React.JS

**Getting Started**

The first assignment I worked on was about adding the last button in the following image.



It was just an introductory app where I have to add the **React vs JS** button that will show the above-mentioned contents.

I added the following code to complete the assignment.

<button  
 className={activeContentIndex === 3 ? "active" : ""}  
 onClick={() => setActiveContentIndex(3)}  
>  
 React vs JS  
</button>

**React Project Setup**

To get started with React and writing code for that, we need a react project. One of the quickest ways to get started with react code, is to open a new chrome tab and type **react.new** it will open a code sandbox tab where everything that you might to build a react project is already present, and I just have to start writing react code only.

If you want to create a local environment to write react code then you can use **vite** or **create-react-app**. Also, you need node.js installed on your computer to make everything work.

To create a new react project using vite, you need to write

$ npm create vite@latest react-project

One special note here is you also have to install all the necessary packages that are used in the project e.g. react, react-scripts etc.

Then you need to write **npm run dev** in a terminal to run the development environment that allows you to visit and preview the website you are working on.

Why do we need a special project set up to write react code?

The reason is, react uses HTML code with Javascript which is also known as jsx. Jsx is not supported in normal .js file if we just use the conventional set up of writing an index.html and app.js and linking them together. Unfortunately, this will give us errors in the web browser. The reason is as mentioned for jsx not being supported in web browser. And that is why we need a special set up that can convert this jsx to simple js that the web browser understands.

NOTE: React projects use a build process to include the script tags in the html file automatically for you, so you will never have to add those files explicitly. It simply means the code that you write is the code that is executed (like this) in the browser. Your code is transformed behind the scenes before it is handed over to the browser. One reason for the build process is because the raw, unprocessed code won’t run on the browser and another reason is the code is not optimized for production (i.e. minified). Fortunately, react projects come with this build process without any custom tweaking or set up.

**Import and Exports**

In vanilla Javascript, we can always make variables, functions, classes available to other .js files by exporting them. Then, we can use import statements to import the specific variables, functions or classes from other files.

Let’s look at how we can do it.

index.html

<!DOCTYPE html>  
<**html**>  
 <**head**>  
 <**title**>JavaScript Refresher</**title**>  
 <**link** rel="stylesheet" href="assets/styles/main.css" />  
 <**meta** charset="UTF-8" />  
 <**script** src="assets/scripts/app.js" type="module"></**script**>  
 </**head**>  
  
 <**body**>  
 <**header**>  
 <**img** src="assets/images/js-logo-xs.png" alt="JavaScript logo" />  
 <**h1**>JavaScript Refresher</**h1**>  
 </**header**>  
  
 <**ul**>  
 <**li**>Base Syntax & Rules</**li**>  
 <**li**>Variables, Values & Operators</**li**>  
 <**li**>Functions</**li**>  
 <**li**>Objects</**li**>  
 <**li**>Arrays</**li**>  
 <**li**>Control Structures</**li**>  
 <**li**>Browser APIs & The DOM</**li**>  
 <**li**>Essential Features Used By React</**li**>  
 <**li**>Tricky Parts</**li**>  
 </**ul**>  
 </**body**>  
</**html**>

assets/scripts/util.js

export let apiKey = "12345";

assets/scripts/app.js

**import** { apiKey } **from** "./util.js";  
console.log(apiKey);

It is necessary to write **type=”module”** for any script where imports are used. Although, in react it is not needed for the same reason as **react-script** takes care of all modules by making a big file or set of big files that has all code included in it.

You just have to use **export** key to export the value and **import** syntax to import the particular value from the module. When mentioning the module name, you must specify its relative path. In vanilla JS, you must use .js but in react it is not mandatory.

Default export: Here you specify the default keyword after export to say that the value is the only value that will be exported from the file and there should no other default export. In default export, you should not use any let/const/var and the name of the variable.

assets/scripts/util.js

export default "12345";

assets/scripts/app.js

**import** apiKey **from** "./util.js";  
console.log(apiKey);

When using default to export the value, you can use any name to assign that value when importing in another file.

Alias: When importing variables or functions from one file to another, you can use alias to store all imported variables in a javascript object and access the values using this object.

assets/scripts/util.js

**export** **default** "abs";  
**export** **let** apiKey = "123";  
**export** **let** abc = 1;

assets/scripts/app.js

**import** \* **as** util **from** "./util.js";  
console.log(util.apiKey, util.abc, util.default);

The default export is saved in the default key. Other exports has their key name same as their variable name.