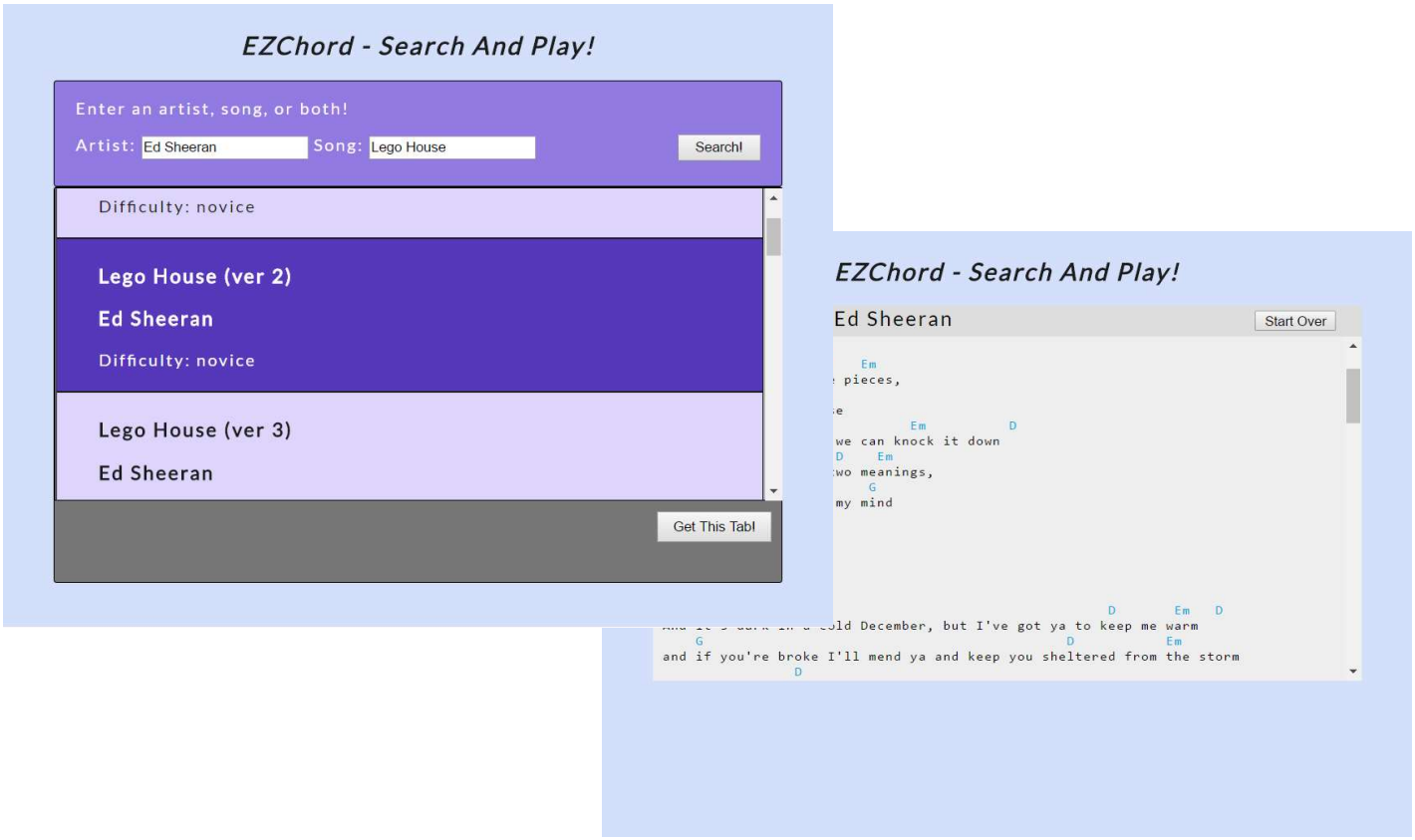
EZChord

Search And Play!



Designed by Alec Bielanos
IGME330 - Spring 2017
Professor Cody Van De Mark

Overview and Features

EZChord is a web service designed for beginner to intermediate guitarists who would like to quickly and efficiently retrieve their favorite tablatures (chords matched up with lyrics to a song). It's lighter and more basic than other services like it, keeping confusion to a minimum, with graphic examples of chord shapes. The full list of features includes:

- **Search Queries** – The user can search existing guitar chord databases; they will be able to choose from thousands of artists and songs to find the exact song they're looking for.
- **Intuitive User Interface** – Armed with the tablature of the song, the user can effortlessly click and view chords side-by-side with the tab to see variations on the fly. The UI will be kept simple to keep the focus on the music!
- **Chord Recommendations** - The service will collect the chord names in order and suggest additional ones that may be added in between. When the user hovers over one of the existing chords, a box will appear with the suggested additional chord and hand shape.
- **Pleasantly Simple –** As mentioned above, this lightweight service offers only what you need. It's tastefully colored, animated, and includes nothing but the necessary information you need when rapidly searching for tabs to play.

High-Level Architecture

The EZChord service is implemented via both a client and a server for various reasons explained below. There are two main 'Requests' that the user can make. These are standard REST style queries which communicate via 'get' routing to a simple proxy server made in Node.js and hosted on *Heroku*.

The first of these requests asks for a list of available tabs based on the artist and/or song name that is entered in a form on the client side. On the server side, it uses an API that reaches out to a tab database (website) with this information and returns the available songs and their artists.

The latter is called when the user has selected a tab from the search results returned by the former. It accepts a URL and makes use of a Node.js implementation of YQL (Yahoo Query Language) to scrape the page using a hard-coded X-PATH to return the HTML from the page. This is then passed back to the client and displayed on the page.

Once the user has the tab they want, they may click on any of the tabs (highlighted in blue) to get a basic hand/finger configuration for that chord. Technically, there is a third ajax request here done by the API that gets the chord images, although it simply contacts its home server to grab the correct image to display on the screen. More information on this can be found in the technical decisions section below.

Design Decisions

The design decisions for this website came out of both time and API constraints. Over the course of the two weeks, I had three APIs fail me in some way or another, as well as being declined access to a fourth due to licensing issues. The constant switching of design to fit around these APIs delayed production somewhat, and all of the extras I ended up with still could not do everything I had intended to do in the original design document.

For styling of the webpage, I went with simple colors and primitives (mostly rectangles). Currently, the styles are not responsive but the entire app does fit into an 800x600 frame as specified in the prompt. The color swatch was chosen in a triad pattern based on a blue that I liked:

Although only a few of these colors made it to the final product, it leaves the door open for more colorization later. Interestingly enough, there is more of the secondary purple palette than the blue that I originally chose.

For the fonts, I went for my “go to” sans-serif of Lato (the font that this document is written in). For the tab itself, I used a monospace variant of Droid Sans, as I enjoy the font and needed a monospace font because of the pre-formatting that the tab was using.

Overall, I designed this project out of my passion for music as well as to test myself on my knowledge of client/server and ajax coding. Additionally, this serves as a valid tool for me to use when rapidly scrubbing through songs for gigs or to get ideas and serves as a proof of concept for page scraping in general.

Technical Decisions

As I mentioned, much of the functionality depended on the APIs which may or may not have made it into the final version. I chose Node.js as my ajax recipient mainly because it had libraries that proved instrumental (no pun intended) to the final product. The extras that I included all have a purpose and were chosen not necessarily because they were easy, but because they would make the project function as intended.

The decision to keep the HTML and CSS separate from my Node.js server as opposed to sending the page directly from Node was essentially because of my ties to Banjo at this time, and my familiarity with static hosting. To host on the server would mean more research which would have detracted from the overall production.

I did not separate my code into actual modules because I did not feel that the codebase was large or ‘diverse’ enough. Most of the methods were helpers that relied on the others to be present in the same class/module, so I decided to put everything in an Immediately Invoked Function Expression (IIFE) to ensure encapsulation of all variables and methods. A rather interesting note on this: In order to get the chord image requests to work and display the chord shapes as you clicked on them, I had to call the APIs onload function, as that was the only place where it made the ajax request to get the images. Luckily, the entire API was global, because there was no other library that did what it was able to do with such ease.

The standard HTML functionality was used in the design, namely forms and buttons which activated code via onclick events. The HTML itself is fairly minimal, as most manipulation of the DOM is done through JavaScript in response to the ajax responses.

Bibiliography

This Web Service would not be possible without the help of several sources, which have been laid out below along with their implementation in the project:

- **Ultimate-Guitar-Scraper** – A node library used to scrape the ultimate guitar website for tablature search results. https://www.npmjs.com/package/ultimate-guitar-scraper

- **Yahoo Query Language (YQL)** – Approved by the Professor, a library to scrape the URL of the selected tab using the XPATH. I used the Node.js implementation of this as well. https://www.npmjs.com/package/yql | https://developer.yahoo.com/yql

- **Express** – A lightweight node http server library for simplifying the proxy and get requests https://www.npmjs.com/package/express

- **Scales-Chords API –** The API used on the client side for getting pictures of the chords when they were clicked. https://www.scales-chords.com/api/

- **JQuery** – For some functionality on the client side, mainly animations, as well as adding and removing attributes to some elements https://jquery.com

- **StackOverflow Users –** For this one specific solution of using a set list of headers with express. http://stackoverflow.com/questions/31661449/express-js-how-to-set-a-header-to-all-ress

- **Paletton –** For the color palette. http://paletton.com

- **Google Fonts** – For the spectacular fonts, as always. https://fonts.google.com

- **Documentation** – A lot of learning was done on this project, a good portion of which was done by simply looking at the documentation for the various libraries; I am thankful that they all had examples, as well as W3Schools for all of the information it has.

Grading

Looking back at this project, I see the incredible potential for improvements and expansion, the sure signs of a great portfolio piece (or future great portfolio piece). Although my design was not exactly what it was two weeks ago, I'm proud with what I put out.

I'll run over the basics. The code is clean, well commented, and does what it needs to do. The page itself is not an eyesore both when viewing the source files and in the browser, and although all of the animations were not implemented, I was still able to find solutions for things like status updates in lieu of an animated progress indicator. Additionally, I don't believe I was able to use dynamic sizing that scaled with the viewport (media queries). The extra HTML5 feature that I used was semantic markup (section, header, etc.) tags in my HTML, but I am not sure if that qualifies.

I think the real 'above and beyond' here lies in a deeper understanding of the client/server and ajax framework, as well as the inclusion of unique external APIs and libraries. (Google Maps need not apply!) We didn't cover much of what I did in class, especially the page scraping, but I was able to handle it on my own. I did not give up when my APIs did not work and tried anything possible to get the ideas that I had in my head into the browser.

Because of the above criteria, I would give my project a 94 out of a possible 100.