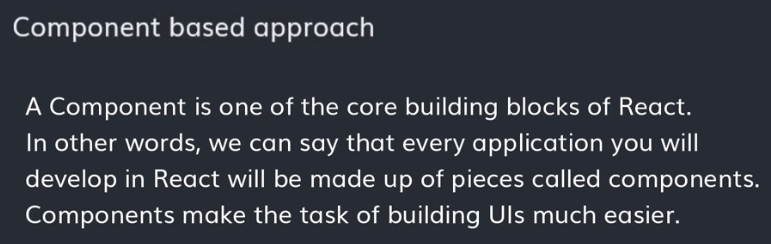
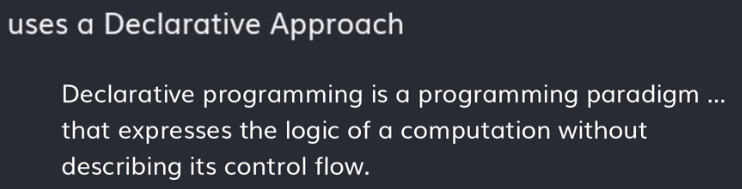
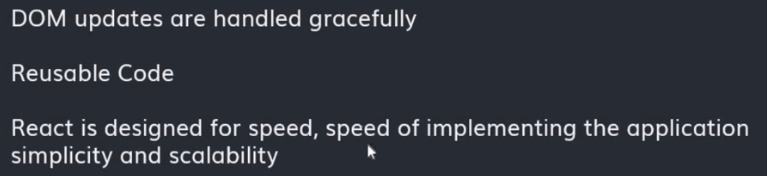
**REACT JS**

React is a JavaScript library (not framework) that forces you to think in terms of components and use for building awesome user interfaces.

**ABOUT REACT**







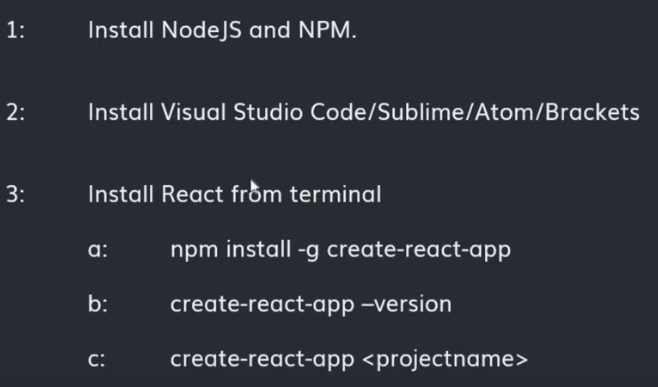
**PREREQUISITES FOR REACTJS**

1: Basic knowledge of HTML, CSS, and JavaScript.

2: Basic understanding of ES6 features.

3: Basic understanding of how to use npm (node package manager).

**REACTJS ENVIRONMENT SETUP**



**Install NodeJS and NPM (nodejs.org):**  
Download and install NodeJS. After installing check whether it will properly installed or not:

Command Prompt: **C:\Users\ap\_aa>node –v** (it will give your version installed onto your system)

NodeJS automatically install NPM and to check whether it will install or not:

Command Prompt: **C:\Users\ap\_aa>npm –v** (it will give your version installed onto your system)  
  
**Install Visual Studio Code:**  
  
Download and install Visual Studio

**To Create React App**  
  
Command Prompt: **C:\Users\ap\_aa>npm install -g create-react-app** **//g represent global**

Visual Studio Code: **D:\MERN Project\frontend> npx create-react-app .**



If successful, you should be able to get version using:  
  
Command Prompt: **C:\Users\ap\_aa>create-react-app –version**

**Create react project:**  
  
Change your drive in command prompt so that you can create your project in which drive you want:

D:\>mkdir reactlearning **//create a new folder under D drive**

D:\>cd reactlearning

D:\reactlearning>create-react-app reactlearn **//reactlearn is your project name (it will create an app for you)**

**NOTE: package.json is used to store the metadata associated with the project as well as to store the list of dependency packages.**

**To Run Project;**

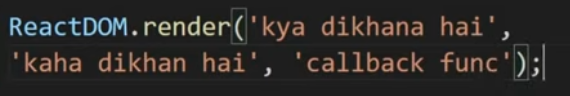
D:\reactlearning>cd reactlearn

D:\reactlearning\reactlearn>npm start **//to launch app or start app**

**localhost:3000**

**ReactDOM.render{ //to display content(src/index.js)**

**}**



# REACT FOLDER STRUCTURE

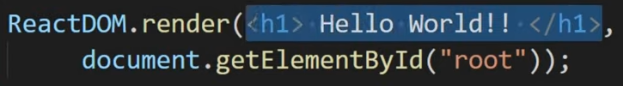
**package.json** is used to store the metadata associated with the project as well as to store the list of dependency packages

**index.js under src folder (src/index.js):** react code to be written in this file

**index.css under src folder (src/index.css):** this file is used to write CSS code

# JSX IN REACT JS

# JSX stands for JavaScript XML. JSX allows us to write HTML in React. JSX makes it easier to write and add HTML to React. JSX is an extension of the JavaScript language based on ES6 and is translated into regular JavaScript at runtime.

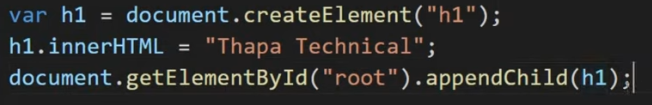


Highlighted part in above image is **JSX Expression in React** and React (import React from ‘react’;) must be in scope when using JSX.

|  |  |
| --- | --- |
|  |  |

**COMPILING of REACTDOM (using babel)**

**Another Method**



# HOW TO RENDER MULTIPLE ELEMENTS INSIDE REACTDOM.RENDER() IN REACTJS

# In React, we can only render one element. Even if we have multiple elements to render, there can only be a single root element. This means if we want to render two or more elements, we have to wrap them in another element or component. Commonly, the element used for this is a div tag.

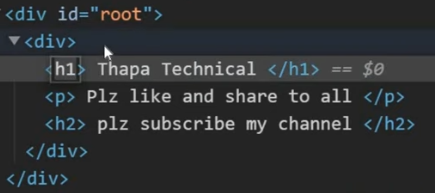
# 

# OR

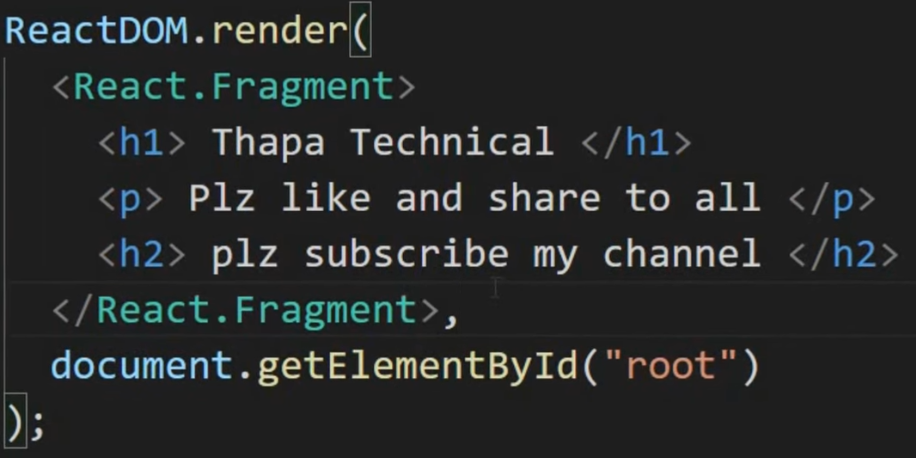
# 

# UNDERSTANDING REACT FRAGMENT IN REACT JS

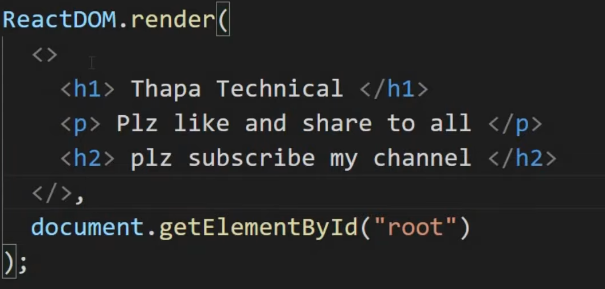
A common pattern in React is for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM.



We can see div tag inside another div tag. So to remove this problem we use **React.Fragment**

Use **React.Fragment** instead of div tag for multiple elements. (time consuming, take more memory)

**OR**



We can even replace React.Fragment with just opening and closing tags.

# JAVASCRIPT EXPRESSIONS IN JSX

# JSX is an Expression Too After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

# You can put any valid JavaScript expression inside the curly braces in JSX. With JSX you can write expressions inside curly braces { }. The expression can be a React variable, or property, or any other valid JavaScript expression. Inside {}, we cannot use statements like if else etc.

# 

# If we want to use JavaScript inside HTML (this HTML is inside JavaScript), then use curly braces {}:

# 

# Example : <p> Random Number is : {Math.random()} </p>

# ES6 TEMPLATE LITERALS IN JSX IN REACTJS

# Template literals are string literals allowing embedded expressions inside JSX in React. You can use multi-line strings and string interpolation features with them.

# 

# ANOTHER METHOD:

# <h1> My name is {fname + “ “ + lname} </h1>

# ANOTHER METHOD (USING TEMPLATE LITERALS with back tick) :

# 

# NOTE: TO FETCH CURRENT DATE AND TIME(new Date(Year, Moth, Day, Hours, Minutes, Seconds))

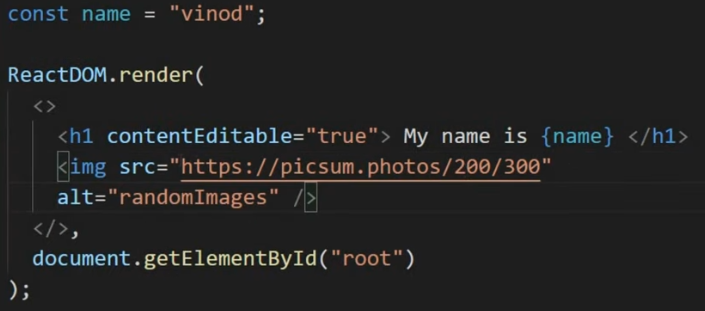
const currentDate = new Date().toLocaleDateString();

const currentTime = new Date().toLocaleTimeString();

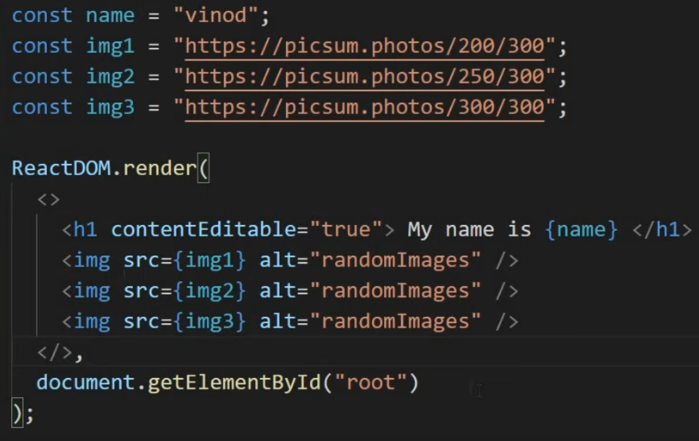
# JSX ATTRIBUTES IN REACTJS | HTML ATTRIBUTE VS JSX ATTRIBUTE IN REACT

JSX Attribute is in camel case. Example: **contentEditable**

**Self Closing Tag:** Those tags who do not have any children like <img>. So we write as **<img />**



**OR**



**Example: Create Linked Image**

const links = “<https://www.google.com>”;

const img = “<https://picsum.photos/200/300>”;

ReactDOM.render(

<>

<a href = {links} target = “\_blank”>

<img src = {img} alt = “randomImages”/>

</a>

</>,

document.getElementById(“root”)

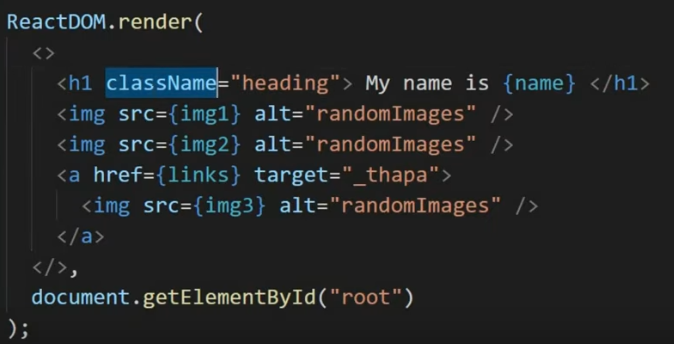
);

# CSS STYLING & IMPORTING CSS FILES IN REACT JS | CLASS VS CLASSNAME IN REACT JS

# To use CSS External file in index.js (src/index.js), we need to import using:

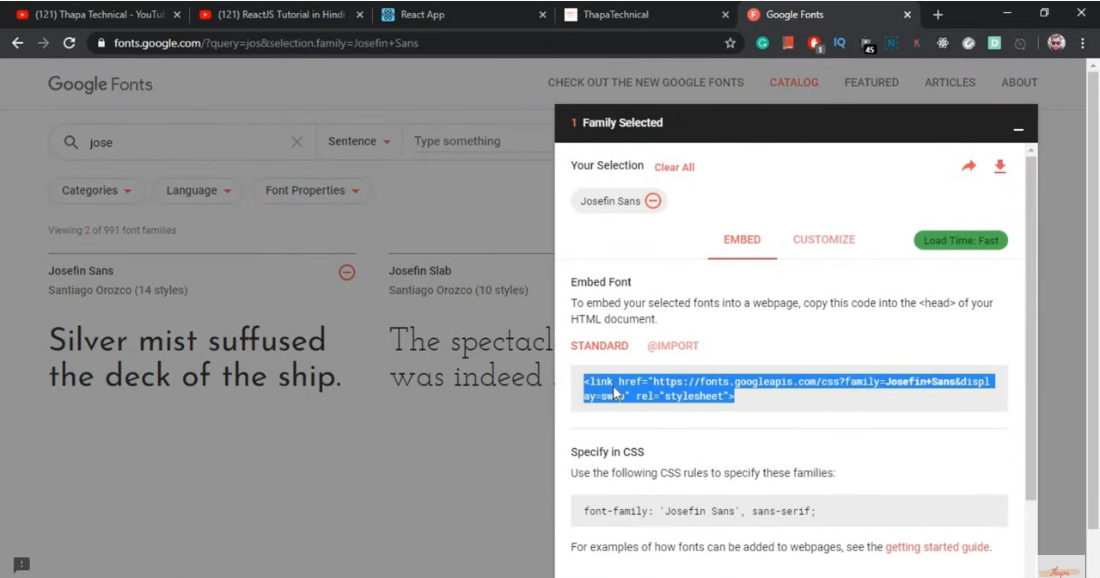
**import './index.css';**

In HTML, we use CLASS to style our elements but in React, we use **className**

.

# HOW TO USE GOOGLE FONTS IN REACT JS APPLICATION

Visit to fonts.google.com



# Now go to index.html under public folder in your VS Code (public/index.html)

# Under <head> tag, paste the link of google font:

# <link href="https://fonts.googleapis.com/css2?family=Supermercado+One&display=swap" rel="stylesheet">

# Now copy the following CSS rules to specify these families in above image and paste in your index.css file under src folder (in the class of which you want to change the font):

# font-family: 'Supermercado One', cursive;

**NOTE: CSS FOR IMAGE GALLERY**

.div\_image{

  display: flex;

  justify-content: center;

}

.div\_image img{

  width: 200px;

  height: 300px;

  padding-right: 10px;

}

# INTERNAL CSS & INLINE CSS STYLING IN REACT JS

# In CSS, we use kebab-case as text-tranform but in React, we use camelCase as textTransform

# In React, Inline or Internal CSS is used as an object. Example of an object is as:

const heading = {

key: ‘Value’,

key: ‘Value’

}

**Internal CSS in React (creating object)**

const heading = {

color: ‘#fa9191’,

textTransform: ‘capitalize’,

textAlign: ‘center’,

fontWeight: ‘bold’,

textShadow: ‘0px 2px 4px #ffe9c5’,

margin: ‘70px 0’,

fontFamily: '”Supermercado One”, cursive’

}

**Implementing Internal CSS in ReactDom.render function:**

<h1 style= {heading} >

  Image Gallery

</h1>

**Inline CSS in React**



**CHALLENGE:**

**index.css**

body {

  margin: 0;

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

    sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

  background-color: #b4f2e1;

  box-sizing: border-box;

  padding: 0%;

}

code {

  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

    monospace;

}

div{

  width: 100%;

  height: 100vh;

  display: flex;

  justify-content: center;

  align-items: center;

  margin-top: -30px;

}

span{

  color: #eb6383;

}

h1{

  padding: 20px 20px;

  background-color: #ffe9c5;

  color: #fa9191;

  border-radius: 20px;

}

**index.js**

const currDate = new Date().getHours();

const cssStyle = {};

let greet = '';

if(currDate > 1 && currDate < 12){

  greet = 'Good Morning';

  cssStyle.color = 'Green';

}else if (currDate >= 12 && currDate < 20){

  greet = 'Good Afternoon';

  cssStyle.color = 'Orange';

}else{

  greet = 'Good Night';

  cssStyle.color = 'Red';

}

ReactDOM.render(

<>

<div><h1>

  Hello, <span style= {cssStyle} > {greet} </span>

</h1></div>

</>,

document.getElementById("root")

);



# REACT FUNCTIONAL COMPONENT

# In React components are the building blocks of any React app. Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and returns HTML via a render function. Components come in two types, Class components, and Function components

# Example:

# 

# NOTE: We can create a component for every jsx elements. If we want to create a component in React, then add a new file under src as filename.jsx (first letter of file must be in capital letter) and then create a function under that file which contain jsx elements as shown below.

# Example: if we want to create a component for heading tag named as Heading.jsx, paragraph tag as Para.jsx and ol tag as List.jsx

# 

# 

# 

# 

# NOTE: How we can import our different component in our main index.js file:

# 

# NOTE: App.js (file in which we store all our components) and this file is import in index.js file

# 

# 

# index.js

# 

# ES6 MODULES IMPORT EXPORT IN REACT JS

# ES6 provides two ways to export a module from a file: named export and default export.

# With named exports, one can have multiple named exports per file. Then import the specific exports they want to be surrounded in braces. The name of the imported module has to be the same as the name of the exported module.

# With default export, one can have only one default export per file.

# Example:

# App.js

const youtuber =  'Aayush Agarwal';

const favProg = 'React JS';

function myName() {

  let name = 'Priyanka Agarwal';

  return name;

}

function myNames() {

  let names = 'Neelam Agarwal';

  return names;

}

export default **youtuber**; **//only one default export**

export **{favProg, myName, myNames}**; **//multiple named export**

# index.js

import React from 'react';

import ReactDOM from 'react-dom';

**import youtuber, {favProg, myName, myNames} from './App';**

ReactDOM.render(

<>

  <ol>

    <li> Aayush </li>

    <li>  **{youtuber}**  </li>

    <li>  **{favProg}**  </li>

    <li>  **{myName()}**  </li>

    <li>  **{myNames()}**  </li>

  </ol>

</>,

document.getElementById("root")

);

# PROPS IN REACT JS

# Components need to communicate (send data to each other) and the way to pass data between components is by using props. “Props” is a special keyword in React, which stands for properties and is being used for passing data from one component to another.

# Example: Use of Cards in React

# 

# CHALLENGE:

# 

# SOLUTION:

# We created a custom component named as Card.jsx

# 

# index.js file in which we import Card component and also create custom properties or attributes of a component (name of properties can be anything)

# C:\Users\ap_aa\OneDrive\Desktop\Untitled.png

# Now, to simplify index.js more, we can create another custom component named as Sdata.jsx:

# 

# 

# Now, index.js (changes in this file)

# 

# 

# index.css file

# 

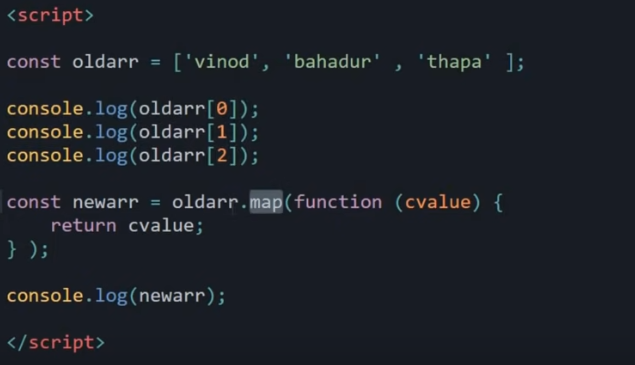
# 

**ARRAY MAP METHOD**

The map() method creates a new array with the results of calling a function for every array element. The map() method calls the provided function once for each element in an array, in order. Note: map() does not execute the function for array elements without values. Note: this method does not change the original array.



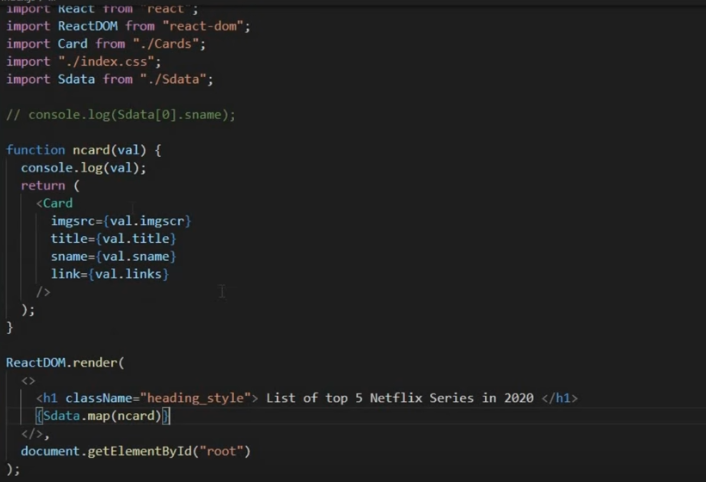
**Example**: How to use map() function instead of using loop for accessing values of an array.



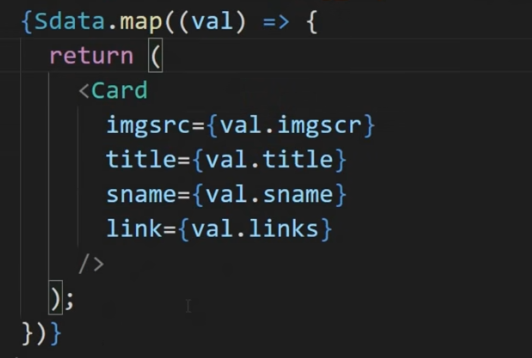
**Example**: Usage of map() function for array of objects.



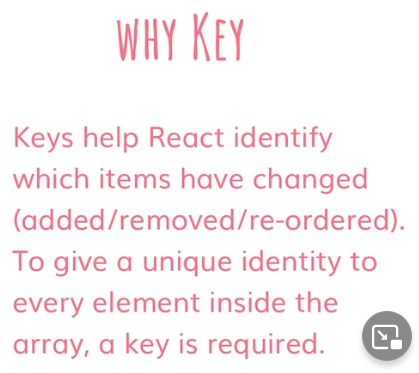
**Example**: Previous challenge index.js file (with map())

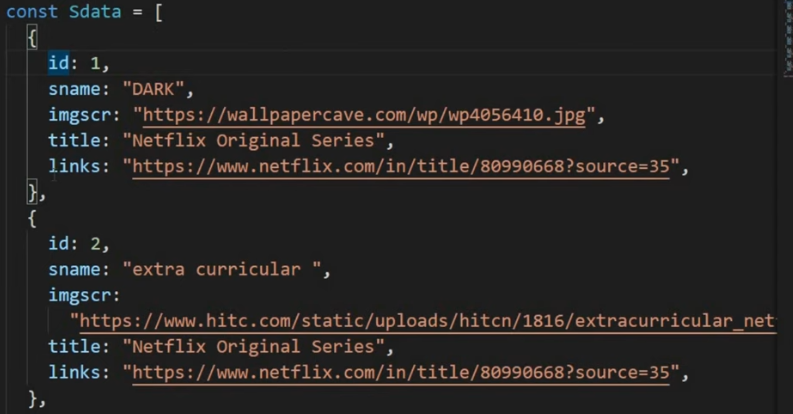


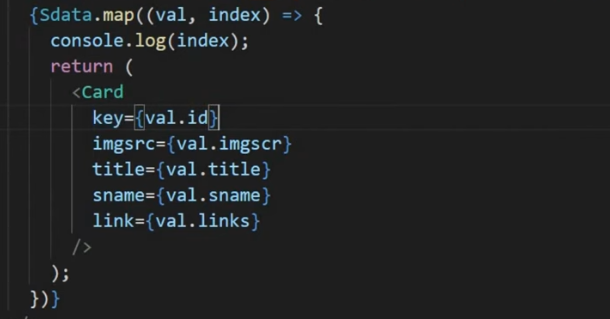
We can also use fat arrow function inside map function as:



Now we must have **unique ‘key’ prop** for every child in a list. So we should add one more key value pair for every elements which provide better understanding when we have to access specific elements details.







# REACT DEVELOPER TOOLS | DEBUGGING & ERROR HANDLING IN REACT JS

# React is one of the fastest-growing front-end frameworks. It makes creating complex and interactive UIs painless. Like other frameworks, it has a debugging toolset called React development tools. React development tools (React DevTools) is a browser extension available for Chrome, Firefox, and as a standalone app that allows you to inspect the React component hierarchy in the Chrome developer tools. Add extension into your browser: React Developer Tools by React

# Right click on any page -> inspect elements -> go to component

# 

# IF ELSE STATEMENT | CONDITIONAL RENDERING IN REACT JS

# In a conditional render, a React component decides based on one or several conditions which DOM elements it will return. Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.

# Example: If we want to show cards based on title of series:

# C:\Users\ap_aa\OneDrive\Desktop\Untitled.png

# We can build separate component based on title like for Netflix, we have Netflix.jsx:

# 

# Similarly, for Amazon, we have Amazon.jsx

# Then, how we can import both component in our main index.js file

# 

# REACT CONDITIONAL RENDERING | TERNARY OPERATOR IN REACT JS

# The conditional (ternary) operator is the only JavaScript operator that takes three operands: a condition followed by a question mark ( ? ), then an expression to execute if the condition is truth followed by a colon ( : ), and finally the expression to execute if the condition is false.

# Ternary Operator Syntax: condition ? exprIfTrue : exprIfFalse

# 

# HOW TO TYPE EMOJI 👍 IN VS CODE

Step by Step on How to type emoji on VS Code

1: Open VS Code IDE

2: Click on the Extension Tab

3: search for **:emojisense**

4: Install it

**Now, you have to edit the .json file to make it work perfectly**

5: Go to **setting**

6: click on the **extension** tab

7: click on **:emojisense**

8: now on the right side of the screen, click on Edit in **setting.json** file

"emojisense.languages": {

    "plaintext": {

      "markupCompletionsEnabled": true,

      "emojiDecoratorsEnabled": true

    },

    "javascript.validate.enable": false,

    "abap": true,

    "bat": true,

    "bibtex": true,

    "clojure": true,

    "coffeescript": true,

    "c": true,

    "cpp": true,

    "csharp": true,

    "css": true,

    "diff": true,

    "dockerfile": true,

    "fsharp": true,

    "git-commit": true,

    "git-rebase": true,

    "go": true,

    "groovy": true,

    "handlebars": true,

    "html": true,

    "ini": true,

    "java": true,

    "javascript": true,

    "javascriptreact": true,

    "json": true,

    "jsonc": true,

    "latex": true,

    "less": true,

    "lua": true,

    "makefile": true,

    "markdown": true,

    "objective-c": true,

    "objective-cpp": true,

    "perl6": true,

    "php": true,

    "powershell": true,

    "jade": true,

    "python": true,

    "r": true,

    "razor": true,

    "ruby": true,

    "rust": true,

    "scss": true,

    "sass": true,

    "shaderlab": true,

    "shellscript": true,

    "sql": true,

    "swift": true,

    "typescript": true,

    "typescriptreact": true,

    "tex": true,

    "vb": true,

    "xml": true,

    "xsl": true,

    "yaml": true

  }

# HOOKS IN REACT JS | USESTATE IN HOOK IN REACT JS

# Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes. React provides a few built-in Hooks like useState. The useState hook allows us to make our function components stateful. When called, useState returns an array of two items. The first being our state value and the second being a function for setting or updating that value. The useState hook takes a single argument, the initial value for the associated piece of state, which can be of any Javascript data type.

# Example: To increment number while clicking on a button.

# When you use useState(), React automatically import useState component

# const [count, setCount] = useState(); //array destructuring

# Here useState provides initial value, count provide current data and setCount (setCount is a function) provides update data.

# 

# Example: Update time on Button Click

import React, { useState } from 'react';

const App = () => {

let newTime = new Date().toLocaleTimeString();

const [ctime, setCtime] = useState(newTime);

const UpdateTime = () => {

  newTime = new Date().toLocaleTimeString();

  setCtime(newTime);

}

  return(

    <>

      <h1>{newTime}</h1>

      <button onClick={UpdateTime}>

Click Me

      </button>

    </>

  )

}

# Example: Update time automatically

import React, { useState } from 'react';

const App = () => {

let newTime = new Date().toLocaleTimeString();

const [ctime, setCtime] = useState(newTime);

const UpdateTime = () => {

  newTime = new Date().toLocaleTimeString();

  setCtime(newTime);

}

setInterval(UpdateTime, 1000);

  return(

    <>

      <h1>{newTime}</h1>

    </>

  )

}

# HANDLING EVENTS IN REACTJS

# React can perform actions based on user events. React has the same events as HTML: click, change, mouseover etc. Handling events with react have some syntactic differences from handling events on DOM. These are: React events are written in camelCase syntax: onClick instead of onclick. React event handlers are written inside curly braces: onClick={shoot} instead of onClick="shoot()".

# Example: To change background and button text on click event

# 

# REACT CONTROLLED VS UNCONTROLLED COMPONENT

# In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself. To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM.

# In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with [setState()](https://reactjs.org/docs/react-component.html#setstate).

# Example: onClick, data on input field should display at heading position

# 

import React, { useState } from 'react';

const App = () => {

  const [name, setName] = useState("");

  const [fullName, setFullName] = useState();

  const inputEvent = (event) => {

    setName(event.target.value);

  }

 const onSubmit = () => {

    setFullName(name);

  }

  return(

    <>

      <h1>Hello {fullName}</h1>

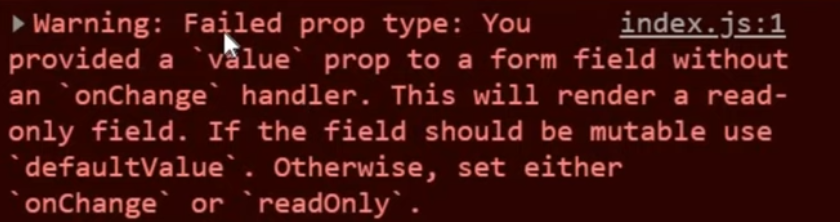
      <input type="text" placeholder="Enter Your Name" **onChange**={inputEvent} value={name}/><br/><br/>

      <button onClick={onSubmit}>Click Me 😃</button>

    </>

  )

}



**onChange** handler must be given if you want to change the field value otherwise it will consider as read-only. This onChange function will pass an object which can be accessed in:

const inputEvent = (**event**) => { **//any name can be given**

    setName(**event.target.value**); **//access value of an object**

  }

|  |  |
| --- | --- |
| **Without <form> tag** | **With <form> tag** |
| <>        <h1>Hello {fullName}</h1>        <input type="text" placeholder="Enter Your Name" **onChange**={inputEvent} value={name}/><br/><br/>        <button onClick={onSubmit}>Click Me 😃</button>      </> | <>  **<form onSubmit={** **onSubmit }>**        <h1>Hello {fullName}</h1>        <input type="text" placeholder="Enter Your Name" **onChange**={inputEvent} value={name}/><br/><br/>        <button **type=”Submit”**>Click Me 😃</button>**</form>**      </> |

**<form> tag send your data to the backend for storage and your page returns to its original state.** But if we want to display any value to the page with this <form> tag, then these changes should be made:

**<form onSubmit={** **onSubmit }>**

**const onSubmit = (event) => { //object should pass on onSubmit function**

**event.preventDefault();**

**setFullName(name);**

**}**

**Example**: to display value of two input fields onto our page

import React, { useState } from 'react';

const App = () => {

  const [name, setName] = useState("");

  const [pass, setPass] = useState("");

  const [fullName, setFullName] = useState();

  const [fullPass, setFullPass] = useState();

  const inputEvent = (event) => {

    setName(event.target.value);

  }

  const inputPass = (event) => {

    setPass(event.target.value);

  }

 const onSubmit = (event) => {

   event.preventDefault();

    setFullName(name);

    setFullPass(pass);

  }

  return(

    <><form onSubmit={onSubmit}>

      <h1>Hello {fullName} {fullPass}</h1>

      <input type="text" placeholder="Enter Your Name" onChange={inputEvent} value={name}/><br/><br/>

      <input type="text" placeholder="Enter Your Password" onChange={inputPass} value={pass}/><br/><br/>

      <button type='submit'>Click Me 😃</button>

      </form>

    </>

  )

}

# HANDLING COMPLEX MULTIPLE INPUT FORM STATES IN REACT JS

# Here we use name attribute and useStateunder every field of form tag.

import React, { useState } from 'react';

const App = () => {

  const [fullInfo, setFullName] = useState({ **//useState will pass an object also**

    fName : "",

    lName : "",

    emailId : "",

    mobileNo : ""

  });

  const onSubmit = (event) => {

    event.preventDefault();

   }

  const inputEvent = (event) => {

    const {value, name} = event.target;

    setFullName((preValue) => {

      if(name === 'fName'){

        return {

          fName: value,

         lName: preValue.lName,

         emailId: preValue.emailId,

         mobileNo: preValue.mobileNo

      };

      }else if (name === 'lName'){

        return {

          fName: preValue.fName,

          lName: value,

          emailId: preValue.emailId,

         mobileNo: preValue.mobileNo

        };

      }else if (name === 'emailId'){

        return {

          fName: preValue.fName,

          lName: preValue.lName,

          emailId: value,

          mobileNo: preValue.mobileNo

        };

      }else if (name === 'mobileNo'){

        return {

          fName: preValue.fName,

          lName: preValue.lName,

          emailId: preValue.emailId,

          mobileNo: value

        };

      }

    });

  }

  return(

    <><form onSubmit={onSubmit}>

      <h1>Hello {fullInfo.fName} {fullInfo.lName}</h1>

      <p>Email Id {fullInfo.emailId}</p>

      <p>Mobile Number {fullInfo.mobileNo}</p>

      <input type="text" placeholder="Enter Your First Name" onChange={inputEvent} value={fullInfo.fName} name='fName'/><br/><br/>

      <input type="text" placeholder="Enter Your Last Name" onChange={inputEvent} value={fullInfo.lName} name='lName'/><br/><br/>

      <input type="email" placeholder="Enter Your Email Id" onChange={inputEvent} value={fullInfo.emailId} name='emailId'/><br/><br/>

      <input type="number" placeholder="Enter Your Mobile Number" onChange={inputEvent} value={fullInfo.mobileNo} name='mobileNo'/><br/><br/> **//name attribute**

      <button type='submit'>Click Me 😃</button>

      </form>

    </>

  )

}

export default App;

# SPREAD OPERATOR (…)

# The spread operator ... is used to expand or spread an iterable or an array. Example:

# Example: you can expand more than one array elements

# 

# Example: array de-structuring

# 

# Here, first returns first element of an array and …remaining stored remaining elements of an array and if we return remaining then it will give an array with some elements.

# Example: spread operator on objects.

# 

# LOGIN PAGE CODE WITH SPREAD OPERATOR

import React, { useState } from 'react';

const App = () => {

  const [fullInfo, setFullName] = useState({

    fName : "",

    lName : "",

    emailId : "",

    mobileNo : ""

  });

  const onSubmit = (event) => {

    event.preventDefault();

   }

  const inputEvent = (event) => {

    const {value, name} = event.target;

    setFullName((preValue) => {

     return{

**...preValue, //spread operator**

**[name]: value**

     };

    });

  }

  return(

    <><form onSubmit={onSubmit}>

      <h1>Hello {fullInfo.fName} {fullInfo.lName}</h1>

      <p>Email Id {fullInfo.emailId}</p>

      <p>Mobile Number {fullInfo.mobileNo}</p>

      <input type="text" placeholder="Enter Your First Name" onChange={inputEvent} value={fullInfo.fName} name='fName'/><br/><br/>

      <input type="text" placeholder="Enter Your Last Name" onChange={inputEvent} value={fullInfo.lName} name='lName'/><br/><br/>

      <input type="email" placeholder="Enter Your Email Id" onChange={inputEvent} value={fullInfo.emailId} name='emailId'/><br/><br/>

      <input type="number" placeholder="Enter Your Mobile Number" onChange={inputEvent} value={fullInfo.mobileNo} name='mobileNo'/><br/><br/>

      <button type='submit'>Click Me 😃</button>

      </form>

    </>

  )

}

# BUILDING A TODO LIST APP PROJECT IN REACTJS

# INCREMENTING AND DECREMENTING THE STATE VARIABLE ON BUTTON CLICKED IN REACT JS

const App = () => {

  const [count, setCount] = useState(0);

  const IncNum = () => {

    setCount(count+1);

  };

  const DecNum = () => {

    if(count>0)

    {

      setCount(count-1);

    }else{

setCount(0);

}

  };

  return(

    <>

        <h1>{count}</h1>

        <br/>

        <button onClick={IncNum}> Increment </button>

        <button onClick={DecNum}> Decrement </button>

    </>

  );

};

# HOW TO USE MATERIAL UI ICONS IN REACT

Go to command prompt and run this script for your project:

**npm install @material-ui/icons**

D:\reactlearning\reactlearn>npm install @material-ui/icons

And then run following script:

**npm install @material-ui/core**

D:\reactlearning\reactlearn>npm install @material-ui/core

# <https://mui.com/components/material-icons/>

# Use this link to add any material icons into your project by just importing the link like:

import AddIcon from '@material-ui/icons/Add'; **and use this component AddIcon where we want to use as:**

<button onClick={IncNum}> **<AddIcon />** </button>

# HOW TO USE MATERIAL UI ICONS IN REACT

[**https://mui.com/getting-started/installation/**](https://mui.com/getting-started/installation/)

Go to **component** section where you will find various components to import into your project.

import Button from '@material-ui/core/Button';

import Tooltip from '@material-ui/core/Tooltip;

<Tooltip title = “Delete”> **//text shown under button**

<Button onClick={IncNum}> <AddIcon /> </Button> **//use Button as component**

**</Tooltip>**

# NPM (NODE PACKAGE MANAGER)

# npm is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js. It consists of a command-line client, also called npm, and an online database of public and paid-for private packages called the npm registry.

# <https://www.npmjs.com/> you can install the package of any module and just import that component in your project

# Example: Display Digital Clock

# Install this package: npm install react-digital-clock

D:\reactlearning\reactlearn> npm install - -force react-digital-clock **//double hiphen**

Now, import component into your project: **import Clock from 'react-digital-clock';**

npm i --save-dev @types/react-digital-clock

Finally, you have to use your component: **<Clock />**

# TODO LIST CHALLENGE:

# https://www.thapatechnical.com/2020/06/create-todo-list-app-project-in-reactjs.html

# HOW TO ADD BOOTSTRAP IN REACT PROJECT

# <https://www.npmjs.com/package/bootstrap>

Several quick start options are available:

* [Download the latest release](https://github.com/twbs/bootstrap/archive/v5.1.3.zip)
* Clone the repo: git clone https://github.com/twbs/bootstrap.git
* Install with [npm](https://www.npmjs.com/): **npm install bootstrap**
* Install with [yarn](https://yarnpkg.com/): yarn add bootstrap
* Install with [Composer](https://getcomposer.org/): composer require twbs/bootstrap:5.1.3
* Install with [NuGet](https://www.nuget.org/): CSS: Install-Package bootstrap Sass: Install-Package bootstrap.sass

# Need to install bootstrap and then you have to import into your project. To identify which file you have to import, goto:

# node\_modules/bootstrap/dist/css/filename

# Example: node\_modules\bootstrap\dist\css\bootstrap.min.css

**import '../node\_modules/bootstrap/dist/css/bootstrap.min.css';**

React Bootstrap autocomplete is a component which predicts the words basing on the first few letters given by a user, while one is typing it.

check this Extension **" IntelliSense for CSS class names in HTML "** for auto-complete.

# CONTEXT API IN REACT

# React Context API is a way to essentially create global variables that can be passed around in a React app. This is the alternative to "prop drilling", or passing props from grandparent to parent to child, and so on. Context is often touted as a simpler, lighter solution to using Redux for state management. In a typical React application, data is passed top-down (parent to child) via props, but this can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.\

# React Context API allows you to easily access data at different levels of the component tree, without passing prop to every level.

# 

# Example:

# ComC.js

import { FirstName, LastName } from "./App";

const ComC = () => {

return (

    <>

    <FirstName.**Consumer**>{(fname) => {

        return(

        <LastName.Consumer>{(lname) => {

            return (<h1>My name is {fname} {lname}</h1>

        );

        }}

    </LastName.**Consumer**>

        );

    }}

    </FirstName.Consumer>

</>);

};

export default ComC;

# App.js

import ComA from './ComA';

import { createContext } from 'react';

const FirstName = **createContext**();

const LastName = createContext();

const App = () => {

  return  (<><FirstName.**Provider** value={'Aayush'}>

  <LastName.Provider value={'Agarwal'}><ComA/>

  </LastName.Provider>

  </FirstName.Provider></>);

};

export default App;

export {FirstName, LastName};

# useContext HOOK IN REACT

# useContext is used to simplify the complexity of Consumer part in context API. (above example)

# Difference:

|  |  |
| --- | --- |
| Without **useContext** | With **useContext** |
| import { FirstName, LastName } from "./App";  const ComC = () => {  return (      <>      <FirstName.**Consumer**>{(fname) => {          return(          <LastName.Consumer>{(lname) => {              return (<h1>My name is {fname} {lname}</h1>          );          }}      </LastName.**Consumer**>          );      }}      </FirstName.Consumer>  </>);  };  export default ComC; | import React, { **useContext** } from "react";  import { FirstName, LastName } from "./App";  import ComC from "./ComC";  const ComB = () => {      const fname = **useContext**(FirstName);      const lname = **useContext**(LastName);  return <h1>My name is {fname} {lname}</h1>;  };  export default ComB; |

# useEffect HOOK IN REACT

**What does useEffect do?** By using this Hook, you tell React that your **component needs to do something after render**. React will remember the function you passed (we’ll refer to it as our “effect”), and call it later after performing the DOM updates. In this effect, we set the document title, but we could also perform data fetching or call some other imperative API.

**Why is useEffect called inside a component?** Placing useEffect inside the component lets us access the count state variable (or any props) right from the effect. We don’t need a special API to read it — it’s already in the function scope. Hooks embrace JavaScript closures and avoid introducing React-specific APIs where JavaScript already provides a solution.

**Example**:

const App = () => {

const [num, setNum] = useState(0);

const [nums, setNums] = useState(0);

**useEffect**(()=>{ **//this will run every time when render will call component**

  alert('I am Clicked');

}, [**num**]); **//empty array means only one time and if you want useEffect to run for particular element, pass that value to this array**

  return (

    <>

    <button onClick={() => {

      setNum(num+1)

    }}>Fix {num}</button>

    <button onClick={() => {

      setNums(nums+1)

    }}>Submit {nums}</button>

    </>

  );

};

export default App;

**Example**: Change title of page after clicking on button

**useEffect(()=>{**

**document.title=`You Click me ${num} times`;**

**});**

# REACT API CALL USING AXIOS AND USEEFFECT IN REACTJS

# Axios is a library that helps us make HTTP Requests to external resources In our React applications, we often need to retrieve data from external APIs so it can be displayed in our web pages. Axios is designed to handle HTTP requests and responses. Axios deals with responses using Promises, so it's streamlined and easy to use in our code. Axios uses methods like get() and post() that perform HTTP GET and POST requests for retrieving or creating resources.

# Installing Axios using NPM: npm install axios

# Example:

import axios from "axios";

import React, { useEffect, useState } from "react";

const ComA = () => {

    const [num, setNum] = useState();

    const [name, setName] = useState();

    const [moves, setMoves] = useState();

**useEffect(()=>{**

**async function getData(){**

**const res = await axios.get(`https://pokeapi.co/api/v2/pokemon/${num}`);**

**setMoves(res.data.name);**

**setName(res.data.moves.length);**

**}**

**getData();**

**})**

    return (

    <>

    <h1>You Select Option {num}</h1>

    <h1>Name is {name}</h1>

    <h1>Moves are {moves}</h1>

    <select value={num} onChange={(event)=>{

        setNum(event.target.value);

    }}>

        <option value='1'>1</option>

        <option value='2'>2</option>

        <option value='3'>3</option>

        <option value='4'>4</option>

        <option value='5'>5</option>

    </select>

    </>

    );

};

export default ComA;

# REACT ROUTER

# Many modern websites are actually made up of a single page, they just look like multiple pages because they contain components that render like separate pages. These are usually referred to as SPAs - single-page applications. At its core, what React Router does is conditionally render certain components to display depending on the route being used in the URL (/ for the home page, /about for the about page, etc.).

# <https://www.freecodecamp.org/news/react-router-in-5-minutes/>

# You need to install React Router: npm install --save react-router-dom

# You'll need to import BrowserRouter, Route, and Routes from react-router-dom package

# You need to place your component inside BrowserRouter package:

**<BrowserRouter><App /></BrowserRouter>**

**Example:**

# About Page

const About = () => {

return <h1>This is About Page</h1>;

};

export default About;

# Contact Page

const Contact = () => {

return <h1>This is Contact Page</h1>;

};

export default Contact;

# Error Page

const ErrorPage = () => {

return <h1>This is Error Page</h1>;

};

export default ErrorPage;

**NavBar Page**

import { **NavLink** } from "react-router-dom";

const NavBar = () => {

return (

    <>

        <NavLink to='/'>About Us</NavLink>

        <NavLink to='/contact'>Contact</NavLink>

    </>

);

};

export default NavBar;

# App Page

import React from 'react';

import **{Route, Routes}** from "react-router-dom";

import About from './About';

import Contact from './Contact';

import ErrorPage from './ErrorPage';

const App = () => {

const Name = () => {

        return <h1>My Name is Aayush Agarwal</h1>;

    }

return (

    <>

**<Routes>**

**<Route exact path='/' element={<About/>}/>**

**<Route path='/contact' element={<Contact/>}/>**

**<Route exact path='/contact/name' element={<Name/>}/>**

**<Route element={<ErrorPage/>}/>**

**</Routes>**

    </>

);

};

export default App;

# Index Page

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import reportWebVitals from './reportWebVitals';

import {**BrowserRouter**} from "react-router-dom";

ReactDOM.render(

**<BrowserRouter><App /></BrowserRouter>**,

document.getElementById("root")

);

# REACT ROUTE RENDER METHOD

# When you use components (instead of render or children, below) the router uses React.createElement to create a new React element from the given component. That means if you provide an inline function to the component prop, you would create a new component every render. This results in the existing component unmounting and the new component mounting instead of just updating the existing component. When using an inline function for inline rendering, use the render.

**Example:**

**<Route exact path='/' render={() => {<About name = ‘About’/>}}/>**

# 

# useParams HOOKS IN REACT ROUTER

# useParams returns an object of key/value pairs of URL parameters. Use it to access match.params of the current Route. It provides access to search parameters in the URL. This was possible earlier only using match.params.

# Example: User.jsx

import React from "react";

import { useParams } from "react-router-dom";

const User = () => {

    const {fname, lname} = useParams();

return <h1>This is {fname} {lname} Page</h1>;

};

export default User;

# NavBar.jsx

import React from "react";

import { NavLink } from "react-router-dom";

const NavBar = () => {

return (

    <>

        <NavLink to='/'>About Us</NavLink>

        <NavLink to='/contact'>Contact</NavLink>

        <NavLink to='/user/aayush/agarwal'>User</NavLink>

    </>

);

};

export default NavBar;

# App.jsx

import logo from './logo.svg';

import React from 'react';

import {Route, Routes} from "react-router-dom";

import About from './About';

import Contact from './Contact';

import ErrorPage from './ErrorPage';

import NavBar from './NavBar';

import User from './User';

const App = () => {

    const Name = () => {

        return <h1>My Name is Aayush Agarwal</h1>;

    }

return (

    <>

    <NavBar/>

    <Routes>

        <Route exact path='/' element={<About/>}/>

        <Route path='/contact' element={<Contact/>}/>

        <Route path='/user/:fname/:lname' element={<User/>}/>

        <Route exact path='/contact/name' element={<Name/>}/>

        <Route path=’\*’ element={<ErrorPage/>}/>

    </Routes>

    </>

);

};

export default App;

# useLocation HOOK IN REACT ROUTER

# The useLocation hook returns the location object that represents the current URL. You can think about it like a useState that returns a new location whenever the URL changes. Provides access to the location prop in React Router. It is similar to window. location in the browser itself, but this is accessible everywhere as it represents the Router state and location. A primary use case for this would be to access the query params or the complete route string.

# useHistory HOOK IN REACT ROUTER

# It provides access to the history prop in React Router. This refers to the history package dependency that the router uses. A primary use case would be for programmatic routing with functions, like push, replace, etc. useHistory: The useHistory hook gives you access to the history instance that you may use to navigate. useLocation: The useLocation hook returns the location object that represents the current URL. useParams. useParams returns an object of key/value pairs of URL parameters.

# Example:

# 

# REACT LIVE SEARCH FILTER (IMAGE)

# Example:

# Search.jsx

import React, { useState } from "react";

import Sresult from "./Sresult";

const Search = () => {

    const [img, setImg] = useState("");

    const inputEvent = (event) => {

        const data = event.target.value;

        setImg(data);

    };

return (<><input type='text' placeholder="Search Anything" onChange={inputEvent} value={img}/>

{ img === "" ? null : <Sresult name={img}/>}

</>);

};

export default Search;

# Sresult.jsx

import React from "react";

const Sresult = (props) => {

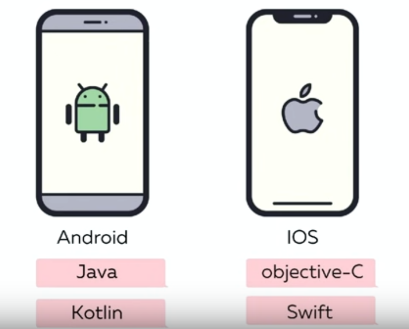
 const img = `https://api.unsplash.com/photos?${props.name}`;

return <img src={img} alt = 'Image'/>;

}; export default Sresult;

**REACT NATIVE**

React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android. It’s based on React, Facebook’s JavaScript library for building user interfaces, but instead of targeting the browser, it targets mobile platforms. In other words: web developers can now write mobile applications that look and feel truly “native,” all from the comfort of a JavaScript library that we already know and love. Plus, because most of the code you write can be shared between platforms, React Native makes it easy to simultaneously develop for both Android and iOS.



**What are the Prerequisites to Learn React Native?**

Javascript

React

HTML

CSS

# SETTING UP THE DEVELOPMENT ENVIRONMENT

If you are new to mobile development, the easiest way to get started is with **Expo CLI**. Expo is a set of tools built around React Native and, while it has many [features](https://expo.io/features), the most relevant feature for us right now is that it can get you writing a React Native app within minutes. You will only need a recent version of Node.js and a phone or emulator. If you'd like to try out React Native directly in your web browser before installing any tools, you can try out [Snack](https://snack.expo.dev/).

If you are already familiar with mobile development, you may want to use **React Native CLI**. It requires **Xcode** or **Android Studio** to get started. If you already have one of these tools installed, you should be able to get up and running within a few minutes. If they are not installed, you should expect to spend about an hour installing and configuring them.

Assuming that you have [Node 12 LTS](https://nodejs.org/en/download/) or greater installed, you can use npm to install the Expo CLI command line utility:

STEP 1: Node js Installation

STEP 2: IDE for React Native (VS Code Editor)

STEP 3: Local Environment

**CREATE A FOLDER IN DRIVE -> OPEN VS CODE EDITOR AND OPEN THAT FOLDER -> D:\reactnative> npm install -g expo-cli**

Then run the following commands to create a new React Native project called "NATIVE-APP":

**D:\reactnative> expo init native-app //NATIVE-APP IS NAME OF PROJECT**  
Choose a template: » blank a minimal app as clean as an empty canvas

**cd native-app  
npm start**

This will start a development server for you.

# CORE COMPONENTS IN REACT NATIVE

# <https://reactnative.dev/docs/intro-react-native-components>

# 

import { StatusBar } from 'expo-status-bar';

import { StyleSheet, Text, View } from 'react-native';

const App = () => {

  return (

    <View style={styles.container}>

      <Text>Hello World!</Text>

      <StatusBar style="auto" />

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

  },

});

export default App;

**How to Import & Export Custom Components in React Native**

Same as we have did in React

CREATE FOLDER UNDER PROJECT AS ‘**SRC’** -> FOLDER UNDER SRC AS ‘**SCREENS’** -> JS FILES UNDER SCREENS

**CustomComponents.js**

import { StatusBar } from 'expo-status-bar';

import { StyleSheet, Text, View } from 'react-native';

const CustomComponents = () => {

  return (

    <View style={styles.container}>

      <Text>Hello World!</Text>

      <StatusBar style="auto" />

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

  },

});

export default CustomComponents;

**App.js**

import CustomComponents from './src/screens/CustomComponents';

export default function App() {

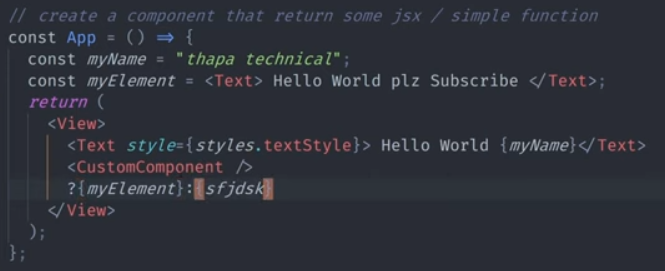
  return  <CustomComponents/>

}

# WHAT IS JSX & RULES TO WRITE PROPER JSX IN REACT NATIVE

# 

**RULE 3:**



**RULE 5:**



**CHALLENGE**:

import React from 'react'

import { StyleSheet, Text, View } from 'react-native'

const App = () => {

  const name = 'Aayush Agarwal';

  return (

    <View>

      <Text style={styles.textStyleOne}>Hello World</Text>

      <Text style={styles.textStyleTwo}>We love All</Text>

      <Text>Hi, My name is {name}</Text>

    </View>

  )

}

const styles = StyleSheet.create({

textStyleOne: {

  fontSize: 40,

  color: 'red',

  fontWeight: 'bold'

},

textStyleTwo: {

  fontSize: 30,

}

});

export default App

# FLATLIST IN REACT NATIVE

# <https://reactnative.dev/docs/flatlist>

The **FlatList** component displays the similar structured data in a **scrollable** list. It works well for large lists of data where the number of list items might change over time. The FlatList shows only those renders elements which are currently displaying on the screen, not all the elements of the list at once.

The FlatList component takes two required props: **data** and **renderItem**.

The **data** is the source of elements for the list, and **renderItem** takes one item from the source and returns a formatted component to render.

To implement the FlatList component, we need to import **FlatList** from **'react-native'** library.

# Example:

import React from 'react'

import { FlatList, StyleSheet, Text, View } from 'react-native'

const App = () => {

  const names = [{

    name: 'Aayush'

  },{

    name: 'Neelam'

  },{

    name: 'Rajiv'

  },{

    name: 'Nisha'

  }]

  return (

    <FlatList data={names} renderItem = {(element) => {

return <Text style={styles.textStyleOne}>{element.item.name}</Text>

    }}/> **//data={names}----------name of array**

  ) **//renderItem---------what to display**

}

const styles = StyleSheet.create({

textStyleOne: {

  fontSize: 40,

  color: 'red',

  fontWeight: 'bold'

}

});

export default App

# FLATLIST KEYS & HORIZONTAL \ VERTICAL SCROLLING EFFECTS IN REACT NATIVE

By default, the keyExtractor prop checks for properties like key and id (in that order). If any of the two is present in the original data structure, it will be considered a unique key by the FlatList component. In this case(as in the previous example), you do not have to explicitly use the keyExtractor prop.

If none of them are provided, the FlatList component will throw a warning "VirtualizedList: missing keys for items ...":

Now, let's consider a scenario where an array of data contains a unique key with each list item, but the name of the unique key is neither key nor id. It contains a unique key property with the name of userId.

const DATA\_WITH\_USER\_ID = [

{

userId: 1,

title: 'quidem molestiae enim'

},

{

userId: 2,

title: 'sunt qui excepturi placeat culpa'

}]

When rendering the list, you will see the warning in this case because the FlatList component doesn't recognize the userId as the key or id name in the original data structure.

For custom key names, such as userId in the example above, the **keyExtractor** prop is used. It extracts the unique key name and its value and tells the FlatList component to track the items based on that value.

For the above array of data, modify the FlatList component and use the **keyExtractor** prop to extract the key:

export default function App() {

const renderList = ({ item }) => {

return (

<View style={styles.listItem}>

<Text style={styles.listItemText}>{item.title}</Text>

</View>

);

};

return (

<View style={styles.container}>

<FlatList data={DATA\_WITH\_USER\_ID} renderItem={renderList} **keyExtractor**={item => item.userId}/>

</View>

);

}

Flatlist have one more prop which is horizontal that display your items horizontally. By default, it displays vertically. In horizontal case, it shows scroll bar and if we donot want to display that scrollbar, then we can use another props: **showsHorizontalScrollIndicator={false}**

return (

<View style={styles.container}>

<FlatList data={DATA\_WITH\_USER\_ID} renderItem={renderList} **keyExtractor**={item => item.userId} **horizontal showsHorizontalScrollIndicator={false}**

/>

</View>

);

|  |  |
| --- | --- |
| **SCROLLABLE** | **SCROLLABLE** |

# HOW TO ADD OR USE IMAGES IN REACT NATIVE

A React component for displaying different types of images, including network images, static resources, temporary local images, and images from local disk, such as the camera roll.

This example shows fetching and displaying an image from local storage as well as one from network and even from data provided in the 'data:' uri scheme.

# Example:

import React from 'react'

import { Image, StyleSheet, Text, View } from 'react-native'

const App = () => {

  return (

    <View>

      <Text>Hello, Image List</Text>

**<Image style={styles.textStyleOne} source={require('../native-app/assets/name.jpg')}/>**

    </View>

  )

}

const styles = StyleSheet.create({

textStyleOne: {

  height:200,

  width: 200

}

});

export default App

# BUTTON IN REACT NATIVE

# A basic button component that should render nicely on any platform. Supports a minimal level of customization.

<Button  
 onPress={() => {console.log(“Hello”)}}  
 title="Learn More"

disabled  
 color="#841584"  
 accessibilityLabel="Learn more about this purple button"  
/>

# Touchable Opacity in React Native

A wrapper for making views respond properly to touches. On press down, the opacity of the wrapped view is decreased, dimming it.

Opacity is controlled by wrapping the children in an Animated.View, which is added to the view hierarchy. Be aware that this can affect layout.

# Example:

import React, { useState } from "react";

import { StyleSheet, Text, TouchableOpacity, View } from "react-native";

const App = () => {

const [count, setCount] = useState(0);

const onPress = () => setCount(prevCount => prevCount + 1);

return (

<View style={styles.container}>

<View style={styles.countContainer}>

<Text>Count: {count}</Text>

</View>

**<TouchableOpacity**

**style={styles.button}**

**onPress={onPress}**

**> //<Button> has title prop whereas TouchableOpacity not**

**<Text>Press Here</Text> //<Button> is self closing whereas TouchableOpacity not**

**</TouchableOpacity>**

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: "center",

paddingHorizontal: 10

},

button: {

alignItems: "center",

backgroundColor: "#DDDDDD",

padding: 10

},

countContainer: {

alignItems: "center",

padding: 10

}

});

export default App;

# PROPS IN REACT NATIVE | PARENT CHILD RELATIONSHIP IN REACT NATIVE

# Example: ImageFile.js

# C:\Users\ap_aa\OneDrive\Desktop\Untitled.png

# CardDetail.js

# 

# OR

# 

**TIPS and TRICKS**

**Example:** if we want to use external url for image, we can use **uri** under source property



Example: If we want to redirect on pressing of button, we have to import **Linking** component of react-native and use under onPress property of Button



# HOW TO USE GOOGLE FONTS IN REACT NATIVE

**What is Expo Fonts?**

Expo has first-class support for all fonts listed in Google Fonts with these packages you can quickly integrate any font or font variants. The first thing you'll need is a font file. The two officially supported font formats for the Expo platform are OTF and TTF. If your font is in another format, try to convert it to one of those formats. If you don't want to use a custom font, your best bet is to just use the platform default font by not specifying a font family. Since your fonts won't be ready right away, it is generally a good practice to not render anything until the font is ready. A great way we can do that is to use the AppLoading component. In its simplest form, you can just render it while you're waiting for your app to load.

**Example**: If we want to use google fonts, then do the following steps:

**Step 1**: Go to <https://directory.vercel.app/>

**Step 2**: Choose which font you want to select:



**Step 3**: You need to install packages for using google fonts as:

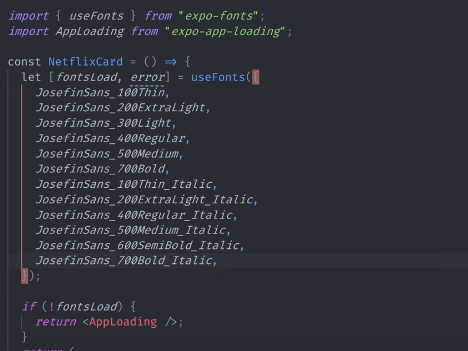
PS D:\reactnative\native-app> **expo install @expo-google-fonts/josefin-sans**

PS D:\reactnative\native-app> **expo install expo-app-loading**

**Step 4**: Import all libraries of that font as:

import { JosefinSans\_100Thin, JosefinSans\_200ExtraLight, JosefinSans\_300Light, JosefinSans\_400Regular, JosefinSans\_500Medium, JosefinSans\_600SemiBold, JosefinSans\_700Bold, JosefinSans\_100Thin\_Italic, JosefinSans\_200ExtraLight\_Italic, JosefinSans\_300Light\_Italic, JosefinSans\_400Regular\_Italic, JosefinSans\_500Medium\_Italic, JosefinSans\_600SemiBold\_Italic, JosefinSans\_700Bold\_Italic } from '@expo-google-fonts/josefin-sans'

**Step 5**: Use this font as:



# CSS in React Native | Text Style Props in React Native

<https://reactnative.dev/docs/text-style-props>

# BOX MODEL IN REACT NATIVE | LAYOUT & VIEW PROPS IN REACT NATIVE

# What is box model?

# Box model is the combination of Content, Padding, Border and Margin.

# Padding simply gives space inside the box which has various properties like padding left, padding right, padding top, padding bottom and then padding vertical and horizontal.

# Border simply represent the outer edge of a box or an object just like a fencing that separates two different objects. It also has various properties like Border width and Border color.

# Margin simply gives space outside the box which have some properties like Margin right, Margin left, Margin top, Margin bottom, then Margin vertical and horizontal.

# 

# <https://reactnative.dev/docs/layout-props>

# <https://reactnative.dev/docs/view-style-props>

# Example:

const App = () => {

  return (

    <View>

    <View style={styles.parentStyle}>

      <Text style=**{[styles.textStyleOne, styles.commonStyle]}**>Image 1</Text> **//Multiple Styles**

      <Text style={[styles.textStyleTwo, styles.commonStyle]}>Image 2</Text>

      <Text style={[styles.textStyleThree, styles.commonStyle]}>Image 3</Text>

      </View>

    </View>

  )

}

const styles = StyleSheet.create({

parentStyle: {

width: "95%",

marginTop: 100,

marginHorizontal: 10,

borderWidth: 3,

borderColor: "#101820FF"

},

**textStyleOne**: {

 backgroundColor: "red"

},

textStyleTwo: {

backgroundColor: "green"

},

textStyleThree: {

  backgroundColor: "blue"

},

**commonStyle**: {

margin: 10,

padding: 10,

textAlign: "center",

borderColor: "black",

borderWidth: 4

}

});

export default App

# REACT NATIVE FLEXBOX LAYOUT

**What flex box is in react native?**

Flexbox works the same way in React Native as it does in CSS on the web, with a few exceptions. There are few difference in flex box in react native and in css:-

The defaults are different, with flexDirection defaulting to column instead of row, alignContent defaulting to flex-start instead of stretch, flexShrink defaulting to 0 instead of 1, the flex parameter only supporting a single number. We have seen various values in flex direction, like column, row, column-reverse, row-reverse.

**What is Justify Content?**

It describes how to align children within the main axis of their container. For example, you can use this property to center a child horizontally within a container with flexDirection set to row or vertically within a container with flexDirection set to column. We have seen various values in Justify Content like flex-start, flex-end, center, space-between, space-around.

**What is Align Items?**

It describes how to align children along the cross axis of their container. It is very similar to justifyContent but instead of applying to the main axis, alignItems applies to the cross axis. We have seen various values in Align Items like stretch, flex-start, flex-end, center. Then we have seen Align Self, It has the same options and effect as alignItems but instead of affecting the children within a container, you can apply this property to a single child to change its alignment within its parent. alignSelf overrides any option set by the parent with alignItems.

<https://reactnative.dev/docs/flexbox>

# CSS POSITION RELATIVE ABSOLUTE IN REACT NATIVE

In react native, position is a type of an element that defines how it is positioned within its parent. In react native there are two types of positions:-

1. Absolute position

2. Relative position

**What is Absolute position?**

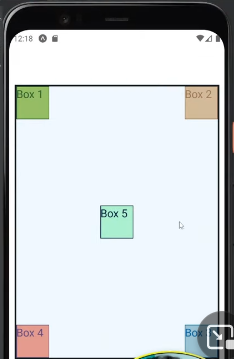
When positioned absolutely, an element doesn't take part in the normal layout flow. It is instead laid out independent of its siblings. The position is determined based on the top, right, bottom, and left values.

**What is Relative position?**

By default, an element is positioned relatively. This means an element is positioned according to the normal flow of the layout, and then offset relative to that position based on the values of top, right, bottom, and left. The offset does not affect the position of any sibling or parent elements.

<https://reactnative.dev/docs/flexbox>

**Example**:



position: “relative” **//parent container**

position: “absolute” **//common container**

right: 0 **//box container 2**

bottom: 0 and right: 0 **//box container 3**

bottom: 0 **//box container 4**

 **//box container 5 (we need subContainer)**

# COUNTER APP USING USESTATE HOOKS IN REACT NATIVE | STATE MANAGEMENT IN REACT NATIVE

**Why we need useState Hooks?**

We are not able to update the value using pure javascript so, we use hooks to update the value.

**What is useState Hooks?**

useState is a Hook that lets us add React state to function components. It simply means that we can now declare state variables to functional components. It returns a pair of values, counterValue which holds the number of button clicks, setCounter which update the counterValue.

**Example: Increment, Decrement and Reset Value on click of Button**

import React, {useState} from 'react'

import { Button, Image, StyleSheet, Text, View, TouchableOpacity } from 'react-native'

const App = () => {

  const [counter, setCounter] = useState(0);

  return (

    <View>

    <Text style={styles.title}>

      {counter}

    </Text>

    <View style={styles.contentStyle}>

    <TouchableOpacity onPress={() => {

        setCounter(counter + 10);

          }} style={styles.appButtonContainer}>

    <Text style={styles.appButtonText}>+10</Text>

  </TouchableOpacity>

  <TouchableOpacity onPress={() => {

        setCounter(0);

      }} style={styles.appButtonContainer}>

    <Text style={styles.appButtonText}>Reset</Text>

  </TouchableOpacity>

  <TouchableOpacity onPress={() =>{

        if(counter > 0)

        setCounter(counter - 10);

          }} style={styles.appButtonContainer}>

    <Text style={styles.appButtonText}>-10</Text>

  </TouchableOpacity>

  </View>

  </View>

  )

}

const styles = StyleSheet.create({

title: {

width: "95%",

marginTop: 100,

textAlign: "center",

fontSize: 50

},

contentStyle: {

alignItems: 'center'

},

appButtonContainer: {

padding: 20,

backgroundColor: 'black',

margin: 10,

width: 200,

alignItems: 'center'

},

appButtonText: {

 fontWeight: 'bold',

 color: 'white'

}

});

export default App

**RANDOM COLOR GENERATOR**

import React, {useState} from 'react'

import { Button, Image, StyleSheet, Text, View, TouchableOpacity, FlatList } from 'react-native'

const App = () => {

  const [newColor, setNewColor] = useState([]);

  const randomColor = () => {

  const red = Math.floor(Math.random() \* 256);

  const green = Math.floor(Math.random() \* 256);

  const blue = Math.floor(Math.random() \* 256);

  return `rgb(${red},${green},${blue})`;

  }

  return (

    <View style={styles.contentStyle}>

    <TouchableOpacity onPress={() => {

        setNewColor([...newColor, randomColor()]);

          }} style={styles.appButtonContainer}>

    <Text style={styles.appButtonText}>Random Color Generator</Text>

  </TouchableOpacity>

  <FlatList data={newColor} renderItem={({item}) => {

    return (

<View style={styles.imageContainer}>

<View style={{

  backgroundColor: item, width: 100, height: 100, borderRadius: 5, marginTop: 5, display: 'flex', justifyContent: 'center',

  alignItems: 'center', width: '80%'

}}>

    <Text style={styles.appButtonText}>{item}</Text></View>

  </View>

    )

  }} keyExtractor={(key) => key}/>

  </View>

  )

}

const styles = StyleSheet.create({

contentStyle: {

marginTop: 100

},

appButtonContainer: {

backgroundColor: "#00b0ff",

color: '#eee',

paddingVertical: 10,

paddingHorizontal: 15,

display: 'flex',

justifyContent: 'center',

alignItems: 'center',

borderRadius: 5

},

appButtonText: {

 color: 'white',

 textTransform: 'uppercase'

},

imageContainer: {

marginVertical: 50,

paddingHorizontal: 30,

width: '100%',

display: 'flex',

justifyContent: 'center',

alignItems: 'center',

fontWeight: 'bold'

}

});

export default App

# REST API, ASYNC-AWAIT, TRY CATCH & USEEFFECT HOOK IN REACT NATIVE

# Example: Fetch API and Load data on webpage using Flatlist

import React, {useState, useEffect} from 'react'

import { Image, StyleSheet, Text, View, FlatList, ActivityIndicator } from 'react-native'

const App = () => {

  const [myUserData, setMyUserData] = useState();

  const [isLoading, setIsLoading] = useState(true);

const getUserData = async () => {

  try {

    const response = await fetch('https://thapatechnical.github.io/userapi/users.json');

    const myData = await response.json();

    setMyUserData(myData);

    setIsLoading(false);

    console.log(myData)

  } catch (error) {

    console.log(error)

  }

}

useEffect(() => {

  getUserData();

}, [])

  return (

    <View style={styles.mainContainer}>

    {isLoading ? (<View><ActivityIndicator size='large' color='#0000ff'/></View>) : (<View>

    <Text style={styles.mainHeader}>List of Students</Text>

   <FlatList data={myUserData} renderItem = {({item}) => {

     return <View style={styles.card}>

       <View style={styles.imgContainer}>

<Image style={styles.imgStyle} resizeMode="cover" source={{uri: item.image}}/>

       </View>

       <View>

         <View style={styles.bioDataContainer}>

<Text style={styles.bioData}>Bio-Data</Text>

<Text style={styles.idNumber}>{item.id < 10 ? `#0${item.id}` : `#${item.id}`}</Text>

         </View>

         <View style={styles.mainContain}>

<Text style={styles.myName}>Name: {item.name}</Text>

<Text style={styles.myName}>Email: {item.email}</Text>

<Text style={styles.myName}>Mobile: {item.mobile}</Text>

         </View>

       </View>

     </View>

   }}/>

   </View>)}

  </View>

  )

}

const styles = StyleSheet.create({

  loader: {

    minHeight: "100%",

    display: 'flex',

    justifyContent: 'center',

    alignItems: 'center'

  },

  mainContainer: {

    width: "100%",

    paddingTop: 50,

    backgroundColor: "#b696d7",

    display: 'flex',

    justifyContent: 'center',

    alignItems: 'center'

},

bioDataContainer: {

  width: "100%",

display: 'flex',

flexDirection: "row",

justifyContent: 'space-between',

alignItems: 'center',

backgroundColor: "#353535",

paddingVertical: 10,

},

card: {

  width: 250,

  backgroundColor: "#fff",

  borderRadius: 5,

  display: 'flex',

flexDirection: "column",

justifyContent: 'space-between',

margin: 20

},

idNumber: {

  fontSize: 20,

  color: 'rgba(255,255,255,0.5)'

},

bioData: {

  fontSize: 30,

  color: "#fff"

},

mainHeader: {

  fontSize: 30,

  color: '#fff'

},

imgContainer: {

  padding: 10

},

imgStyle: {

  width: '100%',

  height: 180

},

mainContain: {

  width: "100%",

backgroundColor: "#353535",

paddingVertical: 10,

},

myName: {

  color: 'white',

  padding: 5

}

});

export default App