# React Notes – Mazan Labeeb

# Environment Setup:

* VS Code
* Node and NPM
* yarn

# Install

npx create-react-app app-name

There are two types of components in react.

* Functional Components
* Class Components

Both do almost the same work in different ways

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

# Class Components:

import { Component } from react;

class **App** extends **Component** {

**render**() {

    return (

      HTML;

);

  }

}

# 

A picture containing graphical user interface

Description automatically generated

# State and Constructor:

class **App** extends **Component**{

    constructor(){

        super();

        this.state = {

            name: "Mazan Labeeb"

        }

    }

**render**(){

        return (

*// NOW HERE WE CAN RENDER "MAZAN LABEEB"*

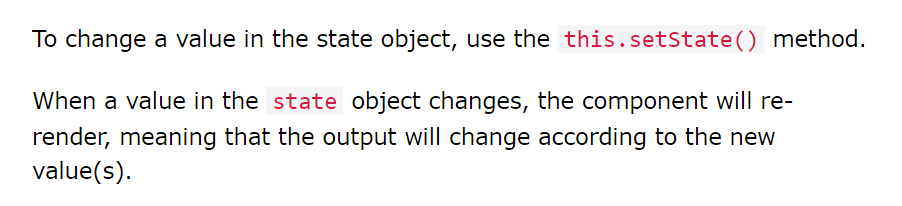
*// AS {this.state.name}*

        );

    }

}

# setState():



This function updates ***this.state*** and re-render the component.

Let say we have a state as in above example:

this.state = { username: “Mazan”}

and now we want to change the this.state.username to a different value. One simple method to do is :

this.state.username = “Nomi”

But, this will not re-render the DOM. In order to do so, we use the **setState()** method as following:

<button *onclick*={() => {

    this.**setState**({ username: "Nomi" })

    }

}>Re-Render</button>

setState() perform asynchronously. We can use it in callback function mode to make the code more professional.

this.**setState**((state, props)=>{

    return {

// *state = this.state*

*// update state*

    }

}, ()=> console.**log**("I am printed after the DOM is re-rendered"));

# Map Function to render Array Objects:

this.state.user = [

    { name : "Mazan Labeeb" },

    { name : "Nomi" },

    { name : "Dawood" },

]

**render**(){

    return (

        {this.state.user.map((row)=>{

**return** (

                <b>{row.name}</b>

            )

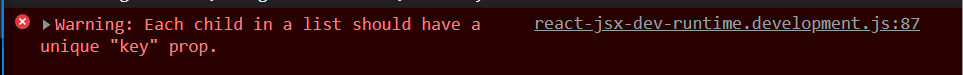
        })}

    );

}

The upper code will produce the following error in the console:

**Key Warning:**



This is because every entry should have a unique value. This is required so that the react can perform better and efficiently render.

In order to fix this add key={} to the parent or individual element:

**render**(){

    return (

        {

            this.state.user.map((row) => {

**return** (

                    <div key={row.id}>

                        <b>{row.name}</b>

                    </div>

                )

            })

        }

    );

}

# Lifecyle Method componentDidMount()

Graphical user interface, text

Description automatically generated with medium confidence

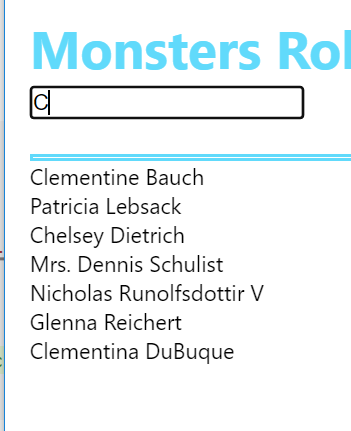
**Use Case:**

Such as to call the fetch API request and render the page when it is loaded.

Graphical user interface

Description automatically generated

# App Project:



# Algorithm:

* Use the “onChange” event handler on the input field.
* Use the “filter” array function to find inside the array.
* Use the “includes” string function along with “toLowerCase” function to compare two strings.

**Don’t change the original “this.state.users” while filtering out the array. Rather, make another array say “filteredUsers” inside the “render” function before the “return” and then render while mapping out of this “filteredUsers” array. As this will sort out the problem of retaining the original state.**

**A picture containing shape

Description automatically generated**



# Performance and Optimization

# Use Short Hands

If variable and property have the same name, then it can be pass to an object as following:

let x = 5;

let y = 10;

let obj = { x , y};

console.**log**(obj);

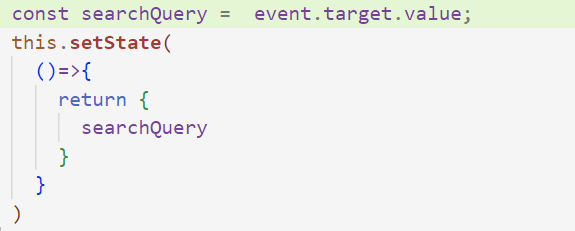
*// {*

*//     x : 5,*

*//     y : 10*

*// }*

**Use Case:**



# Don’t make Anonymous Functions inside render method:

Anonymous functions are the functions that are not stored inside a variable.

e.g.,

**onclick** = function (){

**alert**();

}

Don’t make them inside the render method as it will initialize each time the state is changed and will make our app slow. Rather, make them outside the render method and call them using shorthand method:

Onclick = { this.functionName }

Rather than

~~onclick = {~~ **~~function~~** ~~() { this.~~**~~functionName~~**~~( event ) } }~~

**Even better way is to use destructor:**

    const {**onSearchChange**} = this;

And then consequently,

*onChange*={ **onSearchChange** }

# Components Practices:

* Make a folder name Components.
* Inside this folder make a folder for each component with its name. e.g., card-list
* Make a jsx file as for the component e.g., card-list.component.jsx

**Jsx file**

import { Component } from "react";

class **CardList** extends **Component**{

**render**(){

    }

}

export default **CardList**;

**App.js**

import CardList from './components/card-list/card-list.component.jsx';

and inside render() function

        <**CardList** />

**Note**: Component class name should start with a capital letter.

# Props

Props are arguments passed into React components.

Props are passed to components via HTML attributes.

props stands for properties.

<**CardList** *filteredUsers*={filteredUsers} />

The component receives the argument as a props object:

const { filteredUsers } = this.props;

**Difference between Props and State:**

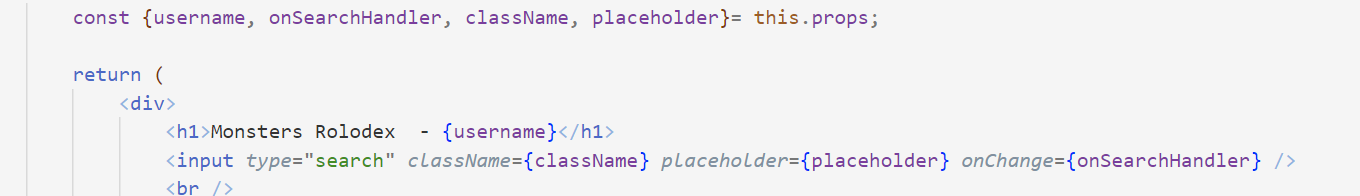
**Props are used to pass data from one component to another.** **The state is a local data storage that is local to the component only and cannot be passed to other components**.

**Component Gets Updated when its props are changed.**

It is a practice to pass all the attributes from one component to the other as:

**Root Component:**

<**SearchBox** *username*={username} *className* = {"search-box"} *onSearchHandler* = {**onSearchChange**} *placeholder* = {"Search..."}/>

**Child Component:**

# Functional Components

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand,

(Pure, Impure functions and Side Effects)

Syntax:

Functional Components have similar syntax to Class Components but in functional components there is no lifecycle methods. Rather, the strategy of designing the program is kept same as Class Components.

const **App** = () => {

  return (

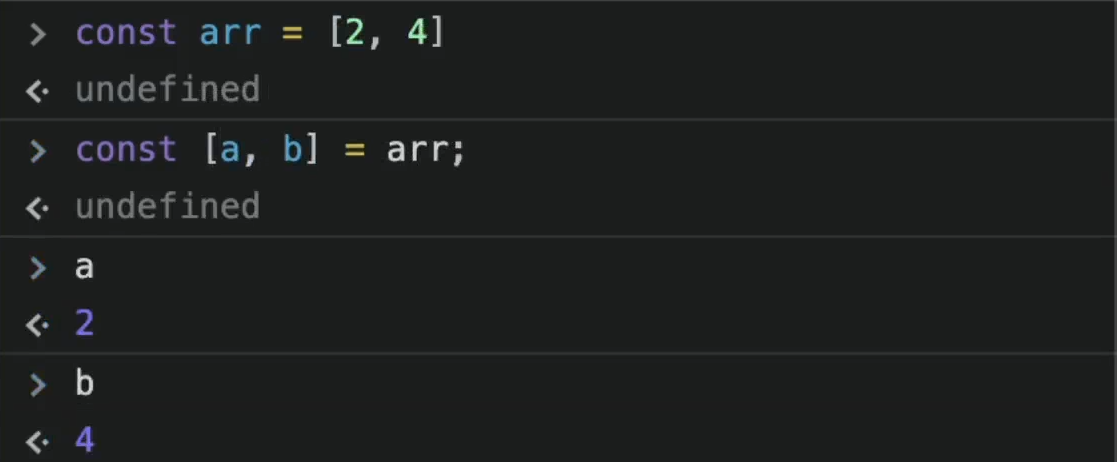
    <div>Mazan Labeeb</div>

  )

}

export default App;

**Array Destructor:**



Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

Although Hooks generally replace class components, there are no plans to remove classes from React.

# React Hooks

Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

Although Hooks generally replace class components, there are no plans to remove classes from React.

# Graphical user interface, text, website Description automatically generated

# What is a Hook?

Hooks allow us to "hook" into React features such as state and lifecycle methods.

## Hook Rules

There are 3 rules for hooks:

* Hooks can only be called inside React function components.
* Hooks can only be called at the top level of a component.
* Hooks cannot be conditional

**Note:** Hooks will not work in React class components.

# React useState Hook

The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.

To use the useState Hook, we first need to import it into our component.

import { useState } from "react";

 const [searchField, **setSearchField**] = **useState**(""); // [value, setValue]

Now we can track the state by   console.**log**(searchField);

and change the state as:

**setSearchField**(searchQuery);

Complete Example:



Notice that the whole App() component will be re-render whenever the