React JS

- ReactJS is a Javascript library originally developed by Facebook.
- It helps in building highly engaging single-page web apps.
- ReactJS helps in breaking down Complex UI into simpler components.

Installation

- Firstly you need to install VS code and NodeJS.
- After installation, Open Windows Power Shell (Shift + Right click and select WPS)
- Write Node - version to check if Node is installed successfully.

* We are going to learn ReactJS by creating a Todo List.

Setting up the Development Environm-
-ent.

- In Windows Power Shell type:

```
npx create-react-app todos-list
```

Name of Your App

Page No.
Date / /

Npx: It is an npm package that is expected to be run only once in a project. (i.e One time package)

⇒ By running the above code, a folder named Todos-List will be created.
⇒ Open the folder in VS Code.

Folder Structure

i) Readme.md: It is used to generate HTML summary, you see at the bottom of projects.

ii) .Gitignore: Files which you do not want to push in Github.

iii) $\overset{\nearrow \text{Folder}}{\text{Public}}$/$\overset{\nearrow \text{File}}{\text{index.html}}$: Main and only HTML file of our React app. This is the page that will be loaded on starting the application.

iv) src/index.js: JS file corresponding to index.html file.

v.) *src/App.js: Main component of any react app. It acts as a container for all other components.

vi) src/App.css: Help in injecting styling in the React app.

Lets Start the development server

=> Write npm start in VS code terminal

=> Open browser and go to http://localhost:3000

=> You have started Your react App. Now lets create the list(Todos).

* Index.js is the entry point for Components.

* Open App.js
You will see.

```
function App() {
    return ( ) ; }
            └> code is written here,
```

Page No.
Date / /

⇒ Classname: This name is used for defining class instead of 'Class' because of Conflict with "Class" keyword in many languages.

⇒ JSX: It allows us to write HTML in Javascript and place them in DOM without any appendChild() or createElement() method.

⇒ To write JS use Curly braces.
Eg. `<div>{12+4} </div>`
Output: 16

⇒ Importance of Wrap
```
return(
    <>
        <h3> My App </h3>
    </>
  }  ──> Wrap
```
* Without wrapping a return, it will throw an Error.

Date / /

Integrating Bootstrap into our React

=> From the Bootstrap starter template copy script src and paste in index.html at end of <body> tag.
=> Also copy-paste CSS <link> into your <head> of index.html

* You can copy the code of any B.S (Component) and paste it in your App.js
Note:=> Element with no closing tag needed a '/' at the End.
Tip: You can use Prettier extension.

Result: Your Component of Bootstrap will be inserted.

=> This is not an ideal way of writing code. As in App.js we create the structure of our website.
We don't write all the code there!!

* We will create the structure by dividing our website in Components.

DOMS Page No.
Date / /

Components in React
=) They are nothing but reusable Javascript functions.
=) Even if the component do not depend on each other, they merge inside a parent component to produce the final UI

Benefits :i) Allows reusability of code.
ii) Make it easier to find Error.

We are creating
i) Header Component
ii) Todo and Todoitem components
iii) Footer component

Creating Header Components
i) Inside Src, Create a folder named
Any Name → My Components.
ii) Create a new file Header.js

File which will contain all code for the Header.

=) We will create a functional-based component in Header.js

Tip: Download/Install the extension
ES7 React/Redux/GraphQL/R-N Snippets
↳ It will make the development easy.

Eg. Write rf and Enter.
⇒ You will get React func-base component
Syntax:

```
import React from 'react'
                    ↱ Remember while importing
Export default function header(Props){
    return(
        //statements that we want to
                  return )
    }
```

We are creating navbar using Bootstrap

Props: They are nothing but JS objects that are passed from parent component to Child component.
For Eg. Using {props.title}
↳ Created in App.js (Later)
⇒ This means that we will be passing the title object from App.js to Header.js

Date / /

In App.js
As mentioned, Every React component is merged in main component i.e, App.js

* So, we need to import Header.

=> To import a default function, type:
Syntax:
```
import Header from './MyComponent/Header';
```
Name ↲ ↳ Location

=> To return the Header Component type:
```
function App() {
return (
    <>
    <Header title = "Todos List"/>
    <>
    ); }
```
↳ component

We have passed title = "ToDos list" in our Header Component.

Check out:-
Q. Similarly, You can create a footer compontent and can pass something in it. Do it by Yourself?

Default Props
It will set default values for the prop attributes if the parent component does not send the values
Eg.
```
Header.defaultprops = {
    title: "Your Title" }
```
↳ If the above passed statement wasn't there or if there was an issue while importing, then the default value would show up.

Creating Todo and Todoitem Components
i) Todo Component: This component will be responsible for keeping track of all items included inside the todos list.

ii) Todo Item: This component will be responsible for keeping track of the individual todo item.

DOMS Page No.
Date / /

⇒ We will create a Todo list in the parent component, and then we will pass the list to Todos Component

In App.js

```
Import {Todos} from './Mycomp/Todos'
Import {TodoItem} from './My Co/
                          Todoitem'
```
Created later

```
function App() {
    Let Todos = [
    { Sno: 1
      Title: "Go to market",
      Desc: "Buy vegetables"}

    {Sno: 2 ---}
    {Sno: 3 ---} ]
```

* We have created a JS object named Todos containing the list of items in the Todos list

```
  return (
      <>
```

<Header title = "ToDos List"/>
↳ The Header (we have created earlier

<Todos todos = {Todos} />
↳ Component (Create it)

</>); }

Here, we have passed todos object to Todos Component Now, we need to create our Todo Component.

* Create new file Todos.js in My Components folder.

In Todos.js

```
import React from 'react'
import {Todo item} from "./Todoitem";

Export Const Todos = (props) => {
    return( <div Classname = "Container">
CSS ← [ <h3 Classname = "Text-center> Todos
in 'Todos List' Text                list: </h3>

    {props.todos.map ((todo) => {
        return <Todoitem todo={todo}/>
    })}              ↳ Created component
        </div> )}
```

DOMS Page No.
Date / /

* Props.todos.map
This calls the call back function one time for each element in the array

=> Let's create our third component that is Todoitem
=> Create Todoitem.js in My component.

In Todoitem.js

```
Import React from 'react'
     ↗ Named Export
Export Const TodoItem =
      ({Todo, Ondelete}) => {
                 ↳ Discussed Later

 return ( <div>
     <h4> {Todo.Title} </h4>
     <p> {Todo.Desc} </p>
      </div>
     )}
```

* Open the server, and all items of our todolist are displayed successfully

Date / /

* Now, we will create delete button to delete items from Todo list.

Creating Delete Button
To create the delete button, we will use the state in React.

In App.js
Firstly import state hooks, by typing -
import React, {useState} from 'react';

useState: It is a hook that lets you add react state to function component

Type the following:
~~const [Todos, set Todos] = useState([~~

```
function App() {
const ondelete = (todo) => {
setTodos ( Todos. filter((e) => {
      return e! ==todo; }));
    }
  Const [Todos, setTodos] = useState([

      { Sno. - - -  // Earlier Coded}
```

Date / /

```
return ( <>
    <Header Title ='ToDos List"/>
    <Todos todos = {todos} On delete =
                        {On Delete} />
  </>) ; }
```

* In the above code, we have created an on delete function which will be called, once the user clicks on the delete button.
* Inside the return function, we have passed the on delete attribute to the Todos Component.

Open the Todos.js

```
Export Const Todos = (props) => {
return(//Earlier code)
      {props. todos. length = = =0 ?" No
Todos to display";
 Props. todos. map ((todo) => {
return <Todo Item  tode = {Todo}
key = { todo. Sno} On delete = {Props.
on delete} />
  })
```

⇒ In the above code, we're checking the length of the todos, so if the user deletes all todo item(length=0) then "No Todos to display" will be shown.

In Todoitem.js

```
Export const Todoitem=({Todo, Ondelete})
  => { return (
 <div> <h4> {Todo.title} </h4>
       <p> {Todo. desc} </p>
<button classname="btn btn-sm btn
-danger" On Click={() => {On delete(todo)
                                   }}>
       Delete </button>
        </div>
       )}
```

In the above code, we have created a Delete Button, which will call the ondelete function once the user clicks on it and the particular todo item will be removed.

Work for You.
⇒ In a similiar fashion, You can create AddTodo.js
You can simply do it by creating a function such that on using it new item, from a form, gets added in our list. (without reloading the page).

React Router
It enables navigation among views of various components in React app, allow changing the browser URL and keep the UI sync with URL.

Install
In terminal write,

```
npm install react-router-dom
```

* Visit documentation of React router, an import react-router-dom by simply copying it.

Date / /

=) Wrap our app using React router
<router> </router>.

=) To specify some components for rendering we will use switch.
Eg.

```
<Switch>
<Route exact path="/" render={() => {
return( <>
<Add Todo addTodo={addtodo}/>
<Todos todos={todos} Ondelete={Ondelete}/>
</>)
}}>
```

=) These two will be render if path is "/"

```
</Route>
<Route exactpath="/about">
```

→ will be render if path is /about.

```
<About/>
</Route>
</Switch>
```

⇓
=) You can create about.js easily

=) You can now navigate among different pages, without reloading the App.

All the Best