Node.js/Javascript based Single Page App

In this task you are going build a small single-page web app that is based on just the native capabilties of javascript. Specifically you will build an app that allows a user to open a chord pro formatted (chords and lyrics) text files hosted on a node.js server and then within their browser properly display the chords above the lyrics and also allow users to transpose the chords up or down to a different musical key.

The components are:

Server: Built with just Node.js and only its built-in modules (e.g. http, path, fs, url) For this task the supplied server is complete. You will only be writting the client-side javascript that runs in the browser.

Client: Browser-based javascript that is provided with a web page from the server consisting of html, css, javascript.

Task Restrictions:

Technology Restrictions: You cannot modify the server.js code in this task.

This task is based on 11 design requirements numbered **R1.1**...**R2.4**

Req Type	Task Grading
R0.x	**Critical Submission and Intent Requirements.** Task gets 0 if **any** criticalsubmission requirement (shown in red) is not met.
R0.x	**Good Practice Requirements.** You lose 2 marks for each good practice requirement (shown in amber) not met.
Rx.x	**Design Requirements.** You earn 2 marks for each design requirement (green) satisfied and well implemented; 1 mark if it's partly met or met but not well implemented; and 0 if it's not met or attempted.

Good Programming Practice Requirements

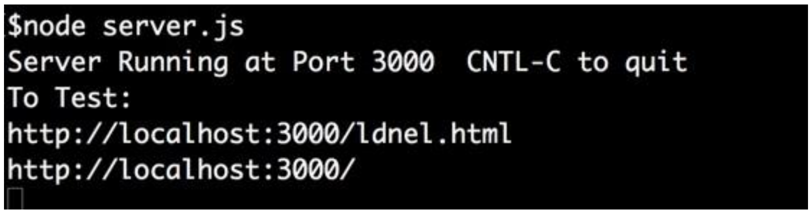
The following requirements pertain to all your tasks regardless of what your application is supposedto do (i.e. regardless of the design requirements). These requirements are to ensure that your code is usable, readable, and maintainable.

R0.0 UNIQUENESS REQUIREMENT. The solution and code you submit MUST be unique. That is, it cannot be a copy of, or be too similar to, someone else's code, or other code found elsewhere. You are, however, free to use any code posted on our course website as part of our task solution. [Task mark =0if this requirement is not met.]

R0.1 CODE SUBMISSION ORGANIZATION AND COMPILATION: You should submit all the code files and data files necessary to compile and run your app. We will execute your app by following the instructionsyou provide in the `README.txt` file. You must submit a single`.zip` formatted file to brightspace. (not .rar or .tar or whatever). Though you are permitted to write code on Windows, Linux, or Mac OS the code must be generic enough to be OS agnostic. (See also the the requirement below about not submitting the `node_modules` directory). Your code must work with at least a current Chrome browser and version 18.x.x of node.js
[Task mark =0 if this requirement is not met.]

R0.2 README FILE: Your submission MUST include a `README.txt` file explaining how to setup and runyour app. We should NOT have to look into your code to figure out how to start up your app. Your `README.txt` MUST contain the following:

- Version: node.js version number and OS you tested on your code on.
- Install: how to install needed code modules. This will likely look like `npm install` or `npm install module_name`
- Launch: Instructions on how to launch your app. e.g. `node server.js`. As the course progresses there will be more launch options so it's important to provide instructions.
- Testing: Provide Instructions on what we should do to run your app. e.g. visit `http://localhost:3000/mytest.html?name=Louis`. If your app requires a userid/password to run then provide one for us to use. Your server should print to the console the URL's that should be visited by the browser to demonstrate your app. List them in the order you want us to visit them:

  ```
  $node server.js
  Server Running at Port 3000  CNTL-C to quit
  To Test:
  http://localhost:3000/ldnel.html
  http://localhost:3000/
  ```

- Issues: List any issues that you want the marker to be aware of. In particular, tell us what requirements you did not implement or that you know are not working correctly in the submitted code. Here you are giving us your own assessment of your app.

  Pay attention to any specific URL's that must be supported by your app.[Task

  mark =0 if this requirement is not met.]

R0.3 INTENT REQUIREMENT: The solution and code you submit must comply with the intent of the task. For example if you are required to build a node.js/javascript server and you choose to build an apache/PHP server instead you will have violated the intent of the task even though the user input- output experience might be the same. As another example, if you are asked to build a "thick client" solution where the server just supplies data and the browser renders it but you build a "thin client" solution where theserver renders all the HTML pages you will have violated the intent even though the user's experience wouldlook the same. [Task mark =0 if this requirement is not met.]

R0.4 VARIABLE AND FUNCTION NAMES: All of your variables and functions should have meaningful names that reflect their purpose. Don't follow the convention common in math courses where they say things like: "let x be the number of customers and let y be the number of products...". Instead call your variables **numberOfCustomers** or **numberOfProducts**. Your program should not have any variables called "x" unless there is a good reason for them to be called "x". (One exception: It's OK to call simple for-loop counters i,j and k etc. when the context is clear and VERY localized.) Javascript variables don't have types which can help clarify their meaning so choosing good names is even more important. Many functions in javascript are annonymous (have no name) and so the name of the variable that refers to them is even more important.

Remember: any fool can write code that a computer will understand; the goal is to write code that we can understand.

R0.5 JAVASCRIPT IN STATIC HTML: Your static html pages should NOT make direct reference to javascript functions. Don't do something like the following:

```
<button type="button" onclick="myFunction()">Try it</button>
```

Instead do something like this:

```
<button type="button" id="submit_button">Try it</button>
```

and elsewhere in your javascript file say:

```
document.getElementById('submit_button').addEventListener('click', myFunction)
```

R0.6 COMMENTS: Comments in your code must coincide with what the code actually does. It is a common bug to modify or cut-and-paste code and forget to modify the comments and so you end up with comments that say one thing and code that actually does another. Don't over-comment your code - instead choose good variable names and function names that make the code "self commenting". Don't be reluctant to create local variables so that the variable name provides more clarity -there is no prize for having the fewest lines of code.

R0.7 MODULARIZATION: Your client-side and server-side javascript should not be in two giant files. Break you client-side javascript into smaller manageable and readable files and include them individually with `<script>` tags in your html document. On the server-side use `requires` or `imports` appropriately to organize your code into managable size files.

R0.8 BLOATED CODE: If your assigment uses external modules installed with npm, **DON'T** submit the `node_modules` directory with your code (it's potentially huge). Remove that directory and only submit the `package.json` and `package-lock.json` files. We will use these files to intall the required modules.NPM modules are platform specific and must be reinstalled on the markers platform (they would have to remove your `node_modules` directory).

R0.9 CITATION REQUIREMENT: If you use code from other sources you should cite the source in comments that appear with the code. If the source is an internet website then put the URL in the comments. You may use bits of code from outside sources but this may not form the complete solution you are handing in.

VERY IMPORTANT: Any sample code fragments provided may have bugs (although none are put there intentionally). You must be prepared to find errors in the requirements and sample code.

Application Design Requirements

Background

For this task we want to render on the client browser contents of chord-pro formatted songs stored onthe server. Below is an example of such a file. The file format represents the lyrics of songs with chord symbols embedded within the lyrics. The chord symbols appear in the [] brackets. This format is convenient for storage because the chords always stay in the correct relative position with the lyrics because they are imbedded within the lyrics.

Chord Pro Text File:

```
Sister Golden Hair -America

verse1:
Well i [E] tried to make it sunday but i [G#min] got so damned depressed
That i [A] set my sights on [E] monday and i [G#min] got myself undressed
I ain't ready for the alter, but i [C#min] do [G#min] believe there's [A] times
When a [F#min] woman sure can [A] be a friend of [E] mine [Esus2] [E]

verse2:
Well i [E] keep on thinkin bout you sister [G#min] golden hair surprise
That i just can't live without you can't you [G#min] see it in my eyes
I've been [A] one poor corre[F#min]spondent, i've been [C#min] too too [G#min] hard to [A] find
But it [F#min] doesn't mean you [A] ain't been on my [E] mind [Esus2] [E]

chorus:
Will you [B] meet me in the middle will you [A] meet me in the [E] end
Will you [B] love me just a little just en[A]ough to show you [E] care
Well i [F#min] tried to fake it i [G#min] don't mind sayin i [A] just can't make it

repeat intro, then verse 2, then chorus, then they do this
doo-wop thing that uses the chorus (B - A - E) thing
```

We want you to build the client-side single page app so that the lyrics (words) and chords are displayed in a browser web page. Moreover, once displayed, the user should be able to transpose the music to another key.

1) Server-Side Requirements

R0.4 The server code is provided for you. You should not modify this code. Here are some properties of the server code provided with this task.

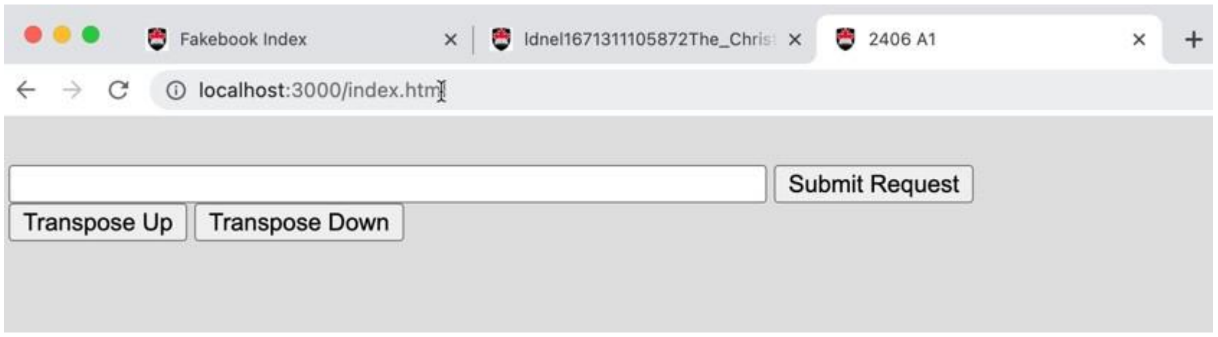
The server has a `songs` directory of chord pro formated text files. The server allows a client, via a browser. For this task the user is required to type the song title exactly. A more general and elegant solution might allow the matching to be more general (e.g. not case sensitive, not sensitivey to white space etc.).

Server is hosted on port 3000 and reachable from a browser on the same machine visiting `http://localhost:3000/index.html` Expect that we will be adding more songs to the `songs` directory of the server when we test your code. That is, your code should not work with just the songs in the demo code.

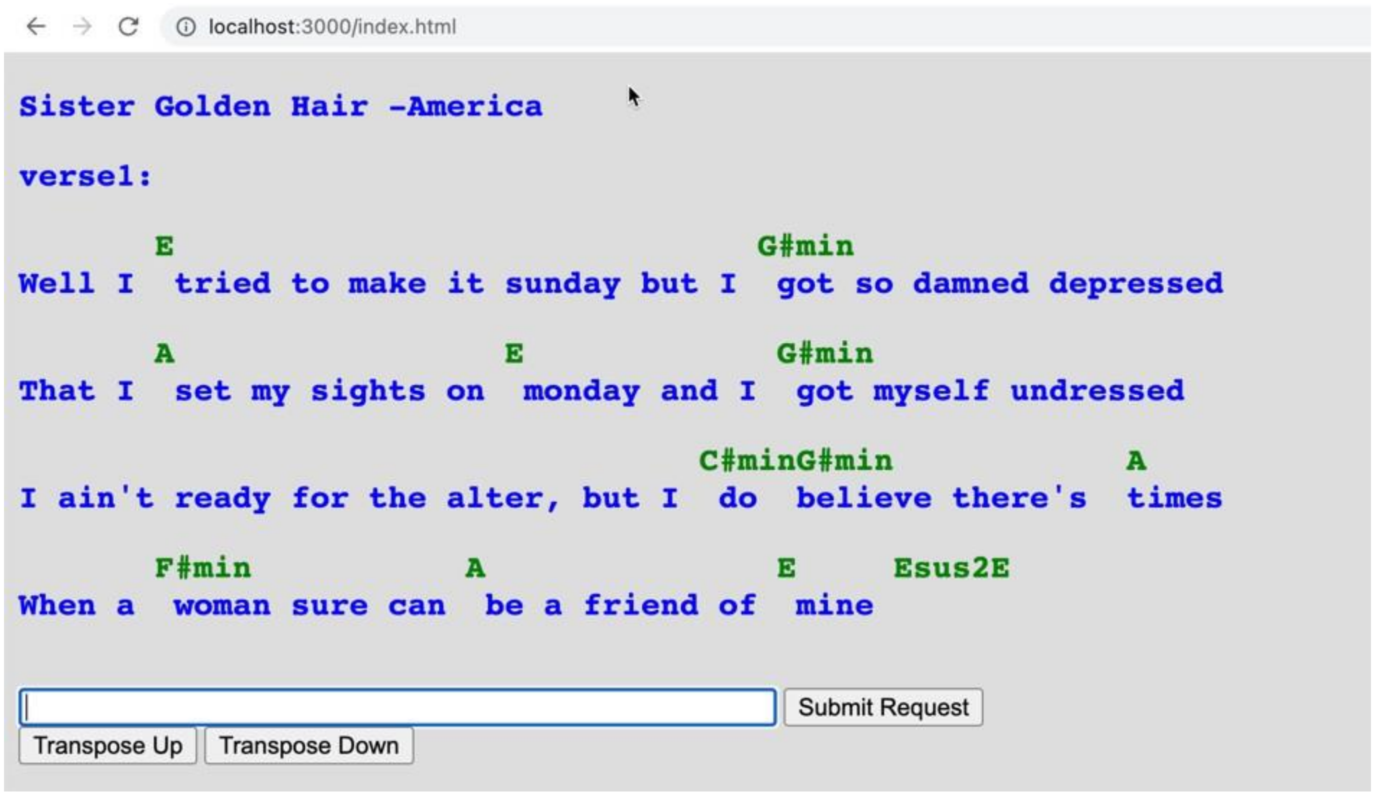
Client-Side Webapp Requirements

(Note: see the mockup video after the requirements are discussed below.)

R1.1 When the client visits the application URL http://localhost:3000/index.html the resulting webpage should have a text input field and a "Submit Request", button a "Transpose Up" button and a "Transpose Down" button and may look something like the following:



R1.2 The client should be able to request a song from the server by typing the song title in the text field and clicking the "Submit Request" button. If the song is available on the server the downloaded song should be displayed for the client as html content something like the following.



```
Sister Golden Hair -America

verse1:

       E                              G#min
Well I  tried to make it sunday but I  got so damned depressed

       A                 E             G#min
That I  set my sights on  monday and I  got myself undressed

                                   C#minG#min             A
I ain't ready for the alter, but I  do  believe there's  times

       F#min           A               E     Esus2E
When a  woman sure can  be a friend of  mine
```

R1.3 The lyrics and chords shown on the webpage should have the lyrics in blue and the chords in green. The colouring should be implmented using CSS styles.

R1.4 The lyrics and chords shown on the webpage should be in the same monospace font (e.g. courier font).

R1.5 The chords should be properly positioned above the lyrics and MUST NOT be in square brackets as they are in the chord pro formatted data provided by the server.

HINT: if your chords are represented as a seperate lines of text with blanks used to space them out you should probably use <pre></pre> tags (prefromatted text) rather than <p></p> since the paragraph tags won't preserve the white space. Check on W3Schools for the difference between the two html tag types.

R1.6 If chords are embedded inside a word in the chord pro format then the chord should appear above the word and not have the word split. For example, if the chord pro data looks like `dep[G#min]ressed`. Then the page should display it like this this:

```
   G#mim
depressed
```

and not like this:

```
   G#mim
dep     ressed
```

R1.7 If the client requests a song from the server that does not exist the webpage could appear blank with only the buttons but the code should not crash as a result of a client requesting a non-existing song.

Transposition Requirements

Musicians often need to transpose the chords of a song to another key. That is, transpose the chords up or down by a certain number of semi-tones (or half-steps). You will implement this feature using the transpose buttons on the webpage.

R2.1 If the user presses the "Transpose Up" button the chords should be transposed up one semitone (or half-step) as they say in music. That is, an Am7 chord should be become A#m7 (or Bbm7), F#7 should become G7, Ebmin should become Emin etc. (see below for more explanation of what chord transposition means.)

R2.2 If the user presses the "Transpose Down" button the chords should be transposed down one semitone (or half-step) as they say in music. That is, an Am7 chord should be become Abm7 (or G#m7), F#7 should become F7, Ebmin should become Dmin etc. (see below for more explanation of what chord transposition means and be prepared to google for more understanding.)

R2.3 Transposition should work correctly for chords that have more than one letter name in them. For example, an A7/C# chord. Here both the "A" and "C#" letter names appear within the same chord. This is called a slash chords -because of the forward slash. It means the musician should play an A7 chord with a C# note in the bass. The example song "Never My Love" contains some slash chords.

R2.4 When transposing the chords of the original key should be shown in green but if the chords have beentransposed they should appear in red.. That way if the user is transposing up and down they will always recognize when they are back to the original key by the colour of the chords. So again: use a red colour to show any transposed chords and a green colour to show chords of the original key. -See the demonstrationvideo below for an example of how this works.

Transposition Explained

Here is an explanation of how chords are transposed but you should google this concept if you need more clarification -remember the programmer will need to learn the domain that the app is about.

In western music there are 12 notes whose names are A, A# or Bb, B, C, C# or Db, D, D# or Eb, E, F, F# or Gb, G, G# or Ab. Moving from one note to the next one is called a distance of a semitone (or a half-step). It's perhaps easiest to visualize these as a table or array.

0	1	2	3	4	5	6	7	8	9	10	11
A	A#	B	C	C#	D	D#	E	F	F#	G	G#
A	Bb	B	C	Db	D	Eb	E	F	Gb	G	Ab

Seven of the notes have simple letter names (A,B,C,D,E,F, or G) and five of the notes have two names: a sharp (#) or flat (b) name (A#=Bb, C#=Db, D#=Eb, F#=Gb, G#=Ab).

To transpose a chord Dm7 up 4 semitones you would locate the chord's root name D in the table at index 5 and then look up the new root name at location 5 + 4 = 9. That is, Dm7 transposed up 4 semitones is F#m7. (If you end up going off the table you would wrap around to the start -i.e. do a mod 12 operation on the index sum.)

To transpose a chord Dbmaj down 6 semitones you would locate Db which is at index 4 then subtract 6 which would be -2 or 10 if we are wrapping around. Thus Dbmaj transposed down 6 semitones would be Gmaj.

Thus transposing up is moving to the right by the required number of semitones wrapping around if necessary. Transposing down is moving to the left by the required number of semitones wrapping around if necessary.