

Tomato Groove is ...

This idea arose as I needed a simple lightweight Pomodoro timer to study with and integration with Spotify to see what song is playing. A to-do list is included to list tasks for the current session!

Development

A bit of brainstorming ...

The main functions I want to include in this project involve 3 sections. A Pomodoro timer, minimalistic Spotify integration UI, and a to-do list. Since I want this project to be lightweight, it will be as minimalistic as I can design it. I am not an expert UI/UX designer, so I guess I will see what I can come up with.

1. *Pomodoro Timer*

- ~~— 25 minutes only~~
- ~~— 5 minute break~~
- Every 4 Pomodoro done prompts a 15 or 30-minute break, and then it loops again?
- ~~— Maybe just make the timer itself a button~~
- ~~— Sound cues on start/stop~~

2. *Spotify Integration*

- Simple UI
 - ~~— Perhaps just album cover, song name, artist name, playlist choosing, and basic controls~~

3. *To-Do List*

- ~~— Basic ability to add, remove, and finish tasks!~~

4. *General*

- ~~— Night Mode — No~~

My idea for the UI is that there will be a header just for the website title and then right below it would be the timer! It will be left side adjusted along with the to-do list just being under the timer. A button next to the timer will disappear when the clock is winding down and then when the timer finishes a pop-up appears with options to start the break. Same thing when the break timer finishes.

Spotify integration will be right adjusted. Button to log in. Then when logged in a right-sided hovering page will pop up that can let you choose playlists, etc... When you minimize it then it will just be a right sided simple UI.

To-do list would mirror the Spotify integration. On start up would be a left-sided hovering page to add tasks and then you can minimize it as well!

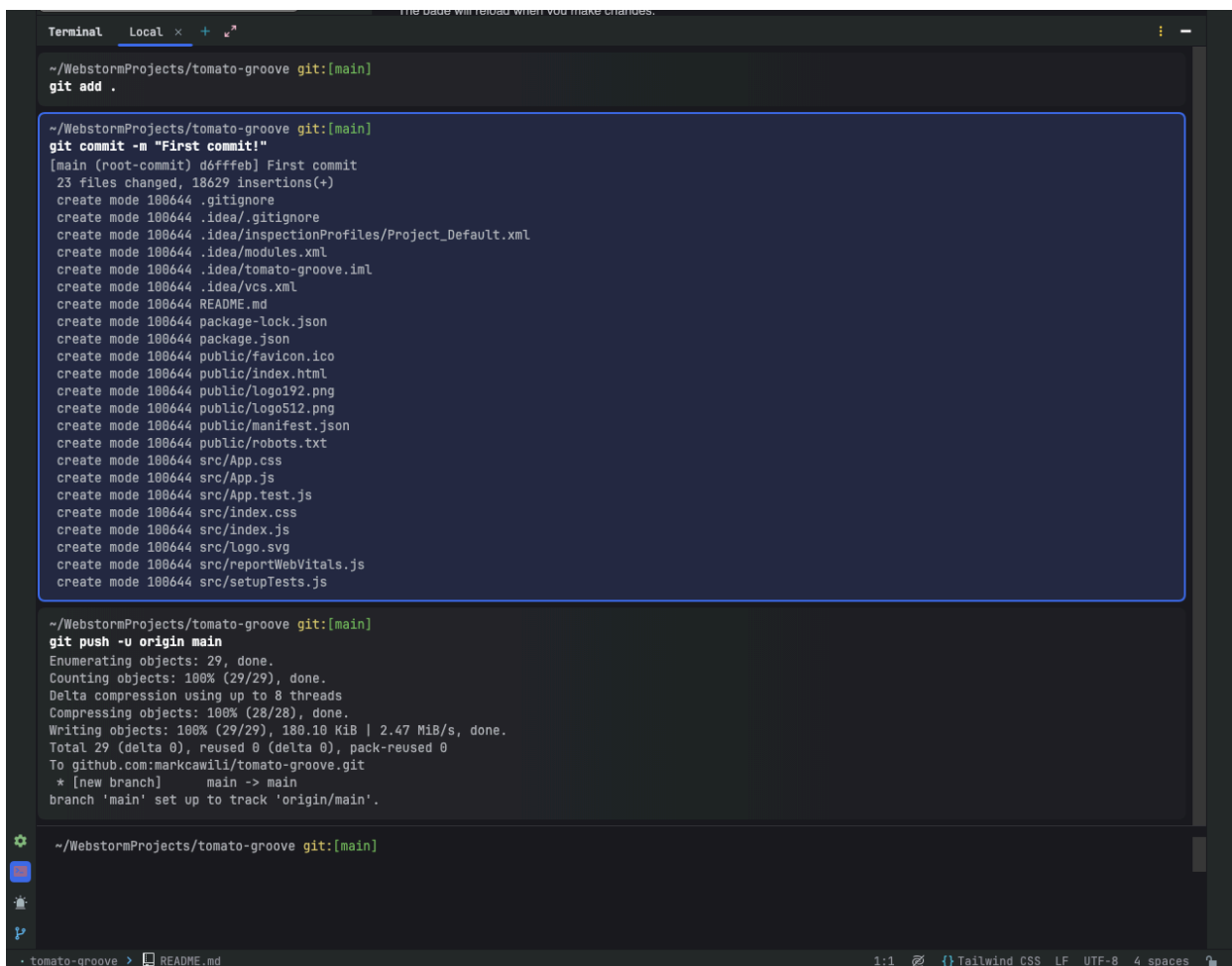
When browsing for colors that can suit the aesthetic of the minimalistic web application I wanted, I came across a couple color combinations that matched the minimalistic and elegant light and dark modes I can try to achieve.

5. Minimalist and Elegant

- <https://huemint.com/brand-3/#palette=ecefee-2a231f-cd2200-767778>

Timeline

Creation of the React application and creation of the repository that will contain my project.
Initial commit pushed!



```
Terminal Local x + ↵
~/WebstormProjects/tomato-groove git:[main]
git add .

~/WebstormProjects/tomato-groove git:[main]
git commit -m "First commit!"
[main (root-commit) d6fffeb] First commit
23 files changed, 18629 insertions(+)
create mode 100644 .gitignore
create mode 100644 .idea/.gitignore
create mode 100644 .idea/inspectionProfiles/Project_Default.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/tomato-groove.iml
create mode 100644 .idea/vcs.xml
create mode 100644 README.md
create mode 100644 package-lock.json
create mode 100644 package.json
create mode 100644 public/favicon.ico
create mode 100644 public/index.html
create mode 100644 public/logo192.png
create mode 100644 public/logo512.png
create mode 100644 public/manifest.json
create mode 100644 public/robots.txt
create mode 100644 src/App.css
create mode 100644 src/App.js
create mode 100644 src/App.test.js
create mode 100644 src/index.css
create mode 100644 src/index.js
create mode 100644 src/logo.svg
create mode 100644 src/reportWebVitals.js
create mode 100644 src/setupTests.js

~/WebstormProjects/tomato-groove git:[main]
git push -u origin main
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 8 threads
Compressing objects: 100% (28/28), done.
Writing objects: 100% (29/29), 180.10 KiB | 2.47 MiB/s, done.
Total 29 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:markcawili/tomato-groove.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

~/WebstormProjects/tomato-groove git:[main]
```

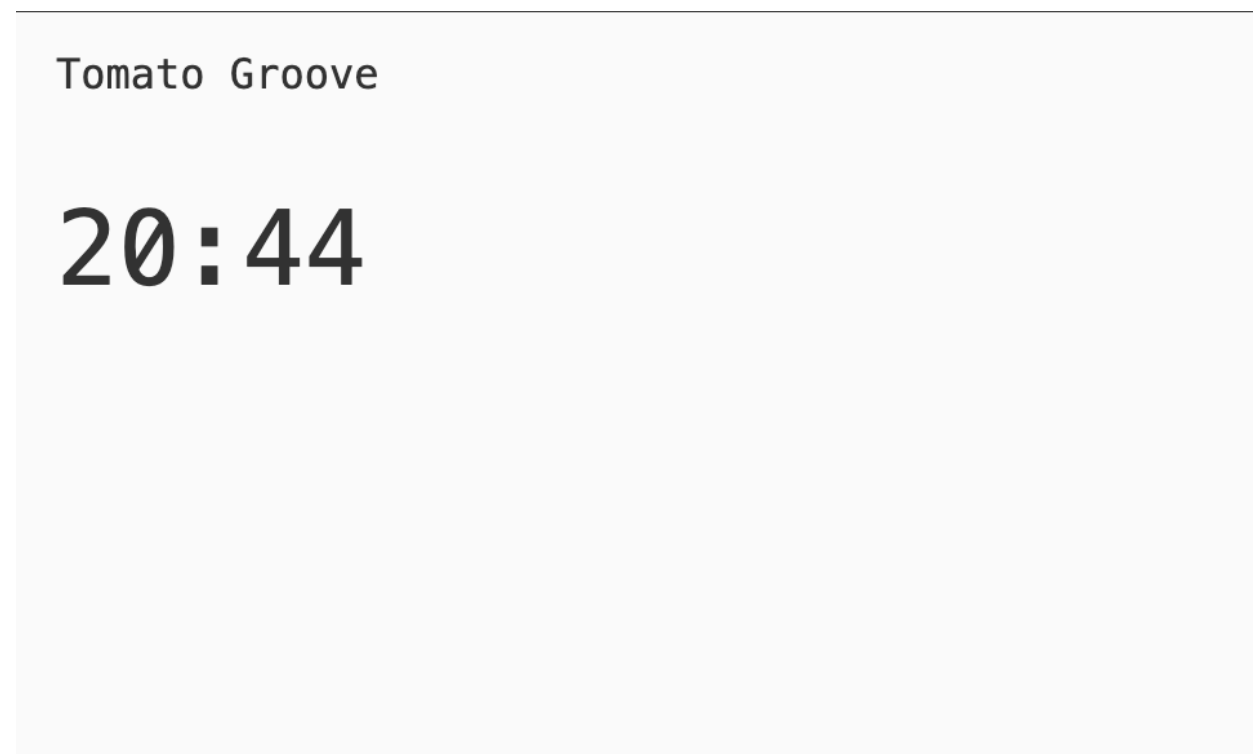
tomato-groove > README.md 1:1 Tailwind CSS LF UTF-8 4 spaces

One of the first things done was to set up my folders for the components, styles, and tests. I then created a Timer.js file which would represent the Pomodoro timer which will be the first component we will be working on. Using the index.css stylesheet, I inputted a [reset CSS tool](#) which clears the default styles on most browsers and will ultimately reduce inconsistencies.

I then implemented the basic timer functionality which handles the logic of a timer starting from 25:00 all the way to 00:00 and the start of a break timer at 4:59. From here on, I am going to add an implementation to specifically only start the timer when a button is pressed and explore the implementation of adding a longer break timer when 4 sessions have been completed.

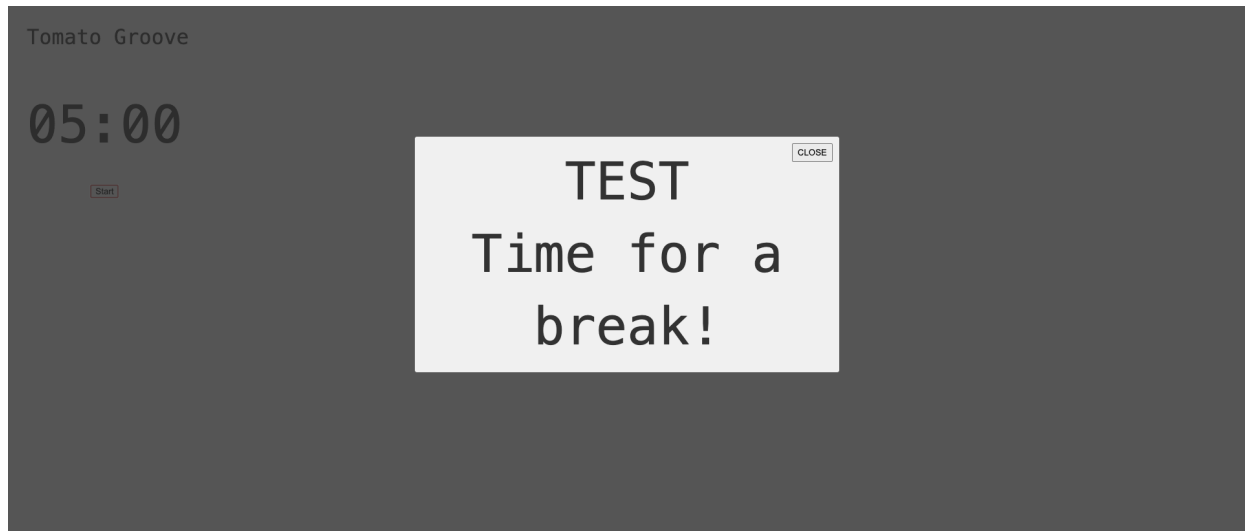
I credit this YouTube video for the initial timer:

📺 How to Build Pomodoro Timer in React | React Tutorial



In my next session, I created a Modal component that would show up when the session and break timers end. I found that to get this to work, I set a showModal useState and initialized it to false. When the timers end withing the useEffect, setShowModal will set the boolean to true which is then passed as the isOpen variable in the Modal component call within the return section of Timer. toggleModal is a function that does the opposite of the timer setting the showModal to true.

In Modal, there is a simple if that returns a null if the Modal is not open aka showModal is false.



Different end of timer messages were set very simply; copying how `isActive` is set for the modal.

I also added a simple timer end sound that plays a bell ring. I found a YouTube video that explicitly calls and imports the sound as an `Audio` object, but I found that it keeps giving an unsupported source error. I looked on Stack Overflow and found a different implementation that imported the sound first, which worked properly.

I then took some time to research some YouTube videos on Spotify integration. I managed to follow a video in setting up Spotify authorization within the website, which gives the user aka me a valid token to use. Now I must think of implementation to get what I would like from Spotify. [Spotify's documentation](#) for authorization proves to be a very useful resource for configuring authorization settings.

I installed, using ``npi spotify-web-api-js`` a wrapper around Spotify Web API that makes it easier to interact with the API itself.

We also want to avoid a concept called prop drilling (`app.player.user...`) which can lead to messy and tightly couple code that we do not want! This kind of code design can lead to breaking of code when we change one small thing. We use context API/React Redux to make a data layer to be able to grab what we want directly.

What we did first was create a `DataLayer.js` which is the wrapper for the App which serves as the data layer for the application. We make `reducer.js` and create the initial state of the data layer as `initalState` which is then passed into the `DataLayer` `initialState` as a prop.

A reducer takes two things, the state of the data layer and the action. The state is how it currently looks and the action is setting HOW it looks. Within this function there is a switch that takes what is pushed into the data layer, essentially listening for actions and does something

based on the action 'heard'. This function is then passed into the DataLayer component as well as a prop.

Note that export const ... is used for constants, non changing functions whereas export default {name} is used for functions!

To get access to the DataLayer, we have a function export const useDataLayerValue which uses useContext and passes in the DataLayerContext.

```
const [{}, dispatch] = useDataLayerValue();
```

Is used to pass in the values we want and uses Dispatch like a gun to shoot at the data layer to update it with values we want. So one of the first iterations we use this is when a user is received from Spotify, we use dispatch to set the user giving it a type because of action.type and its payload which is user. User is popped right into the data layer!

After some time messing around with the annoying height and its application on CSS, I was able to finally find a solution to getting my desired height for the Utility components. I did this by setting the .App as a flexbox and then setting the flex-grow attribute for .utility as 1. This successfully fills out the utility as much as it can to the next div which is the footer.

When I set padding to the player body, it overflowed the div. Searching for a solution on StackOverflow, I learned that by default, content-box model on a display: block element will add padding and height together to determine total height of the element. Border and padding are subtracted from specified height by using box-sizing: border-box.

A great trick for responsive UI from the video was that you set a container to display: flex, and justify its content as space-between. You give it a max-width and it will be confined to that but if the window becomes smaller, then it will shrink to fit!

Using the Spotify API to get Saved Tracks

In App.js, I use the spotify object to call getMySavedTracks for the liked songs and passed in the token as a parameter. With the response we use dispatch and save it into the data layer with type "SET_LIKED_SONGS" and then setting the likedSongs value with response which is the JSON data that we get.

In reducer.js we add in likedSongs with value array as a value in initialState and add the "SET_LIKED_SONGS" state. In Body.js, we import the useDataLayerValue method from DataLayer.js and use it as a useEffect type method with likedSongs as the grabbable parameter. Setting a console.log(likedSongs) shows that we got what we want.

An issue with the overflow of songs and how it is being displayed

Unfortunately, I did not reflect right after for this issue as I had been sidetracked but from what I remember, it was all about putting the songs in its own div and having that div scrollable. That div's overflow-y, in relation to its parent div was also hidden.

Getting the playback to work

To start, I use npm i react-spotify-web-playback, which is a library I will be using to allow us to play songs directly from Spotify. I import, into the Footer.js, the SpotifyPlayer component from the React Spotify library we are using. When I tried loading the player, I get a “**This functionality is restricted to premium users only**” “**Invalid token scopes**” issue. I searched up the library on Github and found documentation on the README about the scopes needed, so I added them in! This library makes it easy to incorporate a simple player into my web app! For now, I have left my original implementation in the code but commented out as it was good practice!

<https://github.com/gilbarbara/react-spotify-web-playback/tree/main>

I was able to get liked songs that are clicked to play but there is a bug where it does not play at first click and then once it is, it can be played. There is also no shuffle at all. I will be going back to this in another session.

An issue with labels and actions

After a couple of hours of work, I was able to get animated closing for the list and body components as well as a responsive layout which I will go over again later. I was having an issue with the list items and the more settings icon. When I click anywhere on the same row the whole thing strikes out which is not what I wanted. The fix was separating the label to ONLY the checkbox and wrapping the whole item as a div to represent it as a list-item. This separates the functionality!

Idea:

Button to switch between Liked songs and playlists

Initial message to show what app is about and how to use it