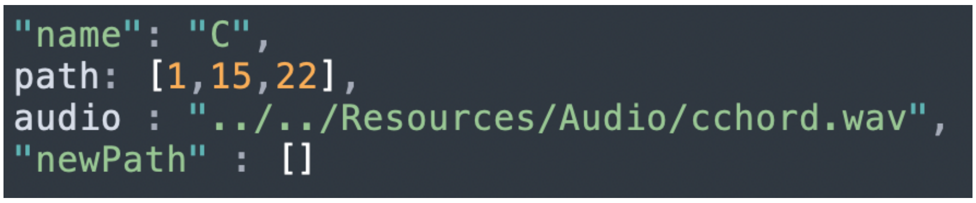
Cheong Hui En, Elizabeth (A0220121J)

Overview

For my final project, I was stylistically inspired by a YouTube video by musician Louie Zong titled “the funk guitar dojo”. I chose to make a guitar learning application that teaches users to play different chords. The user selects their experience level on the guitar, and is directed to the respective pages. Each page has a list of chords the user can learn, complete with fingerings, chord name and audio of how the chord should sound when played. The user has the option to edit the fingering of the chord, and when they click back to the chord in the chord library, their custom fingering will be saved. The page also comes with a notes function, where a user can write notes that will be displayed in the sidebar. These notes persist after reload, as they are saved to the machine’s local storage.

Drawing the Chord

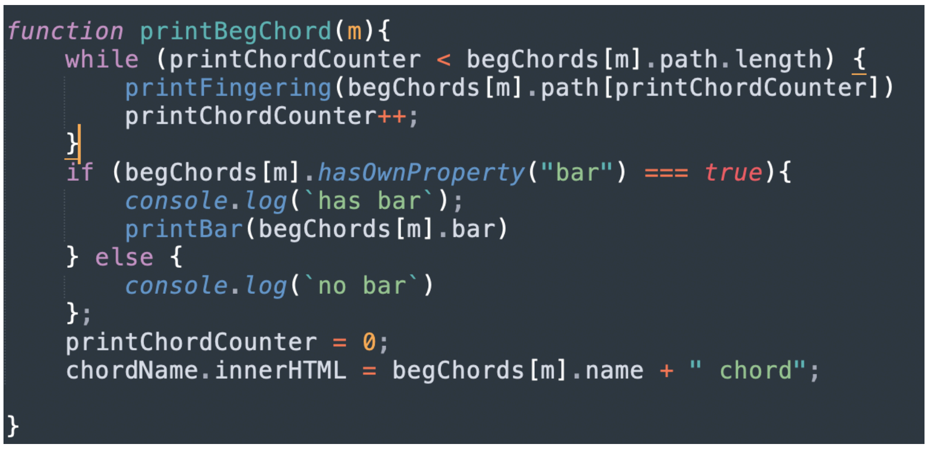
I drew the chord template on the Raphael paper using the path function. Then, I used a function to create fingeringArray, which holds an object for each position on the chord template, such that each fingering has an x-coordinate and a y-coordinate. From there, I created an array for the beginner chords as objects with the properties "name", “path”, which contained an array of numbers corresponding to the fingering positions in the fingeringArray, and “audio”, which held a file path to the corresponding chord audio.



```
"name": "C",
path: [1,15,22],
audio : "../../Resources/Audio/cchord.wav",
"newPath" : []
```

To further modularise my code, I wrote a function printFingering(n), that finds the nth element of the fingeringArray, takes its fret and string values and prints a black circle, and a function printBar(n), that finds the nth element of the fingeringArray, takes its fret value and prints a black rectangle.

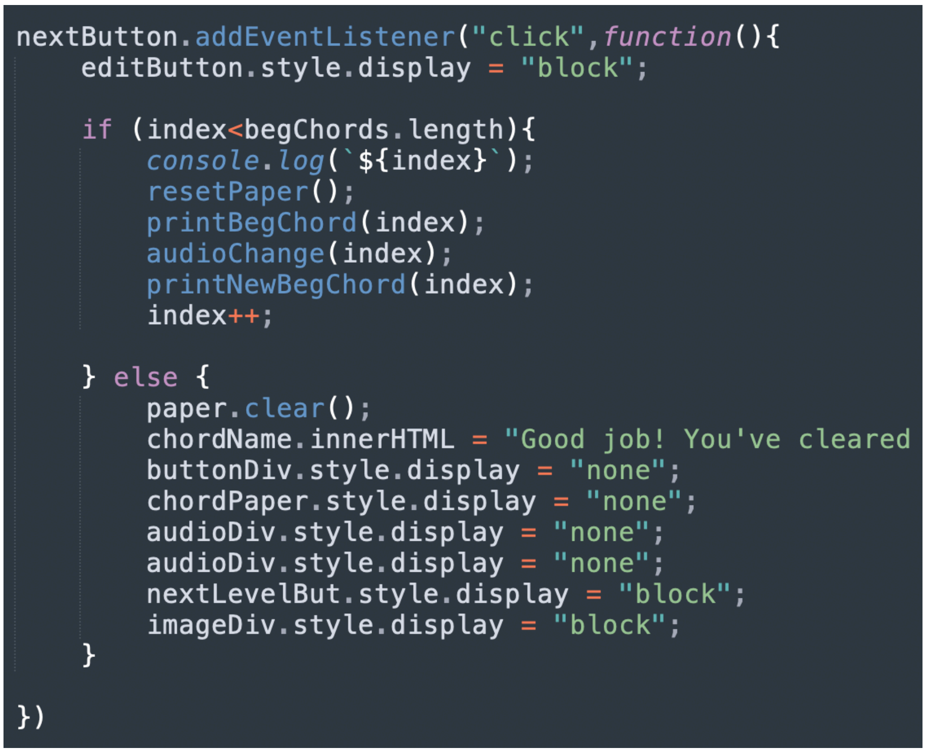
These functions are used in printBegChord(m), which uses a while loop to pass the numbers in the begChords[m].path array to printFingering(m) and print the name of the chord to the chordName div. However, as some chords have a bar and some do not, I had to research on how to check if an object had a property. I learned that this can be achieved using the object.hasOwnProperty(“propertyName”) function. Thus, the function also checks if the chord has a “bar” property and draws it to the paper.



```
function printBegChord(m){
    while (printChordCounter < begChords[m].path.length) {
        printFingering(begChords[m].path[printChordCounter])
        printChordCounter++;
    }
    if (begChords[m].hasOwnProperty("bar") === true){
        console.log(`has bar`);
        printBar(begChords[m].bar)
    } else {
        console.log(`no bar`)
    };
    printChordCounter = 0;
    chordName.innerHTML = begChords[m].name + " chord";

}
```

I created a next button to hold all of the changes that needed to be made. When clicked, the callback function used an if statement to check if there was another chord in the begChords array. If so, it cleared the Raphael paper, redrew the template, printed the next chord, then called a function audioChange(n) to change the audio player's audio source to the correct chord. If there was no other chord, the callback function hid all buttons and displayed some text and a next level button that redirected the user to the next page.



```
nextButton.addEventListener("click",function(){
    editButton.style.display = "block";

    if (index<begChords.length){
        console.log(`${index}`);
        resetPaper();
        printBegChord(index);
        audioChange(index);
        printNewBegChord(index);
        index++;

    } else {
        paper.clear();
        chordName.innerHTML = "Good job! You've cleared
        buttonDiv.style.display = "none";
        chordPaper.style.display = "none";
        audioDiv.style.display = "none";
        audioDiv.style.display = "none";
        nextLevelBut.style.display = "block";
        imageDiv.style.display = "block";
    }

})
```

Editing and Saving the Chord

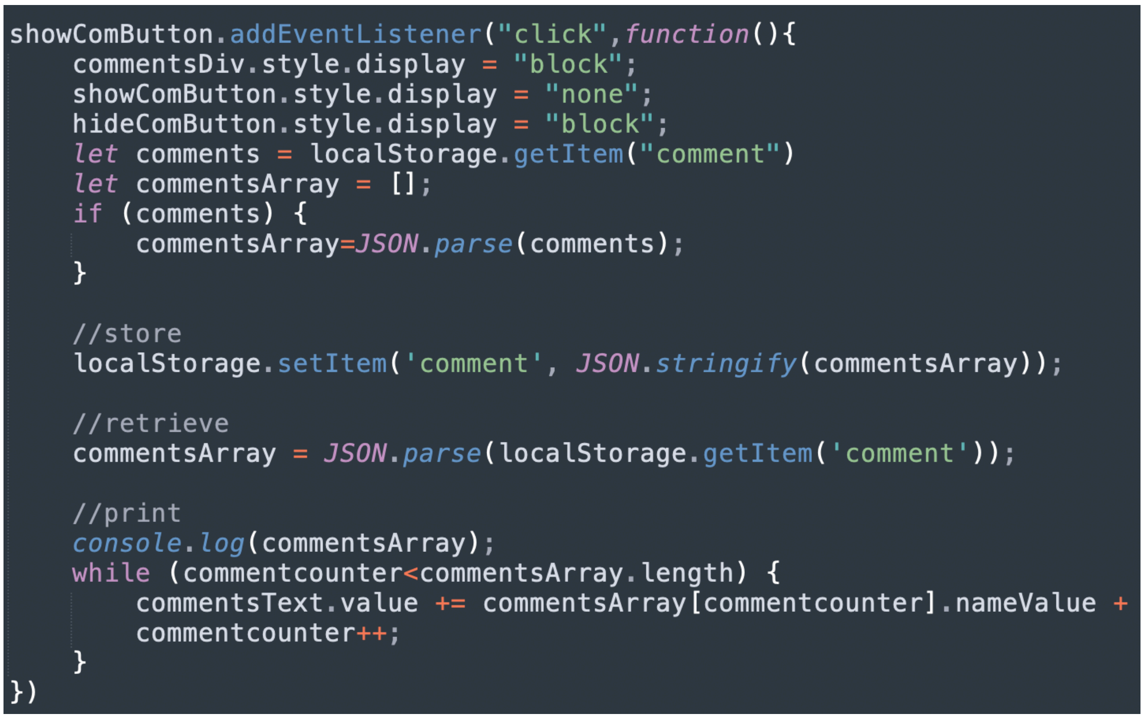
I had some difficulties with the function to edit the chord. Originally, the path property of each chord object was a function that drew the chord pattern with the cx and cy values hard coded in. My edit function thus wrote a new path function and assigned it to the "path" property of that particular chord, but this caused my browser to crash repeatedly. After my consult, I changed the "path" property to an array and created a function to draw the fingering. This was useful as I could just add another property to each chord object called "newPath", and listen for mousedown on the paper and draw on the paper if the edit state was true. Once the save button was clicked, each mousedown event's offset coordinates were added as one object to the begChords.newPath array.

I wrote a function printNewBegChord(), which checked if the .newPath array was empty – if not, it reset the paper, then called another function, editBegChord(), which used a while loop to print all coordinates in .newPath. Then, I updated the Next button's callback function to call printNewBegChord() after the original chord was drawn. Thus, if the user had made edits, the callback function would print the original chord shape, erase it, then print the user's edits.

Notes Function

For the notes function, I wanted to allow notes to persist after reload, as it felt pointless to allow the user to write notes that would be erased. I considered serving the website, but decided it was too demanding. Thus, I looked for a different alternative, and found that some methods for storing data on a website was localstorage, sessionstorage, and cookies. Out of the three, I found that localstorage suited my needs. Localstorage works by assigning a key to a string that is uploaded; to retrieve the string from local storage after a reload, the code must provide that same key.

Thus, I started by writing a function to store title and content of the note to localstorage, and then writing a function to retrieve title and content from localstorage. However, this method only saved and printed the most recent note. I decided that notes should be kept as an object with 2 properties, title and note, and notes should be added to an array with .push(). As localStorage only accepts strings, I used JSON to stringify the commentsArray for uploading, then parse it when it is retrieved. Then, I wrote a function that converts the array into notes and prints it to the notes section.



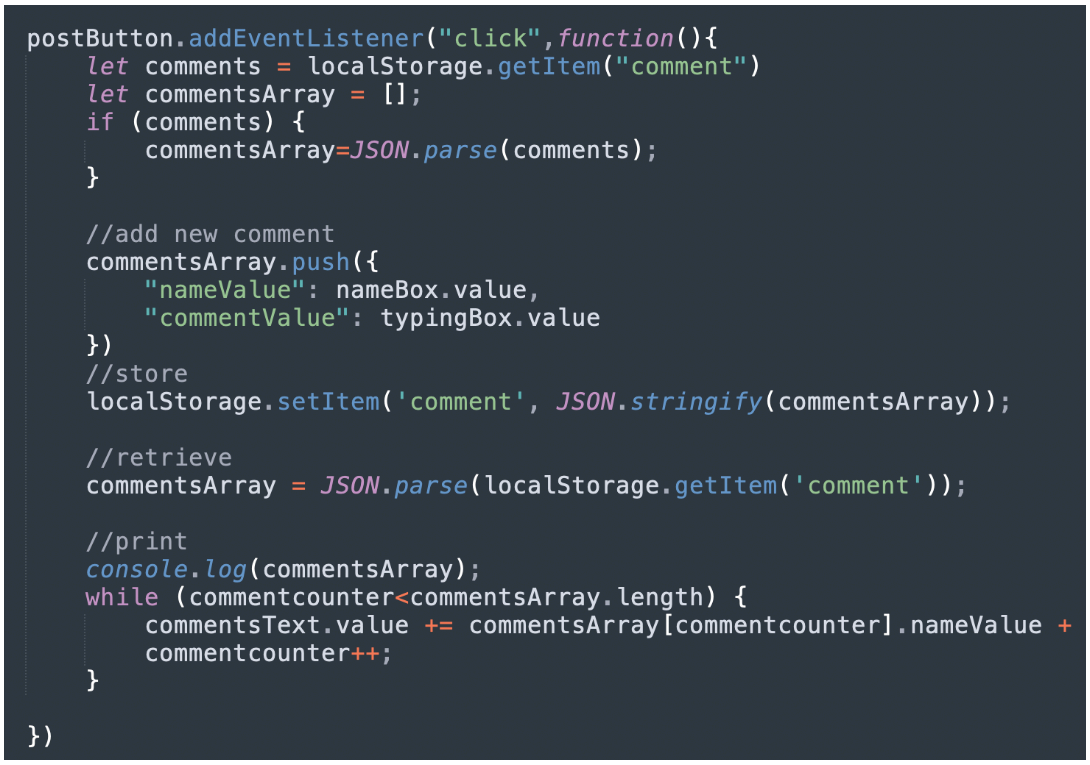
```javascript
showComButton.addEventListener("click",function(){
    commentsDiv.style.display = "block";
    showComButton.style.display = "none";
    hideComButton.style.display = "block";
    let comments = localStorage.getItem("comment")
    let commentsArray = [];
    if (comments) {
        commentsArray=JSON.parse(comments);
    }

    //store
    localStorage.setItem('comment', JSON.stringify(commentsArray));

    //retrieve
    commentsArray = JSON.parse(localStorage.getItem('comment'));

    //print
    console.log(commentsArray);
    while (commentcounter<commentsArray.length) {
        commentsText.value += commentsArray[commentcounter].nameValue +
        commentcounter++;
    }
})
```

The post button has a callback function that retrieves the commentsArray, uses.push() to add the new note, uploads it back into local storage, and retrieves it once more, such that it is printed to the notes section.



```javascript
postButton.addEventListener("click",function(){
    let comments = localStorage.getItem("comment")
    let commentsArray = [];
    if (comments) {
        commentsArray=JSON.parse(comments);
    }

    //add new comment
    commentsArray.push({
        "nameValue": nameBox.value,
        "commentValue": typingBox.value
    })
    //store
    localStorage.setItem('comment', JSON.stringify(commentsArray));

    //retrieve
    commentsArray = JSON.parse(localStorage.getItem('comment'));

    //print
    console.log(commentsArray);
    while (commentcounter<commentsArray.length) {
        commentsText.value += commentsArray[commentcounter].nameValue +
        commentcounter++;
    }

})
```

Conclusion

If I had more time, I would have used the same code from the Notes Function to upload the user's chord edits to local storage. This would have given the user greater personalisation over the website without having to serve it. A more minor addition would be radio buttons that allowed the user to switch between printing circles and barres.