation

A. *The implementation summarized*

bumpORdump is a web application made using JavaScript. The Backend uses Node.js and the Frontend uses React. Node Package Manager (npm) makes it easy to add and remove packages to node and using a package.json file, it also makes it simple to get the necessary packages when transferring the application to other machines. In order to use React we used both webpack and babel. Babel is used to modify the jsx code we wrote for React to js and webpack is used to pack all of the code we wrote together into one file that gets sent to the user upon the first get request.

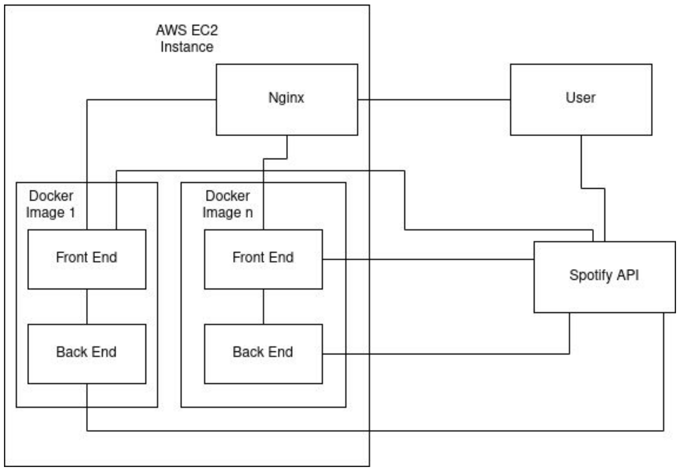
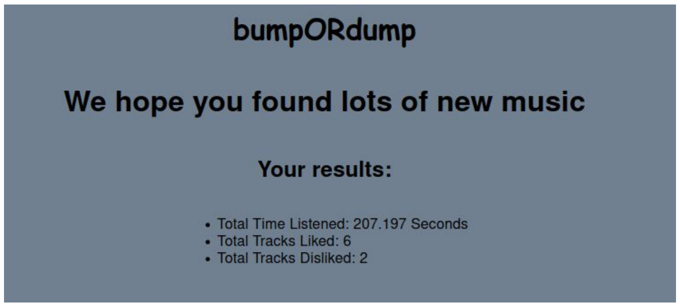


Figure 3: A high level model of our design and the different links connections that occur

B. Front End

As mentioned earlier, the Front End was built using React. React is useful creating reusable components and for making DOM manipulation much more efficient. Creating a full on Spotify Player in React is a difficult task, but we were able to find a solid open source version online that we were able to add into the project.[1] The Spotify Player works through directly making calls to the Spotify API, but the rest of our Front End either uses the Player for data or makes calls to the Spotify API, through calls to our own API on the Back End. The three main views our application offers are the start page, a basic page with information and a link to log in with Spotify, a listen page, the page that contains the player, information about the tracks, and buttons to like or dislike songs, and finally a results page, a simple page that displays some facts about the users usage. The first page contains an image taken from the listen page on the current website. However, it is actually a single page application (with the exception of authentication through Spotify) and no new page is ever loaded. On the change of “pages”, the DOM is just dynamically re-rendered using React. React also makes this more efficient by maintaining a shadow DOM (a copy of the DOM) and making changes on there, then diffing the two so that the only real changes that need to be made are the ones that differ. DOM manipulation can be costly so this greatly improves the Front End efficiency.


bumpORdump

We hope you found lots of new music

Your results:

- Total Time Listened: 207.197 Seconds
- Total Tracks Liked: 6
- Total Tracks Disliked: 2

Figure 4: The results page, showing the users listen time, total liked tracks, and disliked tracks

C. Back End

The Back End utilizes Node.js. We build the server using an express pipeline that is useful for including middleware for json parsing, cors (allowing users to go to Spotify to log in), cookie parsing, using a favicon, and using PUG (a server side HTML template engine). We also created an API that the front end uses to talk with Spotify. Our API consists of two main parts: auth and song. The auth part deals with the user authorization. It sends them to Spotify and then deals with the return. It is also used to send the users id to the Front End. There is also a place for an auto regeneration of the access_token to occur, however it is not fully implemented yet. We found a tutorial on GitHub for Spotify authentication and were able to use that to help us get the authorization working.[2] We made some minor changes to said code so that it would better work in the scope of our application. The song part deals with all the calls made to the Spotify API that deal with music. The Front end can make requests to get new recommendations, skip the current song, or like a song. Liking a song adds that song to the users library and to a playlist created by the application.

We initially thought we would use the Spotify API Node.js wrapper, but after beginning to use it we decided to just write our own calls instead. The wrapper didn’t seem to be too useful so we made use of the request node package.

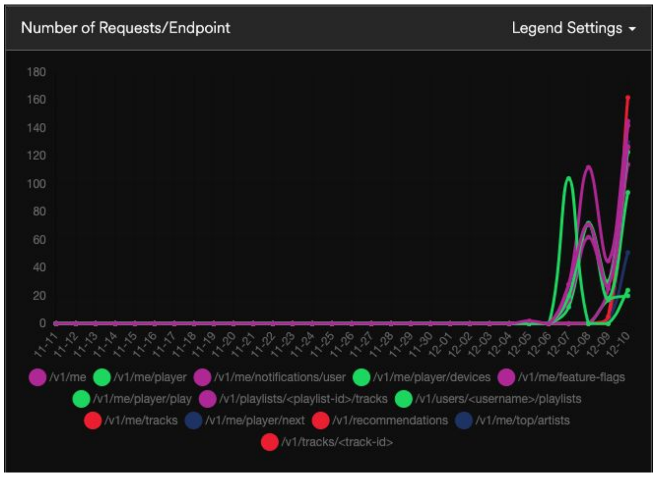


Figure 5: The Spotify for Developers Dashboard shows requests / endpoint and other usage statistics for bumpORdump.

D. Dev Ops

We are using an AWS EC2 instance to host bumpORdump. We chose between AWS and Chameleon cloud for hosting, but decided to go with AWS due to its wide variety of features. We used AWS to buy our domain (bumpordump.media) and to A list it to the public IP of our EC2 instance. On the instance we are using Nginx to load balance and to use a certification we got through certbot to allow us to utilize HTTPS. It forwards all requests over HTTP to HTTPS for secure access of our web application. Our application is running in a docker container on the instance and we have the ability to manually start up more containers, using the ip_hash method with Nginx to make sure users go to the same instance of the application. Currently we have no process of autoscaling and all work is done on one EC2 instance. The next steps for Dev Ops is to move to improve the scalability to make it automatic and be able to handle heavy loads.

III. Challenges

Most of the challenges we came across had to do with authorization. In order to be able to access the user's library and recommendations they must log in to their Spotify, on Spoify's website and then allow us to use their information. They then will be re-routed back to our application. The problem with this is that it makes it much harder to load balance and auto-scale our application. Users must be sent back to the same instance of our application in order for the authorization to properly work, as they are sent out with a cookie that must be checked upon return. We initially planned on using Kubernetes to load balance and scale, but we did not have enough time to fully figure out how to do this properly due to our authentication predicament.

Due to our time crunch, we decided to go with a Spotify player that someone had already made. This saved us a lot of time and trouble in terms of implementation. However, it was difficult to figure out at first, due to a lack of documentation. Much of how we learned how to use it was just by reading through the code itself. It also had some quirks and is occasionally buggy. Fortunately, for the scale and scope of this project, it worked well enough. It is slightly out of place in our project as it does not use our APIs.

Time was a big factor in how this project went for us. We were in a crunch and had to do lots of leaning on the fly as well. There were a few times where after testing something we had to make some major adjustments in a short amount of time to stay on track. We discovered a huge error late, when we went to hosting the application publicly, because we had not had the opportunity to test the application while it was being used by multiple users at the same time. We had to quickly solve some serious privacy and functional concerns and relaunch the application. Despite the shortness of time and issues we faced, I think the project turned out very well and that we met the majority of our goals.

IV. Improvements

Although we were happy with how the project turned out, there are of course some improvements that we may consider making in the future. Firstly, for the scope of this project, we felt it was not necessary to get Kubernetes involved in our workflow. This is mainly due to the fact that our application is just one standalone service and there was not a lot of need for routing between different Services with Kubernetes. Also, since the scale of the application is very small, auto-scaling and using Kubernetes to handle increased loads was something that we left for the future. If we make this change, we will need to use a Kubernetes's Ingress functionality to handle incoming traffic, necessarily deploying an ingress controller such as nginx in our cluster. For load balancing in the future, we may also look into AWS's Application Load Balancer.

There are some features of the web application itself that we'd like to potentially add to make the user experience more enjoyable. For one, we plan to implement a scoreboard to tell you how many songs you've listened to while still on the player page, and a "leaderboard" functionality to show how much music you have discovered compared to your "friends". This would likely use a feature such as Spotify's social integrations with Facebook. The design of the website could also be further touched up and improved, although this would likely require a UI designer or someone more artistically minded. The web player that we used for this implementation was an open-source React Spotify Web Player found on Github. This code eventually worked but seemed a bit buggy and resulted in us having to basically build the rest of the application by working around this component, so it's possible in the future that we might replace this component with a React player of our own design -- one that fits in better with the rest of the codebase. Finally, the application is currently not mobile compatible, though this is due to Spotify not supporting use of the Web Player on mobile devices. In the future, if Spotify decides to make it possible to have the player on iOS and Android, a mobile-compatible version or even iOS or Android native version of the app could be developed.

As far as the backend goes, the website could be improved to be more secure. Currently, we use HTTPS which helps to keep incoming traffic secure, but the authentication process as it stands right now could be improved to be more secure while the user is using the site. We may look into potentially moving away from OAuth if we think a more secure option is available. Additionally, the both the backend and front-end code could be cleaned up to be more readable, efficient, and fault-tolerant.

Overall, the application does what we wanted it to do for this assignment, but we may likely look into developing it further in our spare time in the future by implementing some of the improvements mentioned here.

Conclusion

The application described in this paper served as a fun way to discover new music by combining elements from streaming service and dating apps to produce a unique application that is both enjoyable to use and functional. Through developing this application, we were able to get hands-on experience working with relevant concepts such as containerization, RESTful APIs, and cloud-native web application architecture to bring our idea to life. Although some challenges were faced along the way, such as is usually the case when working with technologies that one does not have extensive experience, we were able to overcome many of these challenges and learn valuable lessons about the technology that we can apply in our future work. Both of us are passionate about the idea of bumpORdump and would like to devote time in the future to implement some of the improvements listed in order to fully complete the application's transition from a school project idea into a living, breathing website. We would like to thank Dr. Gokhale and our TA Ziran Min for their commitment to instruction and their guidance in helping us navigate the concepts throughout this course, and we believe that this application is satisfactory as a culmination of the valuable things we have learned from them this semester.

References

Spotify provides lots of useful guides and information both on their website and on GitHub that we were able to use and learn from. We were also able to find some other open source code on GitHub, specifically, the React Spotify Player was a big part of the bumpORdump application. This and many of the smaller pieces were added into the application using node package manager.

Lots of the other information we got was from stack overflow, various blogs, and official documentation.

[1] G.Barbara,"react-spotify-web-playback", https://github.com/gilbarbara/react-spotify-web-playback

[2] H. Rowlinson, J. Chen, J. Perez, M. Cedaro, M. Thelin, M. Usov, P. Alexander, W. Dynus, "Spotify Accounts Authentication Examples", https://github.com/spotify/web-api-auth-examples