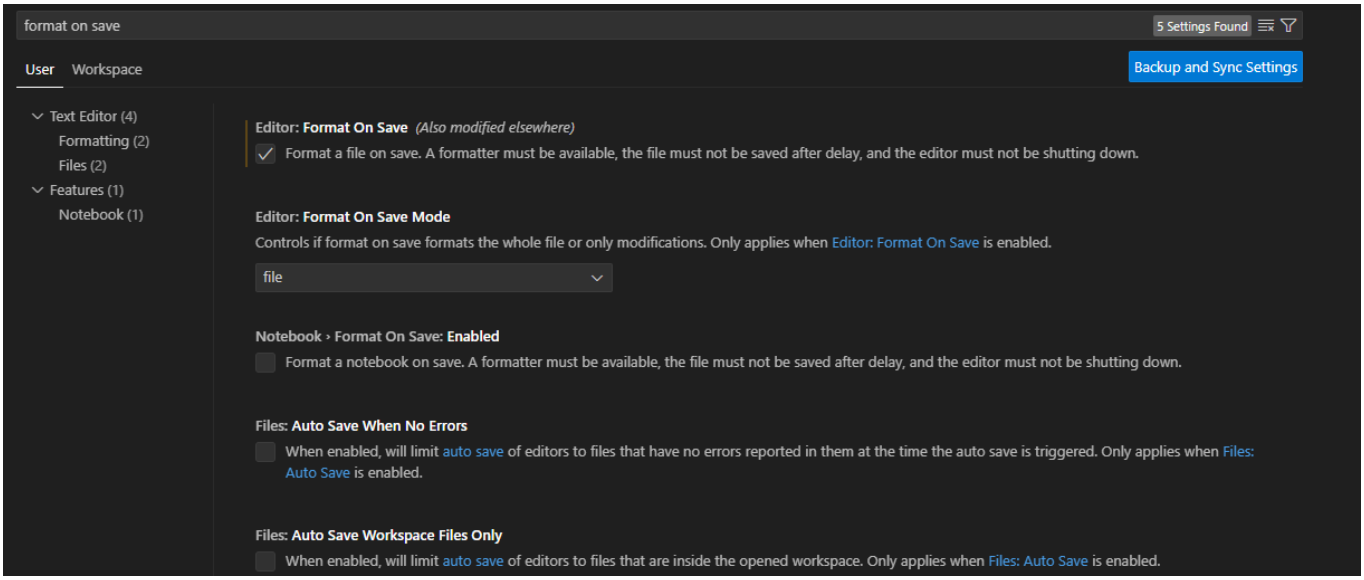
Before starting it is sugessted to add extension for formatting, extension like Prettier.



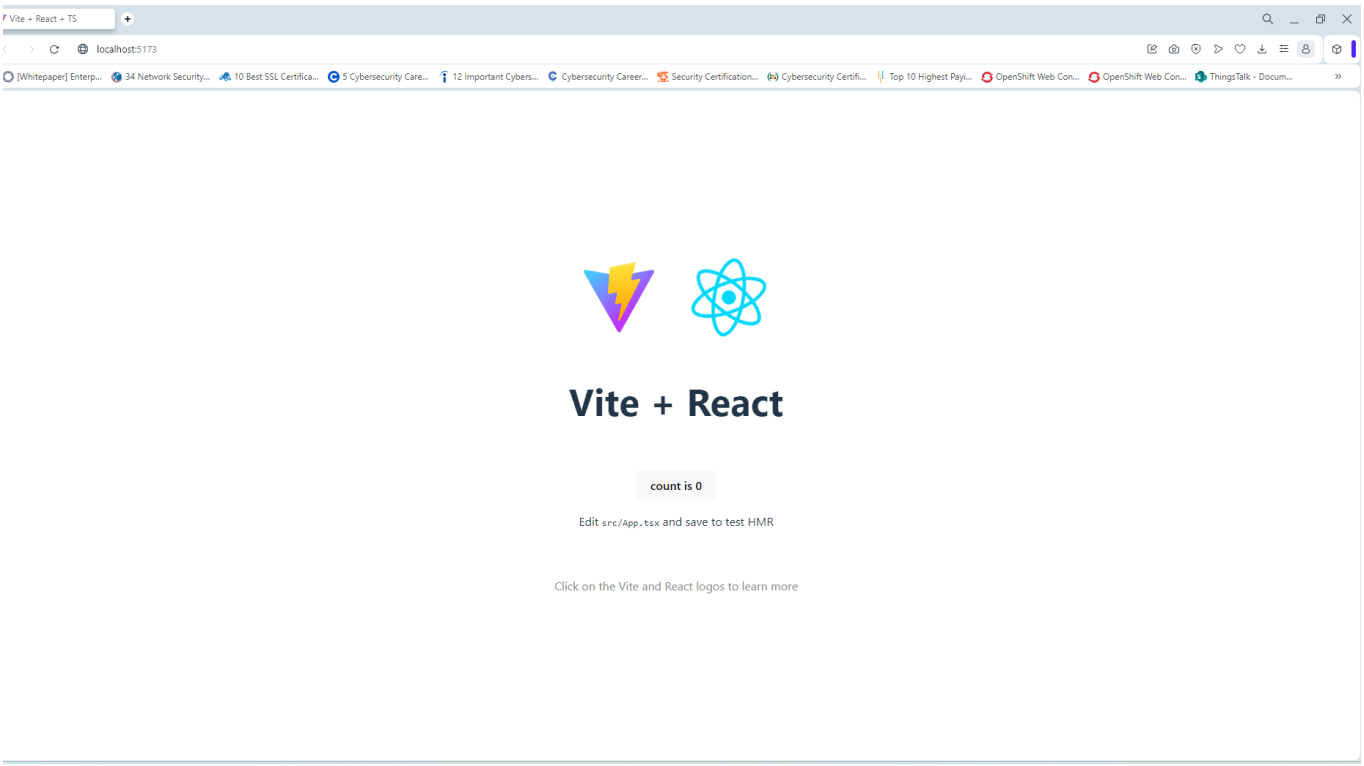
There are two ways to create a React app. We can use the official tool provided by a React team. It's called Create React App or CRA but there is also another tool called Vite. It's much smaller and gives faster bundle.

1. First step is to install Node.js. Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts. You can download it on following link: https://nodejs.org/en .
2. Then you check the version by typing:
   node -v
   After that you start creating a React app using Vite, you can use latest version or specify.
   npm create vite@4.1.0 or npm create vite@4.1.0
   Then you choose framework that you u want React, Typescript, Vanilla. Vue, Svelte or other. Vanilla is a JavaScript without any additional tools. In my example, I choose React. After choosing framework, next step is to choose a language: JavaScript, TypeScript, JavaScript + SWC and Typescript + SWC. I choose TypeScript.

Next step is to position in my project cd react-app and install all third party dependencies using npm install and then run our web server.

`npm install` is a command used with Node Package Manager (npm), which is a package manager for JavaScript. When you run `npm install` in your terminal or command prompt within a Node.js project directory, it reads the `package.json` file in that directory and installs all the dependencies listed in it.

To run my web server I type npm run dev and it launches a web server at this address localhost http://localhost:5173/ and my display currently is showing this:

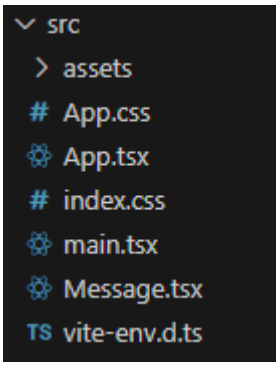



```
Brief overview of files listed in my project:

![[Pasted image 20240320104455.png]]
```

Node modules folder is where all the third party libraries are like React and other tools are installed you will never have to touch this.
Next is public folder where the public assets of my website exists like

images video files and so on. Src or source folder is the source code of my application. In this folder currently I have a component called Appp.tsx. Outside of that is index.html which is very basic HTML template .
In package.json you can find information about the project so name of the project, version and so on.. In tsconfig we can find bunch of settings for telling the TypeScript compiler how to compile code to Javascript. For most of the cases this file doesn't have to be changed unless you are an advanced user. And the last is the configuration file for the Vite. For the most parties you don't have to touch it.

To add a React component, I clicked on a src -> add new file. I named it Message.tsx. Like in the photo:



So extension of TypeScript files should be ts or tsx. Ts is used for plain text Typescript files and tsx for React components.
There are two ways to create a React component using a Javscript class or a function. Function based components are more popular nowdays.
They are more concise and easier to write.
For creating a function, I am going to use PascalCasing.
Pascal casing, also known as UpperCamelCase, is a convention used in programming where multiple words are concatenated together, and each word in the compound word is capitalized except for the initial word. This casing convention is commonly used in naming classes, types, functions, variables, and other identifiers in many programming languages, including JavaScript, C#, Java, and others.

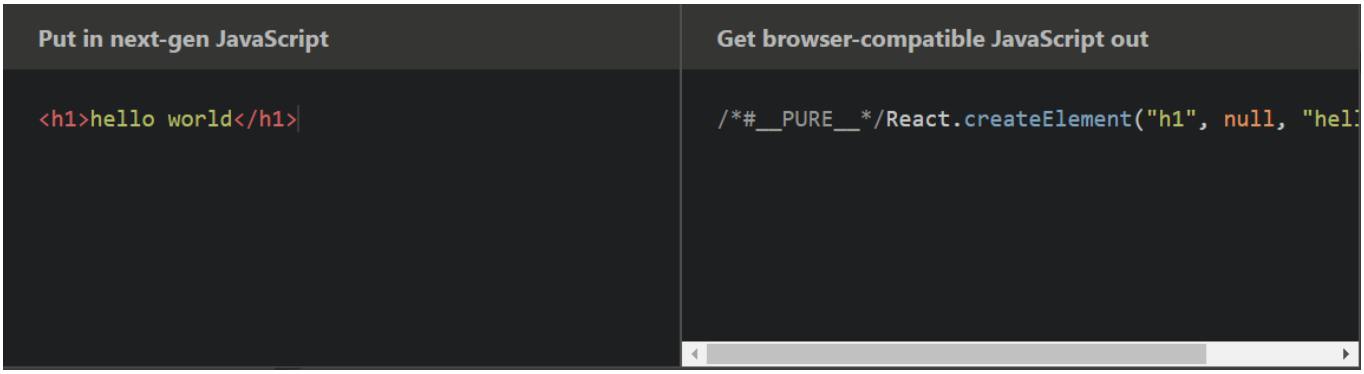
Whenever we use this component we want to render an H1 element with a message like hello world
so here we return an H1 element with hello world.
This syntax is a little confusing at first, it seams like I am writing an HTML code in the middle of the JavaScript code. This syntax is called jsx, which is short for JavaScript XML.
So this code under the hood is going to get compiled to JavaScript.

```
// PascalCasing
function Message()  {
    // JSX: JavaScript XML
    return <h1>hello world</h1>;
}
```

Under the hood, JSX gets transpiled into regular JavaScript code by tools like Babel before it can be executed by the browser. This transpilation process converts JSX syntax into `React.createElement()` function calls, which create React elements representing the UI components.

So if I had over to the Babel https://babeljs.io/ . This is repo where I can see how code gets converted to Javascript
Babel is a toolchain that is mainly used to convert ECMAScript 2015+ (ES6+) code into a backward-compatible version of JavaScript that can be run in older browsers or environments. It is a popular JavaScript compiler and transpiler.

So for example on the left side I wrote jsx code and the display is seen on the photo.



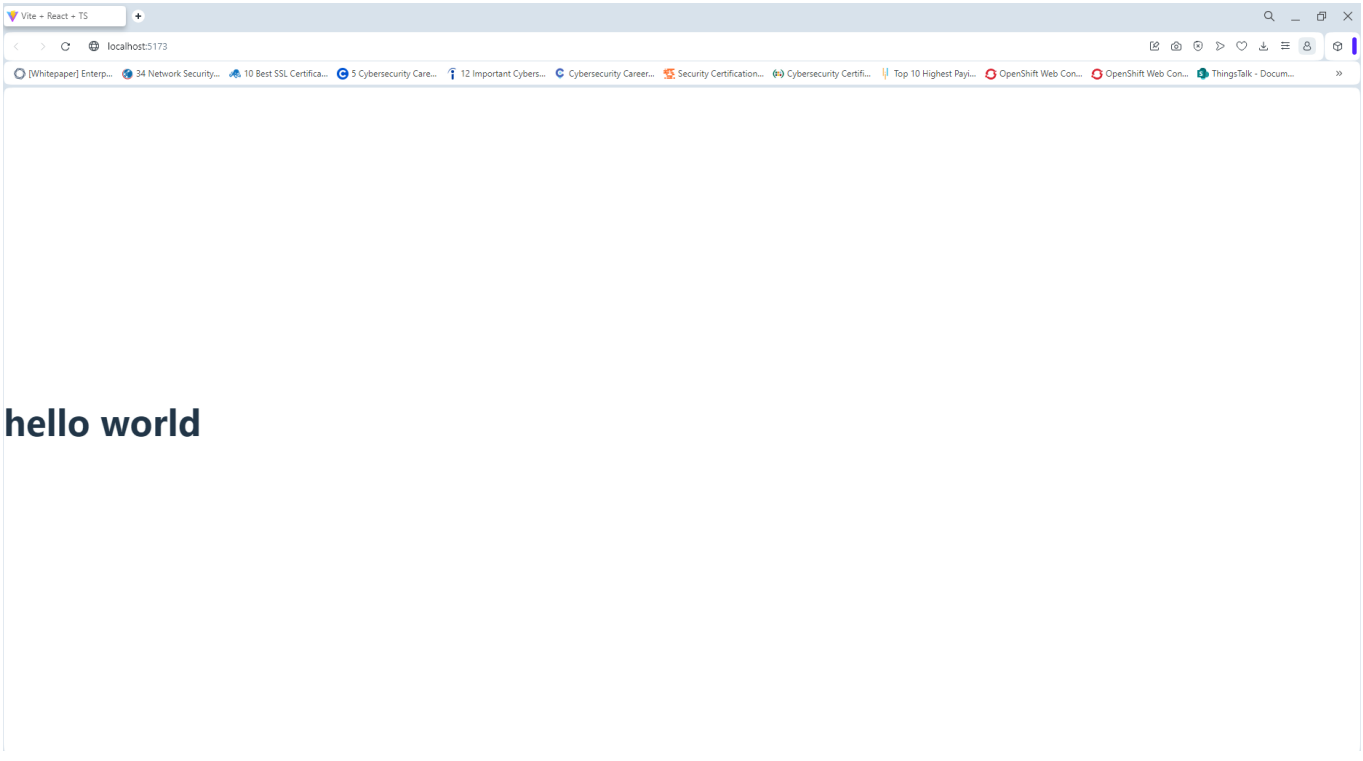
So piece of code eventually gets converted to react.createelement.

So if we want to use it we need to export it as a default object from this Module.

Now I go to the App component and use this new component. I go to App.tsx and delete everything in this file.
First I am gonna create a new component called App.
Now let's say we want to have div and inside the div we want to have our message component.
So next step is to import message component. Now we can use this component like regular HTML elements.
We can use self closing syntax which is more consise.

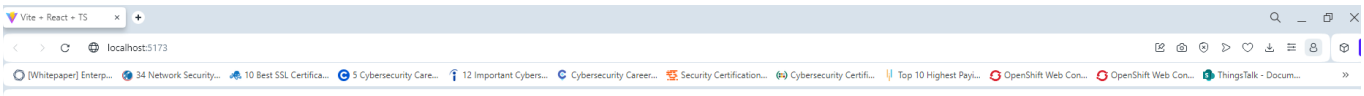
Just like the message component we should export the app component so that it can be used somewhere else.

We can see our terminal is stil running and we can see HMR which is short for
hot module replacement. So read under the hood monitors our files for changes whenever we make any changes it will automatically refresh our page in the browser. So we can see the display has changed.



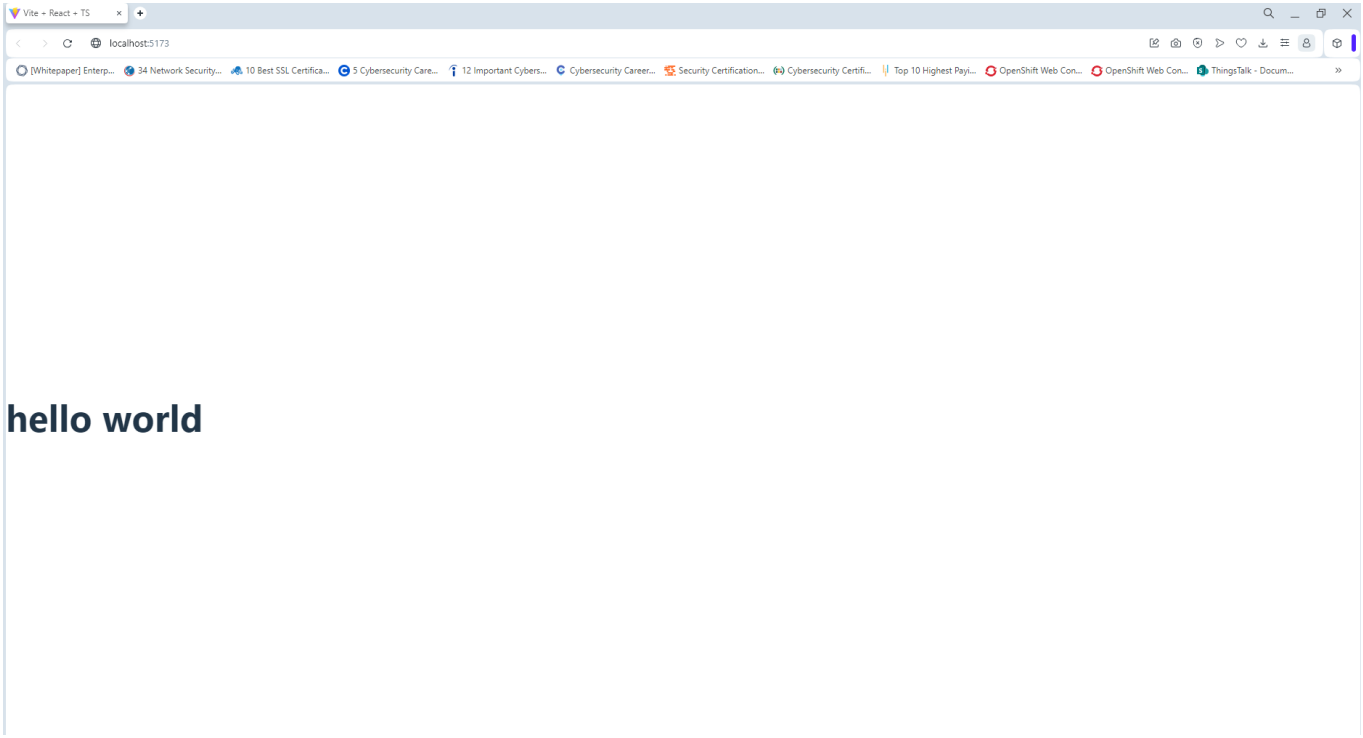
So with jsx we can easily describe user interface of our application with HTML and JavaScript now. The great thing with JSX is that it allows us to easily create Dynamic content. So for example here we can declare a constant code name, I set it to Dora. Now I can replace in hello world, world with name inside the braces. Inside the braces we can easily write any JavaScript expression.
Expression is a piece of code that produces a value so here I can reference a name constant. We can basically call any piece of code that returns a value.

We can also say if name is true return markup otherwise return something else like hello wolrd. If we go back to the browser we can see my name is rendered on the screen, however if I change the name to empty string we see hello world.



hello Dora

and again



So currently there is a component tree consisting of app being the root element or top level component and the message being a child when our applications starts React takes the component tree and build a JavaScripts data structure called the Virtual Dom. The Virtual Dom is different from the actual Dom in the browser. It'a a lightweight in-memory representation of component tree where each node represents a componnet and it's properties when the state or the data of a component changes React updates to the coressponding node in the virtual Dom to reflect the new state then it compares teh current version of Dom with previous version to identify the nodes that should be updated.
It will then update those nodes in the actual Dom. Technically updating Dom is not done by react itself, it's done by companion Library called React.

React is platform agnostic and it can be used to build apps for mobile, web and desktop devices.

So what is a difference between a library and a framework?

Library is a tool that provides specific functionality while framework is a set of tools and guidelines for building apps.

React it creates dybamiy and iteractive user interfaces but we hardly use only React to build modern applications. We often need additional tools for concerns like routing which allows user to go from one page to another making HTTP calls managing the application State internationalization form validations animation and so now great thing about React is that it doesn't have opinion about the additional tools we should use for this concerns.

Next step is installing Bootstrap. Very popular CSS library that gives us bunch of CSS classes for styling our application.

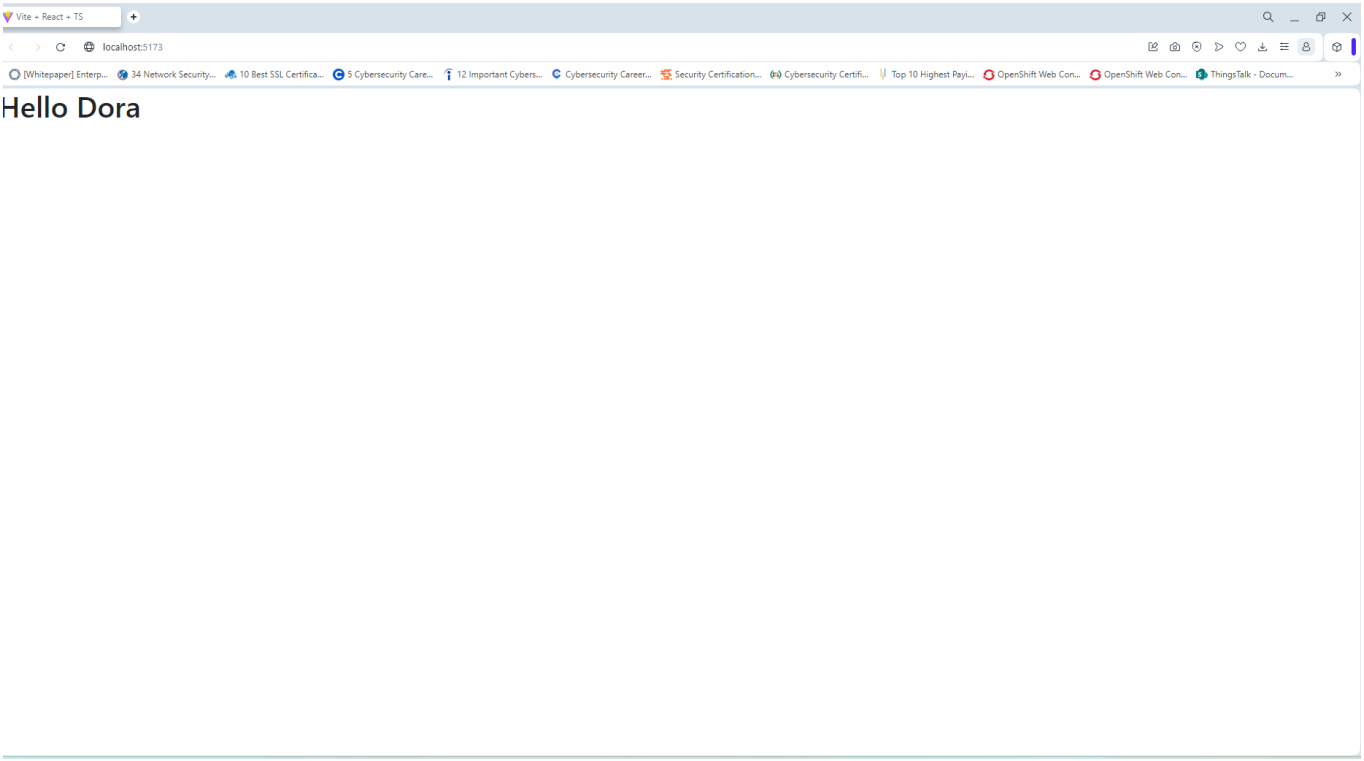
```
PS C:\Users\dklobucar\Documents\React js> npm i  bootstrap@5.2.3

up to date, audited 3 packages in 2s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

So next step is to import it inside CSS files. App.csss contains all the sytles for our app components, so I deleted what was inside.
Index.css conatins global styles for the application. I also deleted code inside index.cs.

I replaced index.cs in main.tsx with the import 'bootstrap/dist/css/bootstrap.css' as it can be seen in the photo:

```tsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import 'bootstrap/dist/css/bootstrap.css'

ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

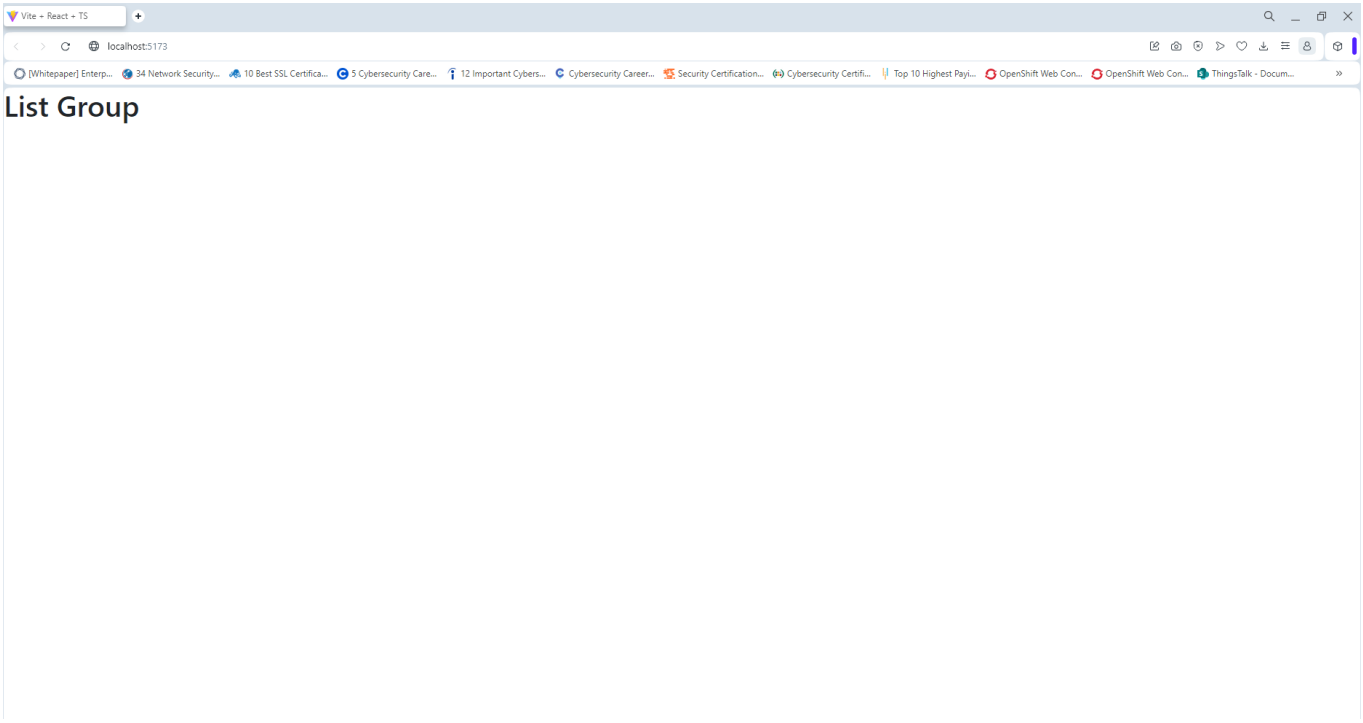
We can see that the title has changed and it has more beautiful fond.



So next step is to create a list group component. So I add a new folder called components. I added a new file called ListGroup.tsx. For now it just return an H1 element List Group and need to export this from the module. So now when we changed the ListGroup inside App.tsx like this:

```tsx
import Message from './Message'
import ListGroup from './components/ListGroup'


function App() {
return <div><ListGroup/></div>
}

export default App;
```

So display is now like this:



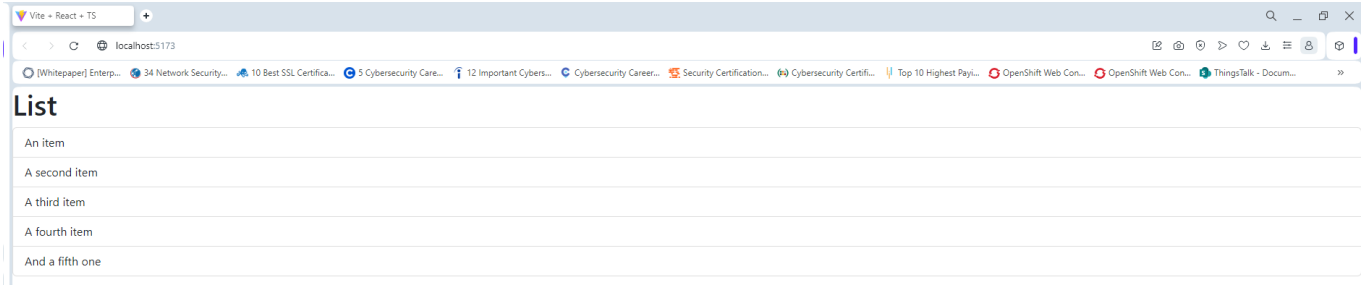
So I copied ListGroup from Bootstrap official website. Now we see this, but it doesn't allow us to render or display these items.

In React, a "fragment" is a feature that allows you to group multiple children elements without adding extra nodes to the DOM (Document Object Model). A fragment can be thought of as a lightweight wrapper that doesn't create an additional DOM element.

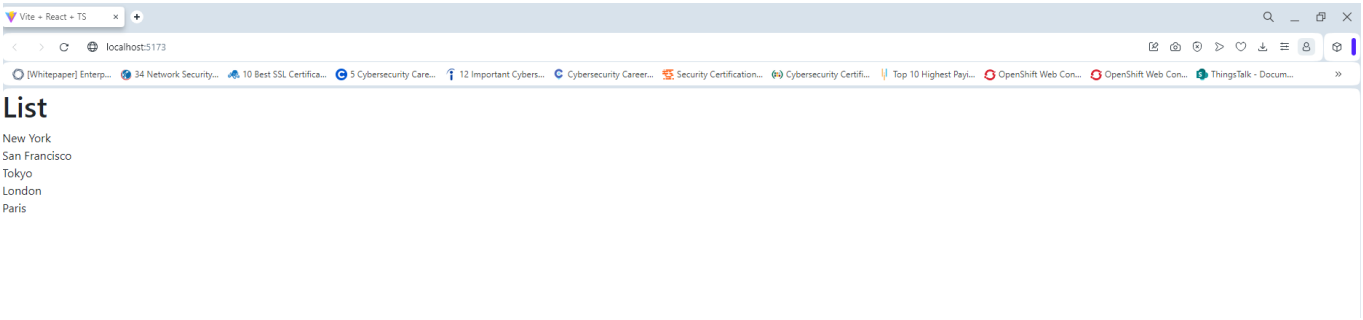
Wrap elements in `<Fragment>` to group them together in situations where you need a single element. Grouping elements in `Fragment` has no effect on the resulting DOM; it is the same as if the elements were not grouped. The empty JSX tag `<></>` is shorthand for `<Fragment></Fragment>` in most cases.

```jsx
import { Fragment } from "react";

function ListGroup() {
  return (
    <Fragment>
      <h1>List</h1>
      <ul className="list-group">
        <li className="list-group-item">An item</li>
        <li className="list-group-item">A second item</li>
        <li className="list-group-item">A third item</li>
        <li className="list-group-item">A fourth item</li>
        <li className="list-group-item">And a fifth one</li>
      </ul>
    </Fragment>
  );
}

export default ListGroup;
```

There is a shorter way instead importing fragmnet component from React. If we leave empty brackets we are telling react to use a fragment to wrap all this children. We can see the display like shown in the photo:



If we want to render a list of items dynamically then we can declare a constant named items adn set it to array of strings here. Map is used for mapping or converting ecah i, so here we want to convert each item to an li element so we type li and inside we want to render or display the item itself. Here we render the item itself, so we can now see display like this:



I can change constant to a variable so I can reassign this on the next line.

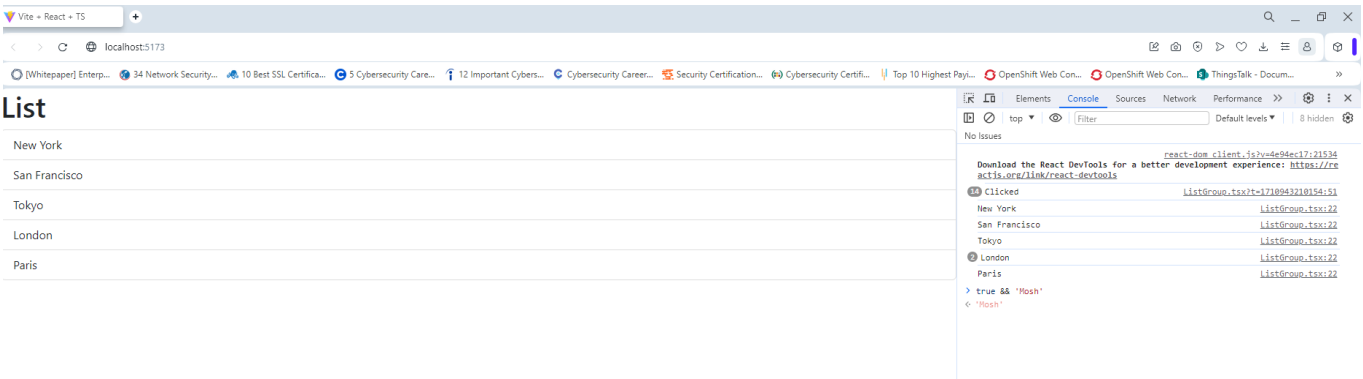
```
import { Fragment } from "react";

function ListGroup() {
  let items = ["New York", "San Francisco", "Tokyo", "London", "Paris"];
  items = [];
  if (items.length === 0) return <p>No item found</p>;
  return (
    <>
      <h1>List</h1>
      <ul className="list-group">
        {items.map((item) => (
          <li key={item}> {item}</li>
        ))}
      </ul>
    </>
  );
}

export default ListGroup;
```

So here are we mapping each item to a list item. So when creating the list item we have access to each utem beacuse we are using that item as the key of each list item right so instead of logging clicked we can simply log item.

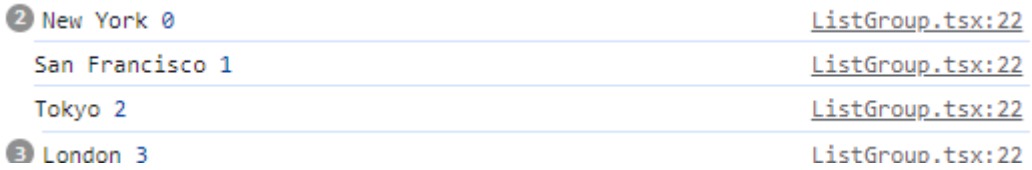
```
{items.map((item) => (

          <li

            className="list-group-item"
```

```
            key={item}
            onClick={() => console.log(item)}
          >
            {item}
          </li>
        ))}
      </ul>
    </>
```

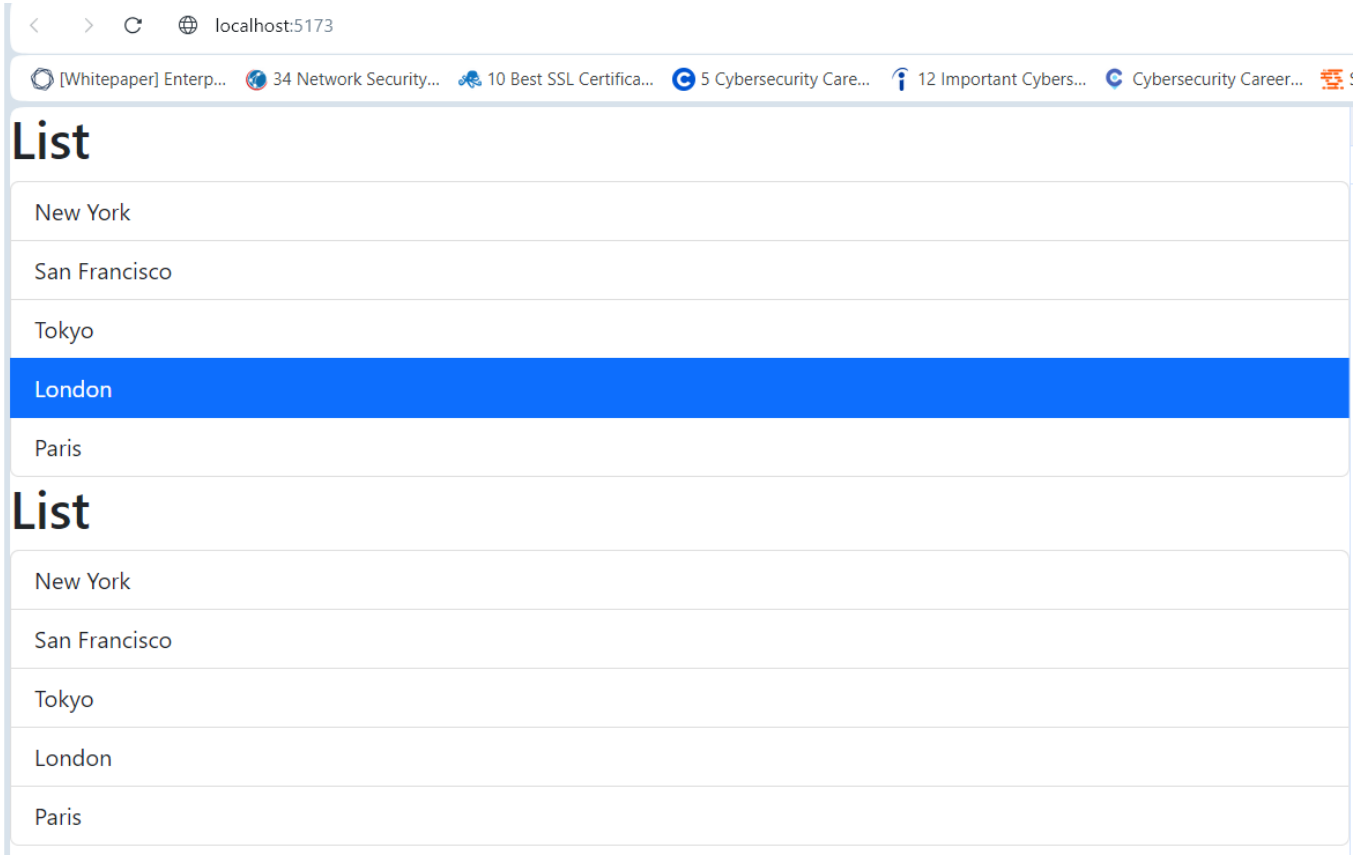
So now we can see this display:



We can also add index to see the index of an item like this:



I am gonna add another LIst inside App.tsx:

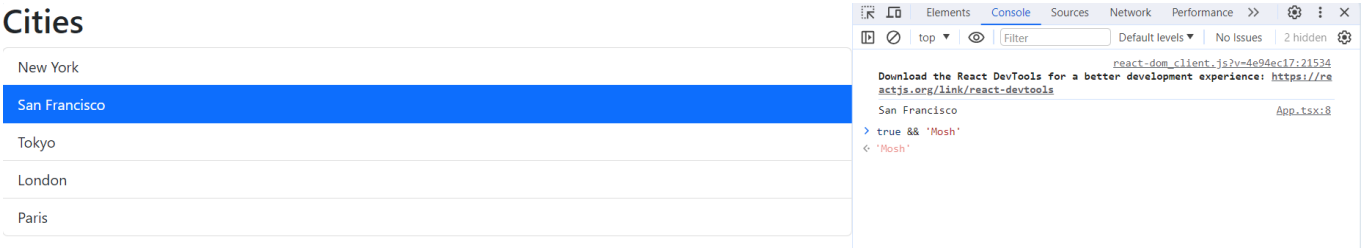


Passing Data via Props

In TypeScript, an interface is a way to define a contract in your code. It describes the shape of an object, specifying which properties and methods the object must have. Interfaces are particularly useful for enforcing a consistent structure across different parts of your codebase and for providing type-checking capabilities.

```tsx
import { useState } from "react";

interface Props {
  items: string[];
  heading: string;
}
function ListGroup(props: Props) {
  let items = ["New York", "San Francisco", "Tokyo", "London", "Paris"];
  const [selectedIndex, setSelectedIndex] = useState(-1);
```

Now when we select an item, it's displaying on the screen:



React Developers Tools

Creating a Bootstrap button component