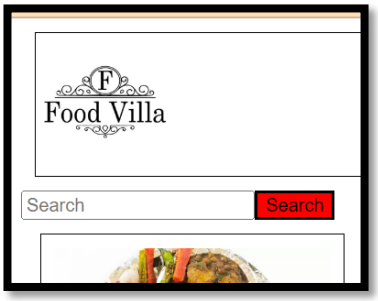
Applying Inline CSS :

Till now we have applied CSS externally, we can create inline CSS for any element by including an attribute named style and then passing a JS object to it inside curly braces. In the below example, I am making the background color of our search button red :

```
const searchBtnStyle = {
    backgroundColor: "red",
}

return (allRestaurants.length === 0) ? (
    <Shimmer />
) :
    (
        <>
            <div className="search-container">
                <input ···
                </input>
                <button style={searchBtnStyle} className="search-btn" onClick={() => {···
                }}>Search</button>
```



You can also directly mention the backgroundColor attribute inside the style attribute without creating a JS object, but this not a good way of writing CSS because

- We can't reuse the styles
- It becomes a heavy task for the browser too because all of the CSS is hardcoded

Applying CSS using external libraries like Material UI, Chakra UI, SASS etc :

- **Cons :-**
  - Including any CSS library will increase our bundle size
  - We lose a lot of control over how our elements look i.e. customizing it becomes difficult
- **Pros :-**
  - Reusable
  - Easy to code
  - Takes care of responsiveness

Applying CSS using Styled Components:

There is also another framework to help developers write CSS called Styled-Components. Just like we are able to write HTML-like code inside javascript files in React in the form of JSX, this framework helps us to write CSS-like syntax inside our JSX. If you want to learn use :- Link1, Link2.

- **Pros :-**
  - Reusable
- **Cons :-**
  - Can be unreadable because of the long codes and no moduling

Tailwind CSS Framework :

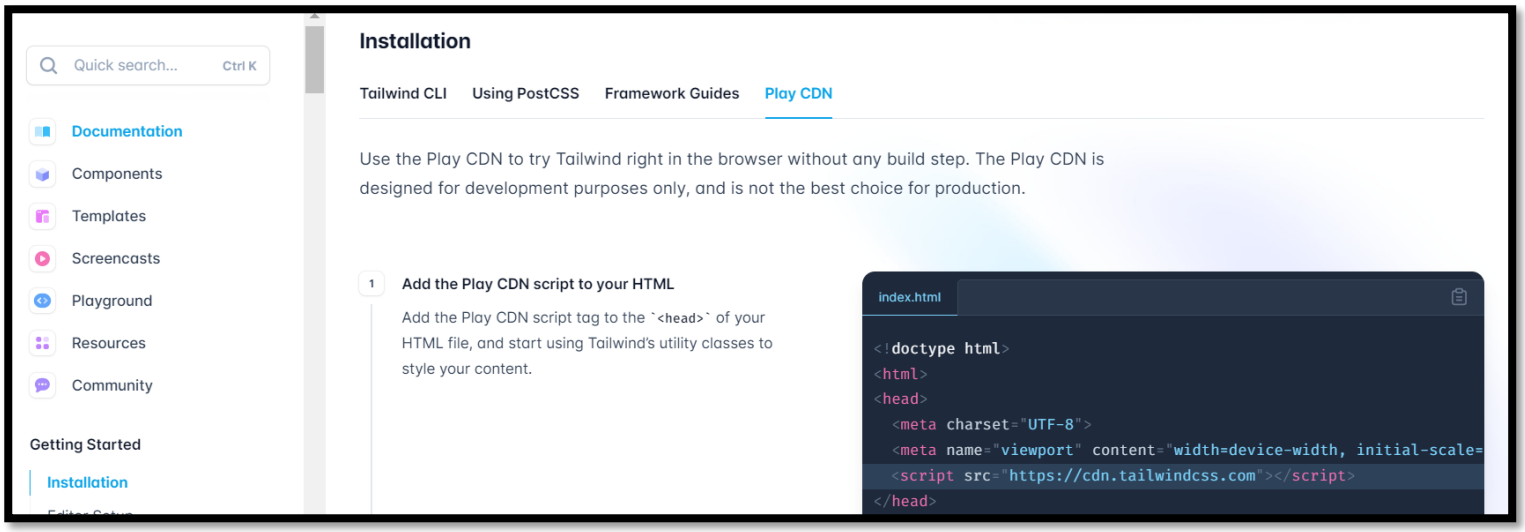
Tailwind is a **Open Source CSS Framework.**

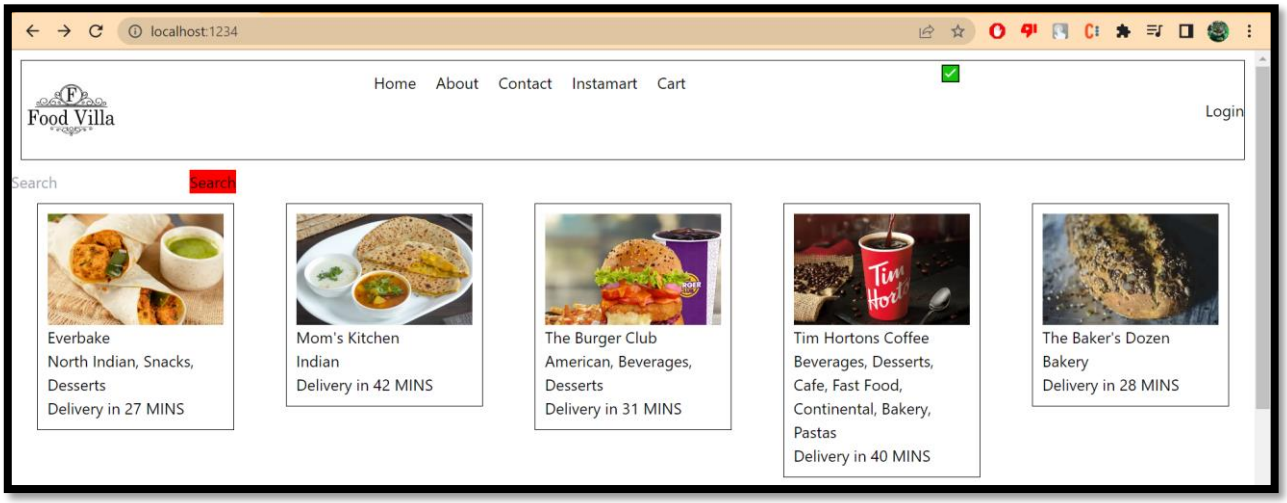
Pros :

- While writing normal CSS, we had to toggle between the CSS file and the JS files to see which class or id do we have to put our CSS in. Tailwind says that we can instead write the CSS on the go (i.e. in the same file)
- Reusable because it comes with a lot of prebuilt classes
- It consumes lesser bundle size because it offers us just minimal CSS to ease up our pain
- Flexible UI i.e. easily customizable.
- Easy to debug (Reason mentioned at last)

Easier version of Integrating Tailwind into our project :

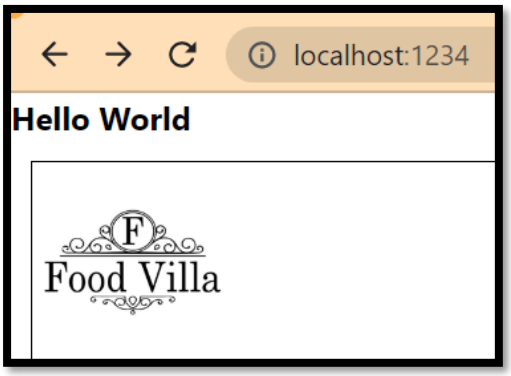
We can install it using CDN links in out index.html file. Remember that at the end of the day all these CSS frameworks/libraries are packages. However, after you include the CDN link, you will see that it overwrites every CSS element and your website will look very different. So, we have to write every CSS in Tailwind CSS.
Here, every class you write for an element is a CSS style.





- **Making a text bold :**

I have added a h1 tag before the root div and now to make it bold by adding the class (if in HTML file) or className (if in JS file) as **font-bold.**

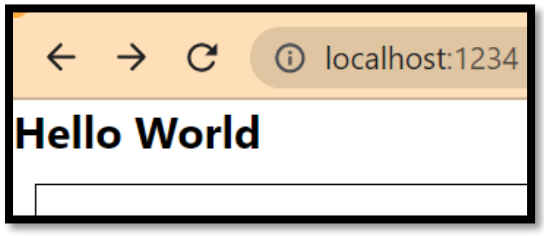



```
    <link rel="stylesheet" href="styles.css">
    <script src="https://cdn.tailwindcss.com"></script>
    <title>Namaste React</title>
</head>

<body>
    <h1 class="font-bold">Hello World</h1>
    <div id="root">
        Not Rendered
    </div>
</body>
<script type="module" src="./src/App.js"></script>
```

- **Making a text extra large :**

Use class as "**text-xl**"



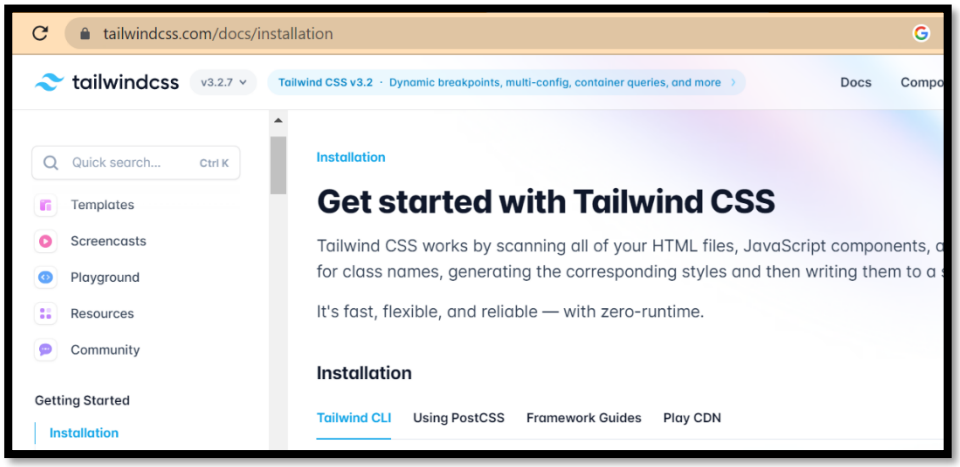

```
<body>
    <h1 class="font-bold text-xl">Hello World</h1>
    <div id="root">
        Not Rendered
    </div>
</body>
```

- **Making it more extra large :**

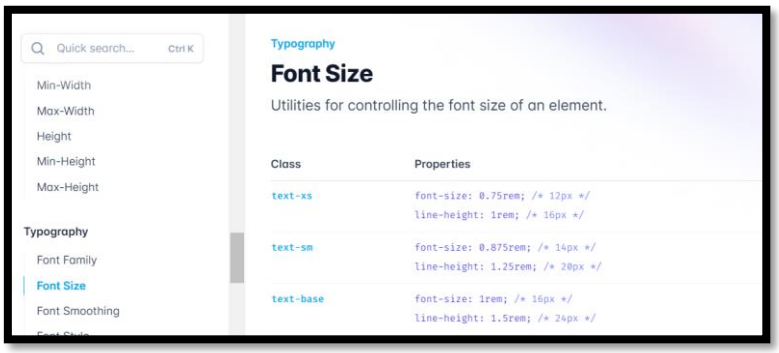
Just add a numerical value before xl -> It will make the text that times bigger like : **text-2xl**, **text-3xl**

Knowing where to find the necessary class names :-

You have to go to the Tailwind Docs and just search whatever you want.

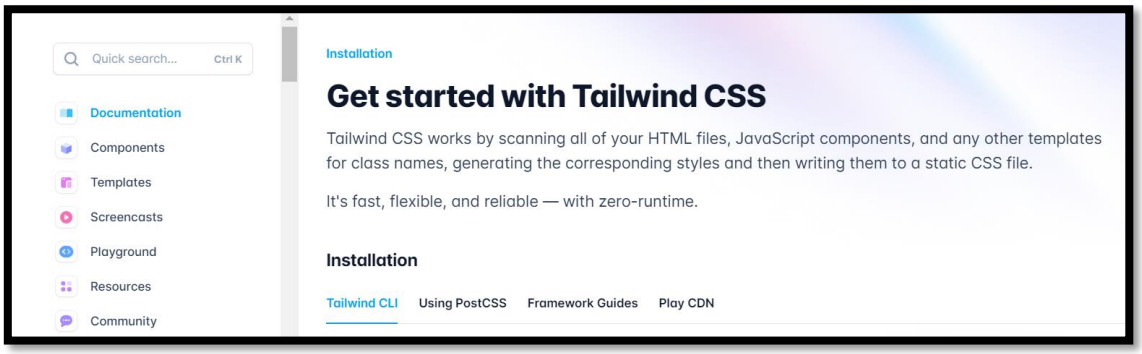


There's a Search area (with a shortcut key as CTRL+K). For ex:- I searched for text enlarge and it led me to thr Typography section, from where I went to the sub-section Font-size and I got all the above class names like "font-lg", "font-sm", "font-xl" to change the size of the text.



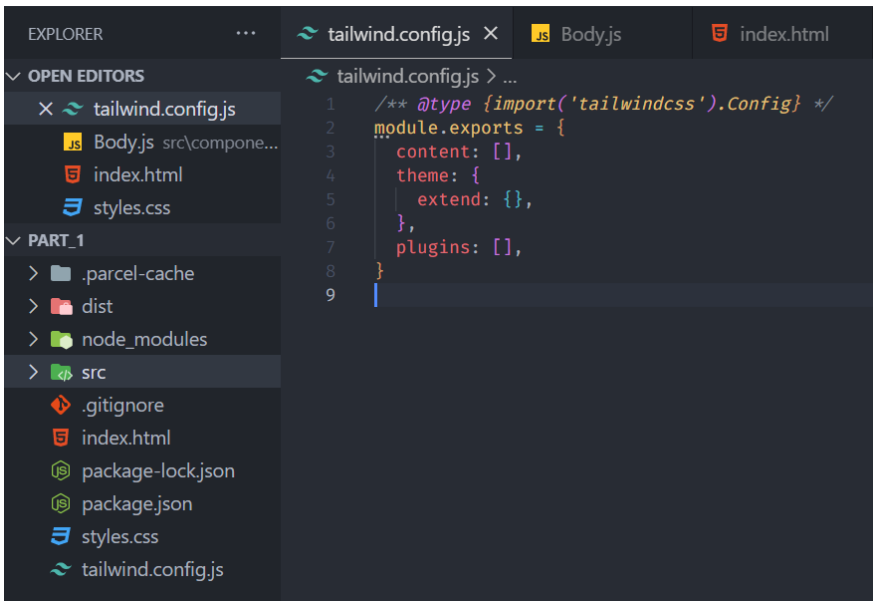
Integrating Tailwind CSS into our project as a package:-

- Go to the Home Page
- Click on Get Started

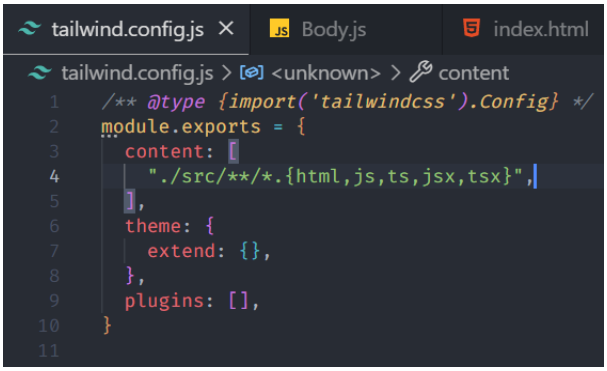


- Click on Framework Guides because we want to install it as a package/framework
- Click on the Parcel one

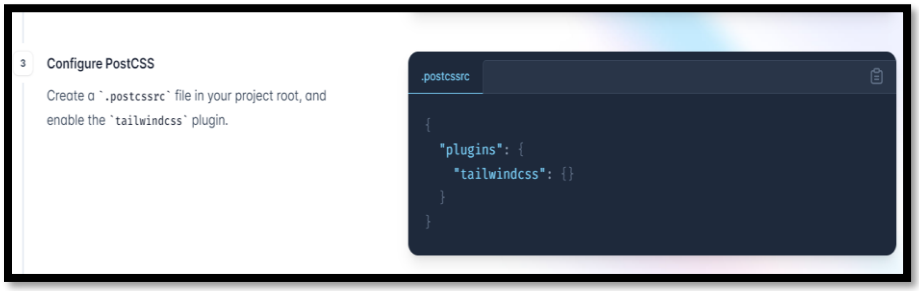
- Now, you will see that it is asking to install Tailwind along with PostCSS as devDependencies because when you build the React app for production, the Tailwind CSS will get converted into normal regular CSS by PostCSS (that's why we are also installing PostCSS because browser does not understand the Tailwind syntax), so we do not need the Tailwind package anymore in the production build.
  The command is something like : **npm install -D tailwindcss postcss**
- Next, it will ask to run a command to initialise the Tailwind like : **npx tailwindcss init**
  This command will create a new Tailwind config file like :

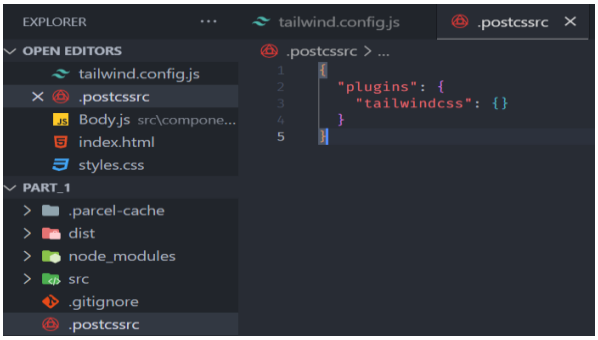


In the content attribute, we have to mention the files in which Tailwind will be used. To do this, we can just write the thing written in the docs which says that all the files ending with js, html, tsx etc. inside the src folder will be using tailwind and after applying that our config file will look like this :-

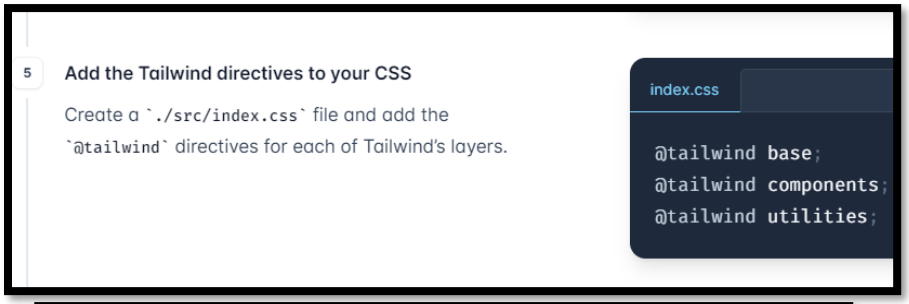


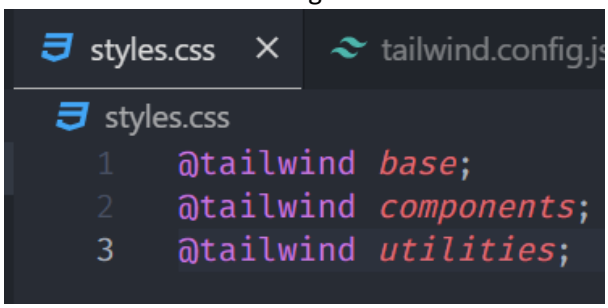
- We also have to make a new postcssrc file in our root directory and configure it accordingly. This tells our bundler (here parcel that we will be using Tailwind in our project, so while bundling files up (i.e. while making the production build, please compile our tailwind and convert it into regular CSS with the help of PostCSS.

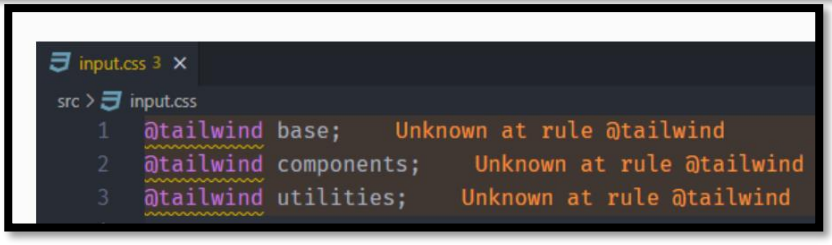




- Earlier we used to write CSS in our index.css file, but now, we just have to write the below things in our file :-







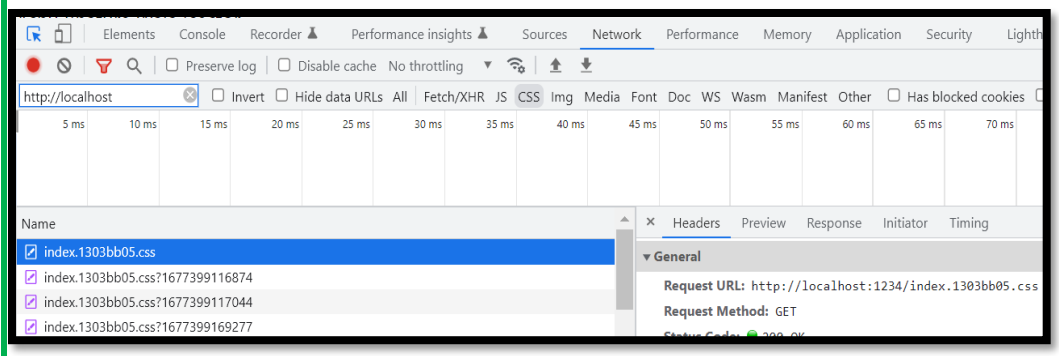
These 3 things simply means that we will be using the Tailwind classes from its base, components and utilities. I also encountered 3 warnings at this step like beside image and so I installed 2 plugins :- PostCSS language Support and TailwindCSS Intellisense.

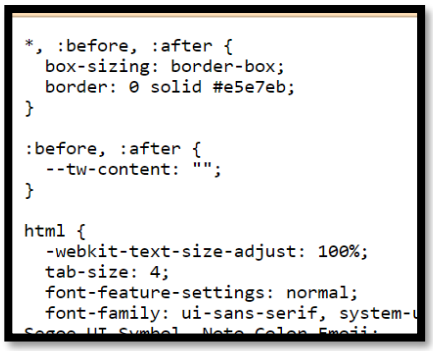
- After this we are ready to use Tailwind CSS classes

Now, you will see that all the things in our app is pretty messed up. First of, we are changing the Header component. Watch the video from 1:40:00 to 1:58.
Then we style our Search button -> watch video from 1:59:00 to 2:04:30.
Then we style our restaurant cards -> watch video from 2:04:30 to 2:21:06.
Then we create hover effect on our search button -> watch from 2:21:30 to 2:25:20
Then we create a focus effect on our input box -> watch from 2:27:45 to 2:29:20.
Then we write media queries -> watch video from 2:30:15 to 2:36:25.

#NOTE :-

- Sometimes Tailwind Intellisense does not provide suggestion. It is most probably a bug. In that case, just press CTRL+Spacebar
- Also, suppose we want to set the width of an element as 200px, but there is no option for doing that automatically. So, we want to give a dynamic value to a Tailwind CSS class. For doing this, we have to use **Square Bracket Notation** and instead of saying like *className = "w-24"* etc. , we have to do *className = "w-[200px]".*
- If you go to the networks tab and filter onlt the CSS files and open the first one then you will see that the file is really small and it includes the CSS of only the classes we used in our project into the production build and not all the classes of the entire Tailwind library.



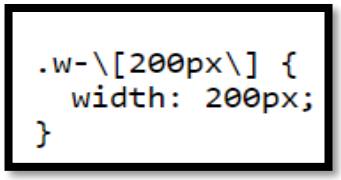


```
*, :before, :after {
  box-sizing: border-box;
  border: 0 solid #e5e7eb;
}

:before, :after {
  --tw-content: "";
}

html {
  -webkit-text-size-adjust: 100%;
  tab-size: 4;
  font-feature-settings: normal;
  font-family: ui-sans-serif, system-u
```

This action is performed by Parcel using the PostCSS and Tailwind library to convert those classes into regular CSS. Also, for the dynamic classes, it creates a new class for us. Ex :- We used a dynamic class for setting width and you can see that class being made separately in the image beside



```
.w-\[200px\] {
  width: 200px;
}
```

- This is the reason it is better to stick to native Tailwind classes, instead of creating new dynamic classes so that Parcel does not create new classes for each dynamic one and will also be consistent.
- This also helps in debugging process because now we do not need to hover from one CSS file to another JS file just to know which class or id some CSS element is referring to.

Cons of Tailwind :

- It has a steep learning curve and compromises the code readability -> watch video from 2:40:45 to 2:44:30