Final Project Documentation

To start the creation of this application, the following steps must be taken:

1. Open the file path in Visual Studio Code in which you want to store the application

2. Open a new terminal and enter the following initialization lines:

```
> npx create-react-app billieeilish
> cd billieeilish
> code .
```

These lines in that order will create a new project folder named **billieeilish** and populate it with all the base assets for a React app, then will navigate to the **billieeilish** folder, and lastly will open a new window instance of Visual Studio Code that is populated with the React assets that were just downloaded.

3. Next, the project requires Tailwind CSS:

```
> npm install tailwindcss postcss autoprefixer
> npx tailwind init -p
```

These lines will install tailwind css capabilities to the application and initialize them for use. A file is created during this process named "**tailwind.config.js**"

4. Navigate to the **tailwind.config.js** file and add the following string into the content section:

```
content: ["./src/**/*.{js,jsx,ts,tsx}"],
```

This line allows the application to access and utilize any tailwind css styling added to the class names later on in the project.

5. Navigate to the **index.css** file and add the following rules at the top:

```
@tailwind base;
```

@tailwind components;

@tailwind utilities;

These lines inject Tailwind's base styles, component classes, and utility classes into the application.

6. Back in the terminal, the following line is ran:

> npm install react-tooltip

This adds tooltip functionality to the referenced sections in the project.

7. After all the dependencies are added, we can minimize the amount of assets in the sidebar menu by deleting what we won't use:

favicon.ico

logo192.png

logo512.png

App.test.js

logo.svg

reportWebVitals.js

setupTests.js

8. This will throw a couple of errors from the files that still reference the deleted assets. On the following pages, deleted these lines to solve the errors:

In **index.js**, delete the **import reportWebVitals** line, both of the **StrictMode** lines, and the entire **commented section and the reportWebVitals** call at the bottom.

In **App.js**, delete the **import logo** line at the top, and the entire **header** from the return inside the function.

9. In the terminal, enter the following install commands to allow the Material UI components to be used:

```
> npm install @mui/lab

> npm install @mui/material @emotion/react @emotion/styled

> npm install @fontsource/roboto

> npm install @mui/icons-material
```

10. In the <head> section of the index.html file, add the following lines to allow custom fonts and access to the Material UI icons:

```
 <link rel="preconnect" href="https://fonts.googleapis.com">

 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link
href="https://fonts.googleapis.com/css2?family=Gabarito:wght@400..900&display=swap"
rel="stylesheet">

<link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&display=sw
ap" />

<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons" />
```

11. In the **tailwind.config.js** file, add the following custom items to the **extend** section:

```
fontFamily: {

        "gabarito": ['Gabarito', 'sans-serif']

 },

backgroundImage: {

        'billiebg': "url('../public/images/other/onstage.png')",

},
```

These customizations allow us to use a custom font from Google fonts, and to include an image as a background element later on.

12. Finally, in order to change the favicon on the browser tab to our project logo, navigate to the index.html file, locate the **link** line near the top, it should be line 5, and change the image reference to our image file, as follows:

```
<link rel="icon" href="billie.png" />
```

Index.js

After initialization, I began setting up my suite of pages and their routes, I wanted a separate page for each component to show the functionalities of each apart from the others. I set up a **pages** folder inside the **src** folder, and added **Home.js, About.js, Career.js, Albums.js, Music.js,** and **Contact.js.** For each page, I started by creating a blank template and including an **export default [page]** command at the end so I could set up the page routing. The routes are set up in the **index.js** file. At the top of the page, the following imports are called:

```
import React from 'react';

import ReactDOM from 'react-dom/client';

import { BrowserRouter, Routes, Route } from 'react-router-dom';

import Layout from "./pages/Layout";

import Home from "./pages/Home";

import About from "./pages/About";

import Career from './pages/Career';

import Albums from "./pages/Albums";

import Music from "./pages/Music";

import Contact from "./pages/Contact";

import "./App.css";

import './index.css';
```

The imports bring in React functionality and the BrowserRouter and Routes functionality for React. From here, I used the following function to create the routes for the pages and call the Layout file to access the styling for the navigation menu:

```
export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="about" element={<About />} />
          <Route path="career" element={<Career />} />
          <Route path="albums" element={<Albums />} />
          <Route path="music" element={<Music />} />
          <Route path="contact" element={<Contact />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}


const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

These lines establish the route paths, identify the **Home** page as the index page, and calls the **Layout** file as the physical representation of the routes.

Layout.js

The **Layout.js** file then uses **Link** components to provide a physical link between the pages. These prevent a full page reload, unlike a traditional <a> tag, which is more efficient for single page applications. It also utilizes **useLocation** and **useState** hooks. The **useLocation** hook gives access to the current URL location, which in this case determines the active route for highlighting the active link on the nav bar. The **useState** hook in this case determines whether the hamburger menu is open or closed by using **isOpen.**

The **isOpen** functionality tracks whether the navigation menu is open or closed, via a **true** or **false** read. **setIsOpen(!isOpen)** toggles the open state for the function. If **isOpen** is set to true, clicking the hamburger menu button will toggle the state to false, and vice versa. There is then a special **className** for the nav bar that uses **{`${isOpen ? "isOpen" : ""} [Tailwind styling here…]`}** to style the nav bar separately for when the hamburger menu is open. It also changes the SVG icon for the menu to a close button when the menu is open.

The **isActive** function checks the current URL path to see if it matches the given URL path. This check will be used for styling to highlight the current active menu link in the nav bar. The **Layout** component also uses a **prop** to pass the title of the page down to the correct location in the nav bar. This allowed me to ensure that all pages have the same title as a default, but also leaves room to me to change them individually if it was needed.

I did have a few issues implementing the hamburger menu at first. When I initially finished the code for it, the hamburger menu would appear, but when clicked, the entire nav bar would be replaced by a blank white space. I researched my problem online, and eventually copied and pasted the code where I suspected the issue was into Copilot to see if it could identify the exact area where I had messed up, in case it was a syntax issue. After a few trial and error attempts at debugging, I eventually realized that I had accidentally set the default menu state to "hidden" when the hamburger menu was opened, instead of when it was closed. Once I removed that typo, the menu worked perfectly.

Footer.js

The **Footer** component is passed to every page by utilizing the self-closing **<Footer />** component tag. Each page imports the **Footer** component, which means each page has a consistent footer across the entire site application. The **Footer** component itself utilizes nested divs to create the icon links that the user sees. The entire content is included in a **section** div. The top row of thinks are traditional **<a>** hyperlinks with text link buttons. These lead to various external sources that were used to populate information for the site. The bottom row of links are also **<a>** tags, however these utilize **<span>** and **<svg>** elements to create social media icons for the user to choose from. This creates a more minimal, cohesive look to the site, and minimizes the amount to text the user needs to read. The icons themselves are pulled from an online database using a **<path>** tag and a unique code for each icon. The **Footer** function is then exported for use in all the pages.

<u>Home.js</u>

The **Home.js** page is the only page that does not have a unique component. This page establishes the theme and color schemes for the design of the application. The design includes dark blues, pops of bright orange, a deep gray, and white. These colors emulate the color scheme for Billie Eilish's newest album, Hit Me Hard and Soft. There is a brief introduction statement that explains what the site is used for, and then the **Footer** component is called to finish off the page.

<u>About.js and Accordion.js</u>

The **About.js** page showcases the **accordion component**. This page starts by importing the React components, the **Footer** component, and the **Accordion** and **accordionData** components. When the **Accordion** component is called, it pulls information from the array located in the **accordionData.js** file and iterates through each object in the array to create the necessary content. This uses the index of each array object as a unique key for each item being iterated, and pulls the data in to display on the web page. The array in **accordionData.js** contains a title and the content for each section of the **accordion** component.

In the **Accordion.js** file, a **useState** React hook is used to tell if the given **accordion** section is open or closed. The **isOpen** variable, as before, tracks whether the **accordion** component is open or closed. By default, it is set to closed by using **setIsOpen,** with a **useState** of **false.**

The title and content information are passed into the **accordion** as **props** from the array. The **accordion** itself is styled using the color scheme set up on the **Home** page, and has a hidden photo of Billie Eilish that appears as a background when each **accordion** element is opened. Another **setIsOpen** function is used in the **accordion** to set styling for the open and close buttons. By default, the button is set to the **+** style while the accordion is closed. When the function tracks the accordion being opened, the button switched to a **–** style. The content information from the array is passed as a **prop** into the **accordion** when the function opens the chosen accordion section.

The only issue I had on this page was a matter of styling. I wanted to set the background image inside the accordion elements, but I had no idea how to do that in React or tailwind. I did some searching around online and tried a few methods I found. After some trial and error, the method that eventually worked for me was adding a section into the **extend** category in the **tailwind.config.js** file that set the image as a custom css element. I then was able to call that custom element in the **className** section of the div tag for the accordion interior.

Career.js and Timeline.js

The **Career.js** page runs its function to call the **CustomizedTimeline** component and the **Footer.** It imports its functionality from React, the **CustomizedTimeline** component file, and the **Footer** component file. The **Timeline.js** component comes from **Material UI.** This functionality was installed separately as described above. The **Timeline** utilizes multiple **Material UI,** or **MUI** components to create a cohesive styled timeline of events.

The timeline itself is set to an alternating style, which has each milestone switching sides of the timeline. Each milestone on the timeline is comprised of a **TimelineItem** component, **TimelineContent, TimelineOppositeContent, TimelineConnectors, TimelineSeparators,** and **TimelineDots.** Material UI utilizes both Tailwind CSS and its own preset CSS elements for styling. I utilized preset colors to stylize the timeline.

My main issue with implementing this component was that I had never worked with it or with Material UI before. I had to do quite a bit of research into how to install and get Material UI functional in Visual Studio Code. I thought I had done it already when I initially added the Timeline page in, but when I connected everything, I got a ton of errors. After researching different issues and asking Copilot what could be wrong, I eventually realized that I had missed a step in installing Material UI. Once I installed the missing piece, the full timeline became functional.

Albums.js and Carousel.js

The **Albums.js** page imports its data from React, the **Carousel.js** file, the **carouselData.js** array, and the **Footer**. The **Albums** function styles the page using the color scheme set up on the **Home** page, and then calls the **CarouselData** and **Footer** components. The **CarouselData** component is populated using a prop that connects the information from the **carouselData** array.

The **carouselInfoArray.js** file contains the **carouselData** array. Each object in the array holds information for the image displayed, the title of the album, and the description of the album that was released. The **Carousel.js** file imports functionality from React, and utilizes a **useState** React **hook** to track which slide is currently open on the carousel, in this case. The

Carousel component receives the **prop**, items, which holds all the information for the albums to be displayed in the Carousel. To manage the **Carousel, currentIndex** and **setCurrentIndex** are used to track the current value of the slide that is visible, and to update the value of the index of the currently displayed slide as it changes. **useState(0)** is utilized to set the index to 0 by default, which shows the first slide any time the page is refreshed. The **nextSlide** function increases the index by one each time it is ran, which will pull up the next slide in line. It then loops back to the beginning of the slides once it reaches the end of the indexed objects. The **prevSlide** function does the exact opposite by subtracting one from the currently displayed index, and loops to the last slide once you get to the first. The div container is styled in such a way that the slides are all shown the same size, in a horizontal row. Each slide is animated with a 300ms transition between slides, and moves the slides to the left or the right accordingly. The items shown in the **carousel** slides are comprised of the information from the array. Each item is assigned a unique key from the index number it holds in the array, and then each item follows the set guidelines for styling and arranges the images and corresponding information accordingly. The images, text, and previous and next buttons are styled using **Tailwind CSS** and the color scheme chosen for the project.

<u>Music.js and the Modal</u>

The **Music.js** file imports the React library, the **Footer** component, and the **MusicList.js** file. It runs the **Music** function, which returns the visual display that the user sees on the music page. This contains the **MusicList** component and the **Footer** component. The div container is styled using Tailwind CSS and follows the color scheme and design set up on the **Home** page. There is also a traditional **<a>** link that sends the user to a Spotify playlist containing the full songs referenced on the page when you click on the visual Spotify logo.

The **MusicList.js** file imports the song card layout and styling from the **SongCard.js** file, and the list of songs from the **musicArray.js** file. The **MusicList** function returns the container that holds all of the card elements that display the songs. In this case, a top 20 list of Billie Eilish’s most streamed songs on Spotify. The container is set to flex so that the individual cards can line up in a row across the page. The function also returns the **SongCard** element, which has passed the songs array as a **prop** to itself, which contains all the information necessary to display the correct information for each song.

The **musicArray.js** file contains the **songs** array. This array holds the image file for the song card, the title, release date, and album to be displayed for each song, a short audio clip file to be played in a **modal**, and a tag line which will display with the song clip to give the user more information on the inspiration behind each song.

The **SongCard.js** file is where all of the information comes together. This file imports the **useState hook** from React to manage which song is currently selected to play playing in the modal. The **SongCard** function takes the songs **prop** array of song information to return later in the function. **currentAudio** and **setCurrentAudio** are used to identify and store which song has been selected to play in the modal. The **showModal** and **setShowModal** are used to control whether or not the modal is currently being displayed. The **handleAudioChange** function takes the index of the song selected by the user based on the button of which card was clicked, sets that song index as the current audio to be played, and activates the modal to be displayed to the user. There is also an error catch to be displayed in the console if there is a problem with this function. The **handleCloseModal** function closes the modal by setting the **setShowModal** value to false.

The **SongCard** function returns the div container that will hold all of the individual song cards to be displayed. Inside of this div is a second div which controls the layout of the individual songs. These cards are mapped using the song objects in the songs array, and the index of each object. The individual card divs are given a key of the index of the array objects, and then styled accordingly using Tailwind CSS. The song card by default displays the image for each song, the title, album, and release date for the song, and then the button that contains the **onClick** function to trigger the **modal** opening.

The **modal** is set to conditionally display only if the **showModal** and the **currentAudio** functions are set to true. If these conditions are met, a modal overlay appears over the screen to gray out the background, and the **modal** appears over the top. The modal displays the song title as the header, followed by a gif of an audio equalizer, a media play bar, the information tag line about the song, and then the close button which handles the **handleCloseModal** function. The audio is set to automatically play as the modal is opened, and the user can control playback using the audio bar on the modal. Audio cuts off if the modal is closed by using the **stopPropogation** function.

This page was easily the most technically intricate page of my entire project. I spent the most time on it, and encountered the most bugs with it. I had the idea of using an array to contain clips of her top 20 most streamed songs, but I wasn't sure if I could get that to work because I had only used audio functionality once before to make a single sound that played when any card was clicked on a previous project. Once I had the array finished, I started with my code from the previous project and tried to implement it into the modal function. At first, I could get the modal to pop up, but no audio was playing. I tried a lot of debug options, rewrote the code multiple times, and consulted Copilot to get a list of potential errors to check for. Finally, I realized that I had accidentally put the code into the wrong section of the file, so it wasn't even being called by the modal function. My next big roadblock was the equalizer. I originally wanted it to be a responsive equalizer that listened to the audio being displayed and updated accordingly. I did some research online and found a method using blobs and started playing with it to teach it to myself. I spent an entire day of work trying to get that functional. At first, nothing would appear above the sound bar. It was just a blank space. I set borders to it to make sure the div was displaying, so I went back in to tweak the code and check for errors. I set up console errors to show me where it was breaking, and I realized that I had accidentally set the show equalizer function to false. I finally got that fixed and it showed my equalizer, but it would play for a single frame and then freeze for the rest of the song. I could never fix that part. After many hours spent on Copilot and various other online sources, I boiled it down to one of two possible issues. Either the functionality of that particular component was outdated and no longer useable, or I needed to figure out a way to implement a constant state update function. I did attempt the state update method. But at this point I really did not understand any of the code that was being suggested to try, and it became more other people's work or AI generated code than my own work. I never could get it functional on my own, and even the other sources weren't working, so I made the decision to scrap the equalizer and add an animated gif instead. If I had more time, I would have tried more routes or looked for a different equalizer component, but I needed to prioritize finishing the project on time.

Contact.js and SignUp.js

The **Contact.js** page imports React, the **SignUp** component, and the **Footer** component. These are called in the **Contact** function to display visually on the page for the user. The **SignUp.js** component imports React functionality. The **SignUpForm** function returns a form for the user to fill out to request to be added to a mailing list to get up-to-date information on current events surrounding the career of Billie Eilish. The form is styled using Tailwind CSS. Currently, submitting the form does nothing except reload the page, because this will not be a live website.