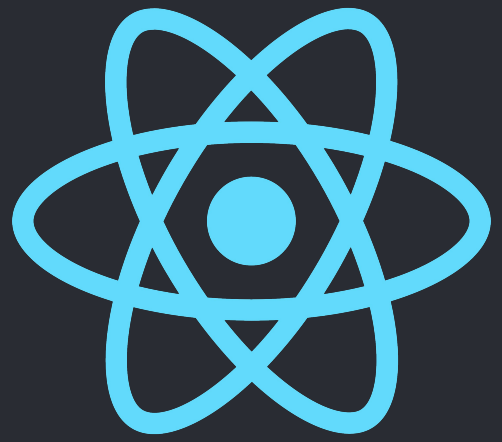
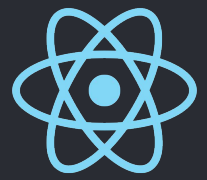


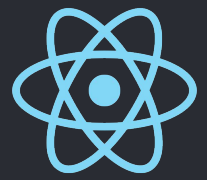
Learn React in 90 Minutes

Learn once, write anywhere!



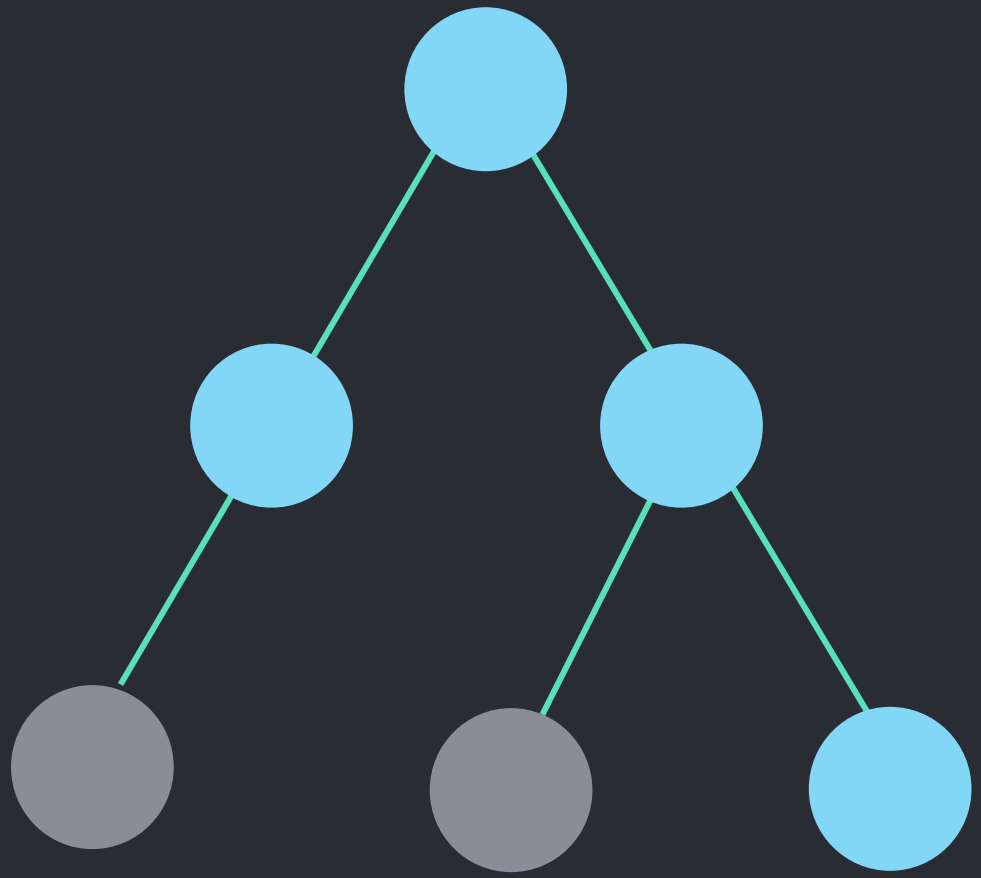
Introduction to React

- What is React.
- When was it released.
- Its effect on the developing scene.
- What distinguishes it from its competitors.
- Why is it called React.

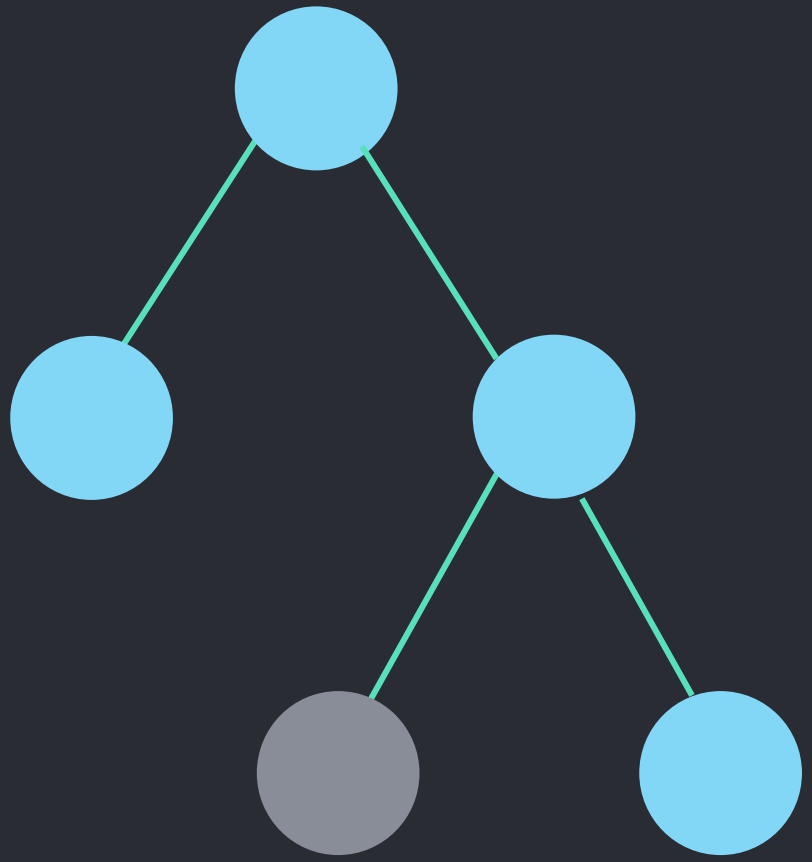


How does React deal with DOM

Virtual DOM

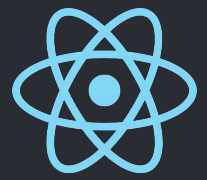


Real DOM

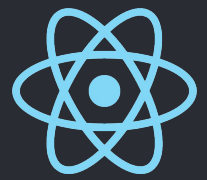


How to create a react app

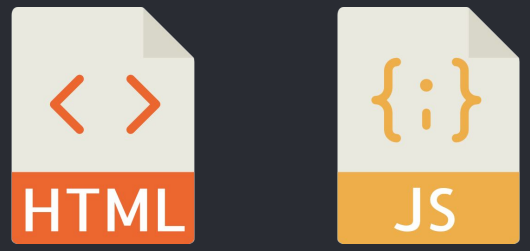
```
npx create-react-app my-app
cd my-app
npm start
```



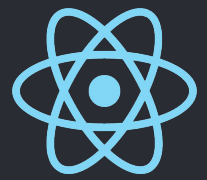
JSX



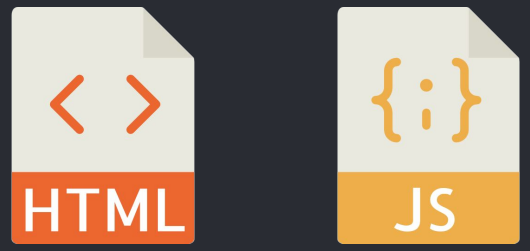
Before using React



Two seperate files



After using React



Only one file

JS

Index.html

```
1   <!DOCTYPE html>
2
3   <html lang="en">
4       <head>
5         <title>React App</title>
6       </head>
7
8       <body>
9       <div id="root"></div>
10      </body>
11
12  </html>
13
```

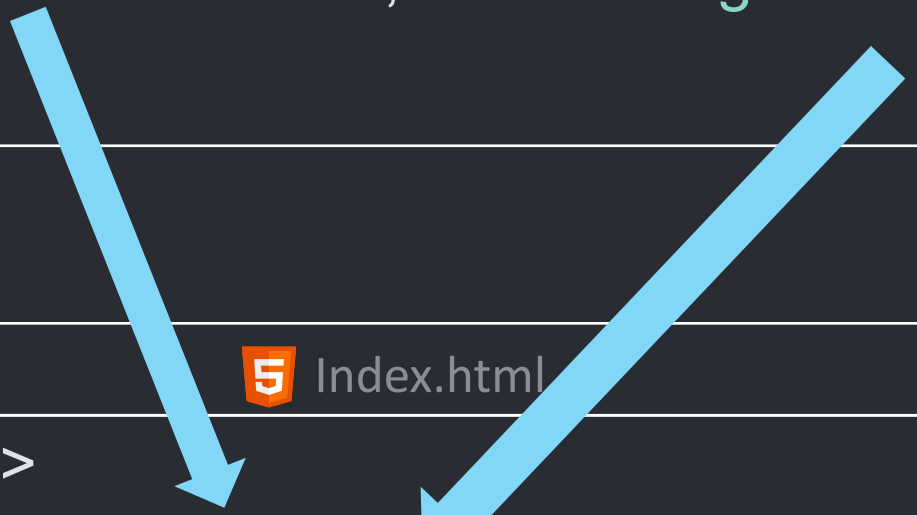
main.jsx

```jsx
var React = require("react");
var ReactDOM = require("react-dom");


ReactDOM.render(<h1>Hello World!</h1>, document.getElementById("root"));
```

```
ReactDOM.render( WHAT TO SHOW , WHERE TO SHOW );
```

main.jsx

```jsx
var React = require("react");
var ReactDOM = require("react-dom");


ReactDOM.render(<h1>Hello World!</h1>, document.getElementById("root"));

```

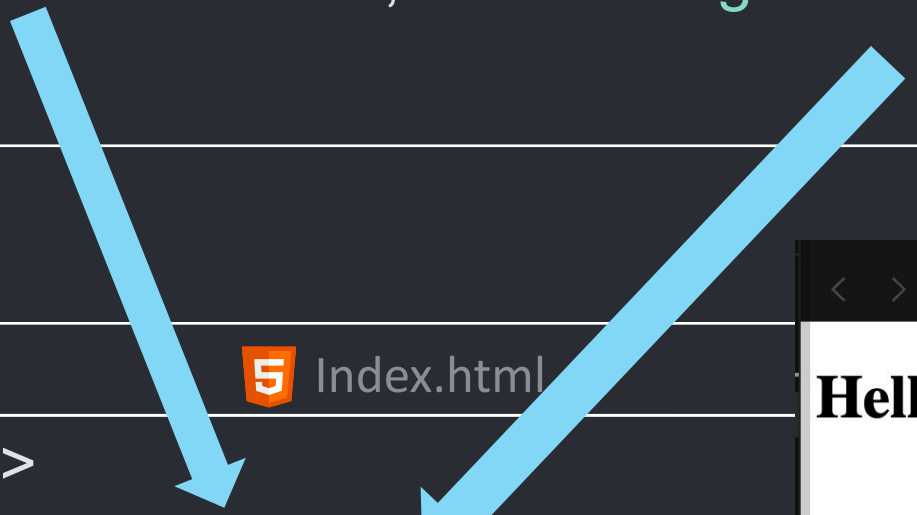


Index.html

```html
<body>
 <div id="root"></div>
</body>
```

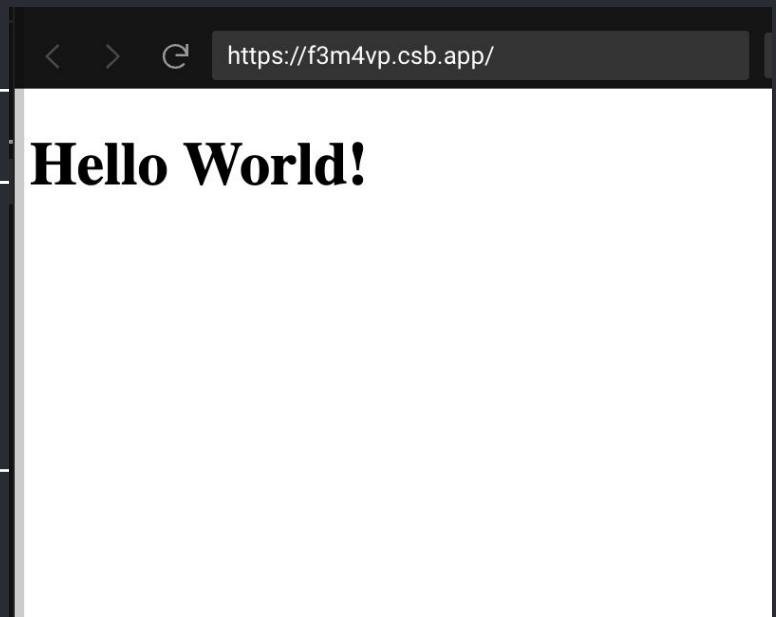
main.jsx

```jsx
var React = require("react");
var ReactDOM = require("react-dom");


ReactDOM.render(<h1>Hello World!</h1>, document.getElementById("root"));

```



Index.html

```html
  <body>
    <div id="root"></div>
  </body>
```



main.jsx

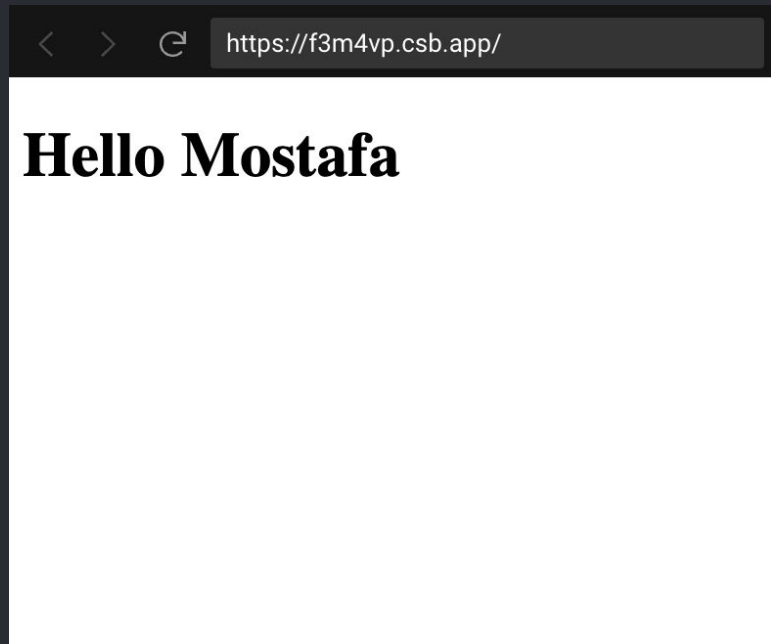
```jsx
var React = require("react");
var ReactDOM = require("react-dom");

const name = "Mostafa";

ReactDOM.render(<h1>Hello {name} </h1>, document.getElementById("root"));
```

JS inside HTML

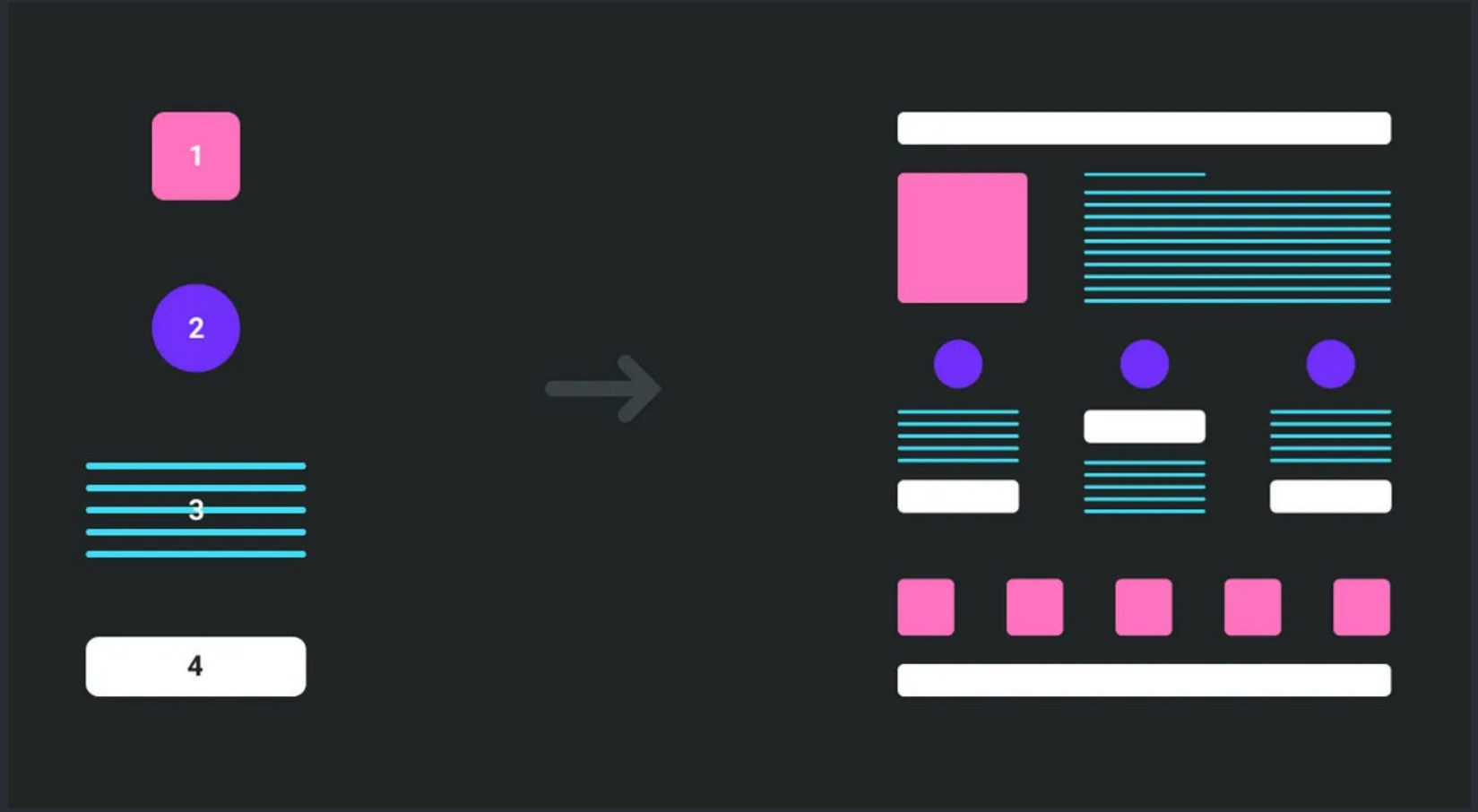
main.jsx

```jsx
var React = require("react");
var ReactDOM = require("react-dom");

const name = "Mostafa";

ReactDOM.render(<h1>Hello {name} </h1>, document.getElementById("root"));
```

JS inside HTML



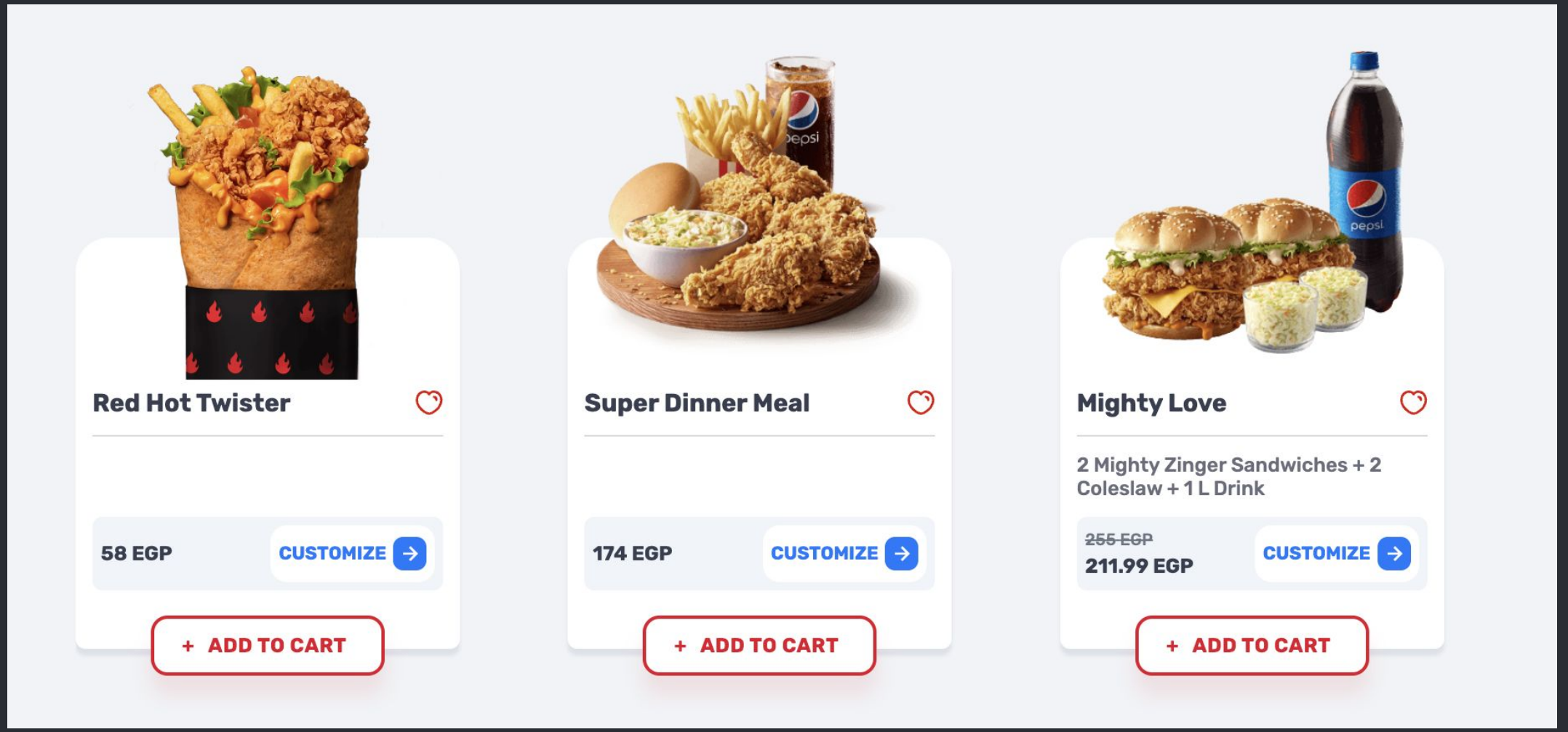
Components



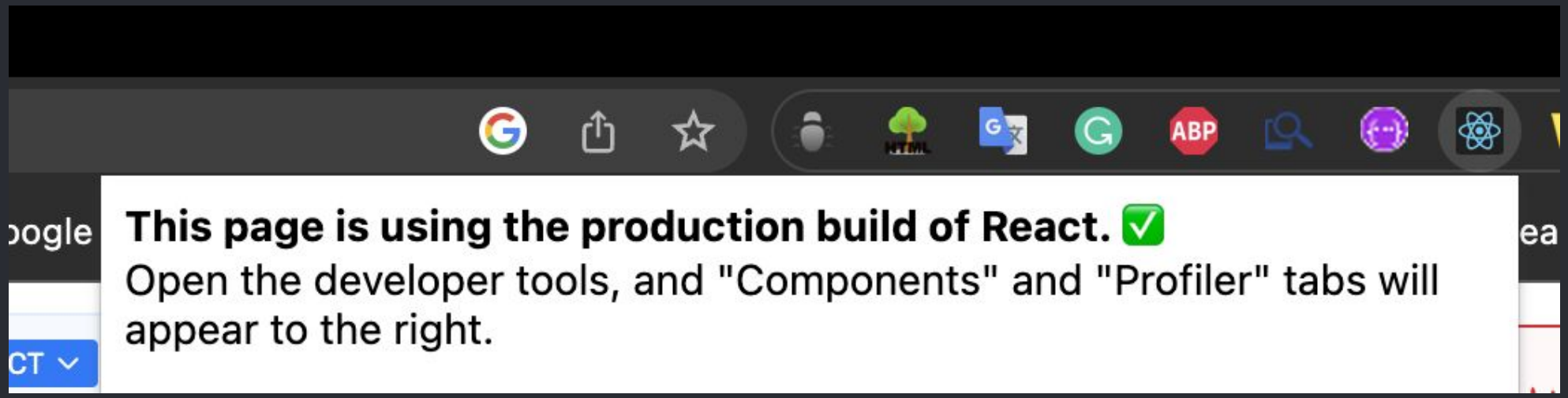
Components Advantage

- Reusable
- Easy to Understand

- Reusable


Red Hot Twister
58 EGP
CUSTOMIZE
+ ADD TO CART
Super Dinner Meal
174 EGP
CUSTOMIZE
+ ADD TO CART
Mighty Love
2 Mighty Zinger Sandwiches + 2 Coleslaw + 1 L Drink
255 EGP
211.99 EGP
CUSTOMIZE
+ ADD TO CART

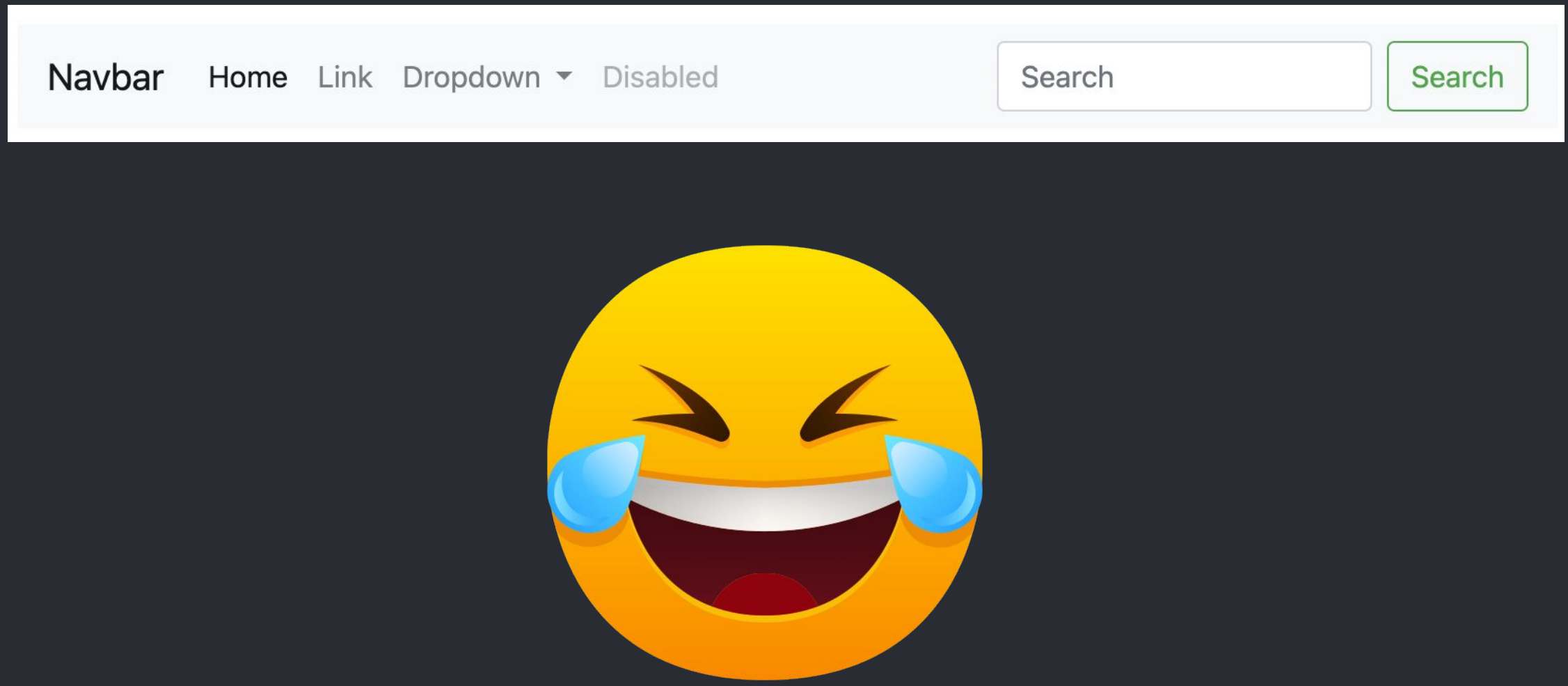
- Reusable



The name of the extantion :React Developer Tools

• Easy to Understand

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
<a class="navbar-brand" href="#">Navbar</a>
<button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
<span class="navbar-toggler-icon"></span>
</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">
<ul class="navbar-nav mr-auto">
<li class="nav-item active">
<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
</li>
<li class="nav-item">
<a class="nav-link" href="#">Link</a>
</li>
<li class="nav-item dropdown">
<a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"
data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
Dropdown
</a>
<div class="dropdown-menu" aria-labelledby="navbarDropdown">
<a class="dropdown-item" href="#">Action</a>
<a class="dropdown-item" href="#">Another action</a>
<div class="dropdown-divider"></div>
<a class="dropdown-item" href="#">Something else here</a>
</div>
</li>
<li class="nav-item">
<a class="nav-link disabled" href="#">Disabled</a>
</li>
</ul>
<form class="form-inline my-2 my-lg-0">
<input class="form-control mr-sm-2" type="search" placeholder="Search"
aria-label="Search">
<button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>
</form>
</div>
</nav>
```


Navbar
Home
Link
Dropdown
Disabled
Search
Search

Using components

```
<NavBar />
<PageContent />
<Footer />
```

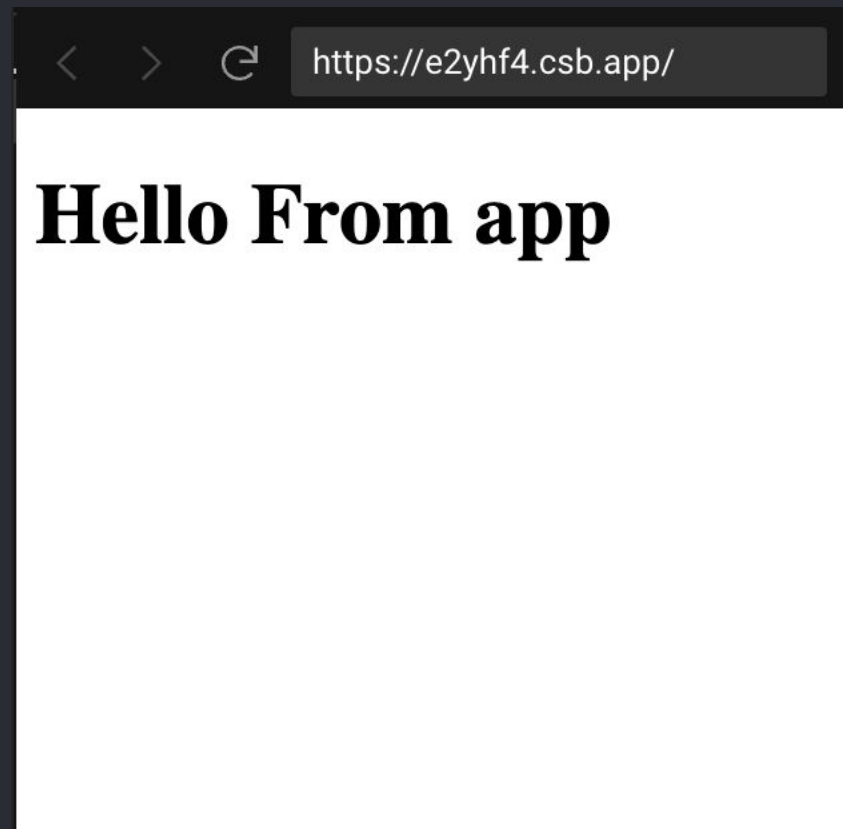
Let’s build our first component

main.jsx

```jsx
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(<App/>,document.getElementById("root"));

```

App.jsx

```jsx
import React from "react";
function App() {

  return (<h1>Hello From app</h1>);

}
export default App;


```

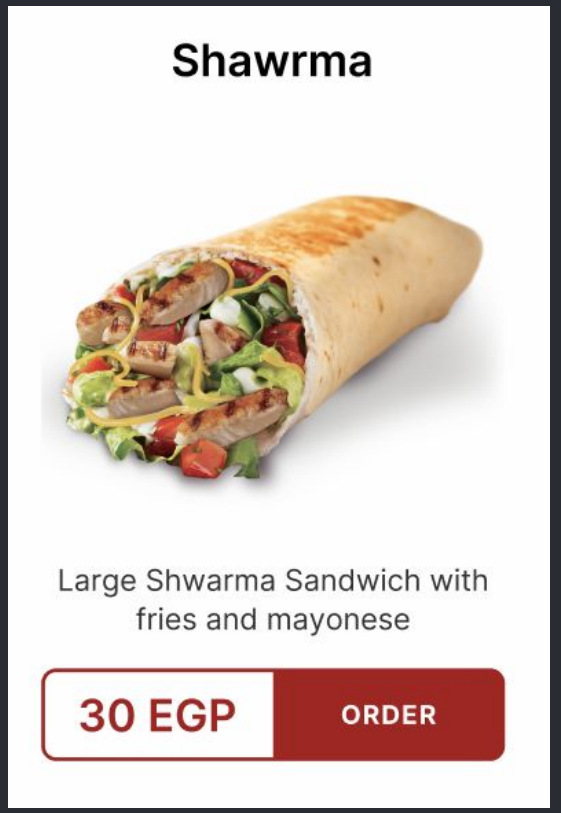


App.jsx

```jsx
import React from "react";
function App() {

  return (<h1>Hello From app</h1>
                        //Component 1
                        //Component 2
                        //Component 3
  );

}
export default App;

```

We Will make

ProductCard



App.jsx

```jsx
import React from "react";
import ProductCard from "../components/ProductCard";
function App() {
    return (
        <ProductCard/>
    );
}

export default App;
```

ProductCard.jsx

```jsx
export default function ProductCard() {
    return (
     <div className="max-w-[312.5px] px-8 py-4 bg-White flex flex-col items-center gap-4">
        <div className="flex flex-col items-center">
            <h2 className="h2 text-Headings">Shawrma</h2>
            <img src="https://i.imgur.com/6BgATBs.png"  className="w-[250px] h-fit"/>
            <p className="p text-Body text-center">
                Large Shwarma Sandwich with fries and mayonese </p>
        </div>
       <div className="flex items-center outline outline-2 outline-RedPrimary rounded-lg w-full
                overflow-clip ">
            <h2 className="text-RedPrimary h2 flex-grow text-center">30 EGP</h2>
            <button className="p-4 text-White bg-RedPrimary flex-grow flex justify-center ">
                <p className="outlinebody ">ORDER</p>
            </button>
       </div>
   </div>
); }
```

localhost:5173

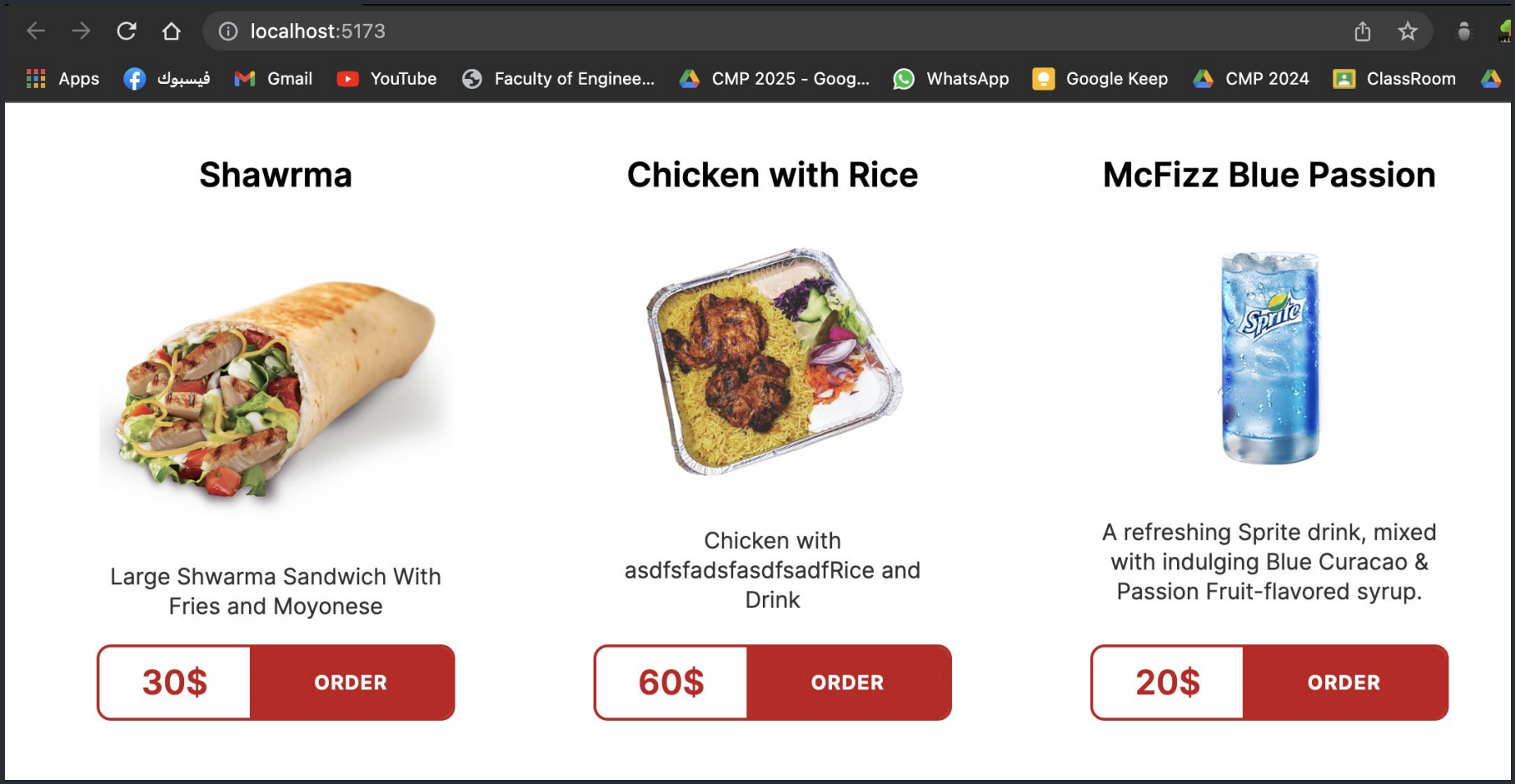
Apps فيسبوك Gmail YouTube

Shawrma

Large Shwarma Sandwich with fries and mayonese

30 EGP ORDER

Now We Want to make the menu page



Props

How to use props

```
<ProductCard
    ProductName="Shawrma"
    ProductImage="https://i.imgur.com/6BgATBs.png "
    ProductDescription="Large Shwarma Sandwich With Fries and Moyonese"
    ProductPrice="30$"
/>
```

ProductCard.jsx

```jsx
import React from "react";
    function ProductCard(props) {
     return (
    <div className="max-w-[312.5px] px-8 py-4 bg-White flex flex-col items-center gap-4">
          <div className="flex flex-col items-center">
           <h2 className="h2 text-Headings">{props.ProductName}</h2>
           <img src={props.ProductImage} alt="" className="w-[250px] h-fit" />
           <p className="p text-Body text-center">{props.ProductDescription}</p>
         </div>
        <div className="flex items-center outline outline-2 outline-RedPrimary rounded-lg w-full overflow-clip ">
          <h2 className="text-RedPrimary h2 flex-grow text-center">{props.ProductPrice} </h2>
          <button className="p-4 text-White bg-RedPrimary flex-grow flex justify-center ">
          <p className="outlinebody ">ORDER</p>
         </button>
       </div>
  </div>
);
}
export default ProductCard;
```

What is the props(**JS Object)**

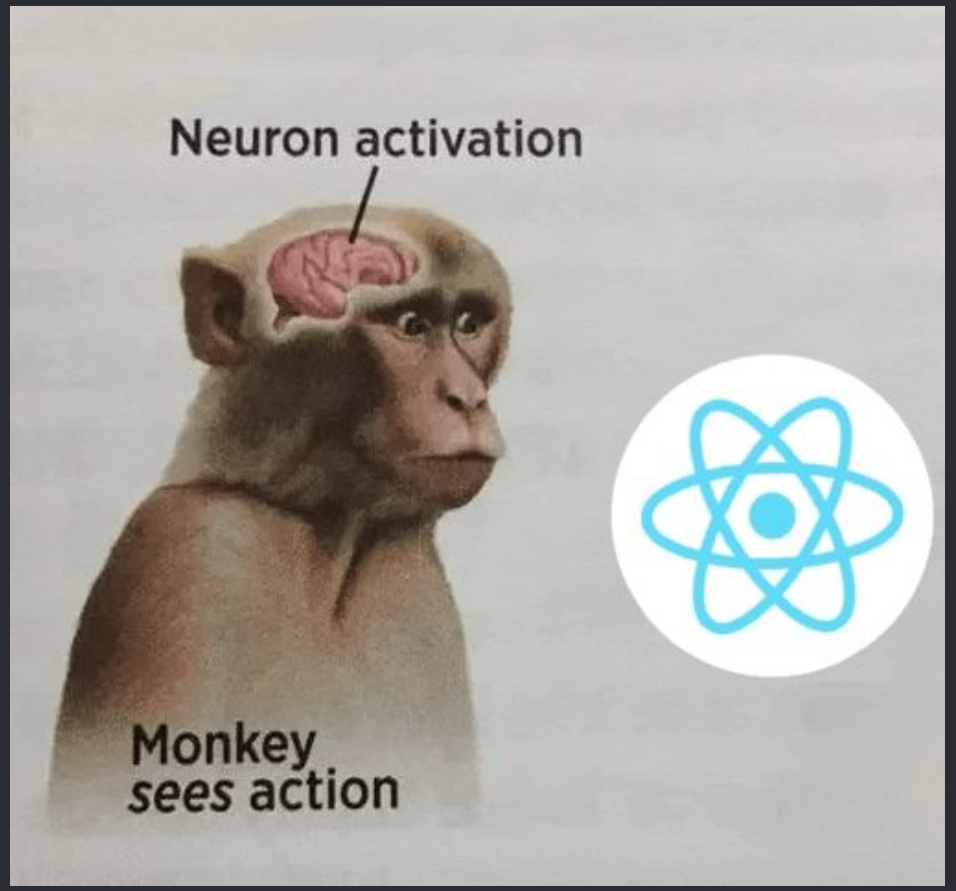
```
const props =
{
ProductName: "Shawrma",
ProductImage: "https://i.imgur.com/6BgATBs.png ",
ProductDescription: "Large Shwarma Sandwich With Fries and Moyonese",
ProductPrice: "30$",
}

```

State

Intro

React's state and props are fundamental concepts for building dynamic, interactive user interfaces.

- State refers to the internal data that a component can hold and update
- Props are passed down to a component from its parent component.
- State and props enable developers to create reusable, modular components that can be easily customized and updated.
- By using state and props, developers can build complex applications with ease while making their code more maintainable and scalable.



useStat

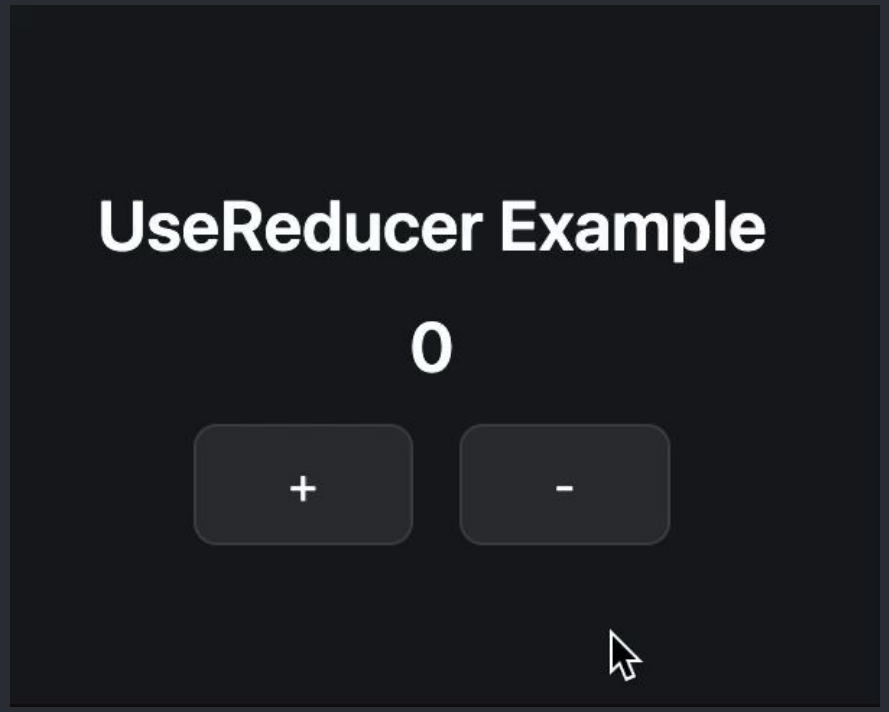
The useState hook is a built-in React hook that allows for state management in functional components. It accepts an initial state value and returns an array with the current state value and a function to update the state. This allows the component to re-render when the state is updated, changing the view of the user interface.

UseState.tsx before styling

```tsx
export const UseState = () => {
  const [number, setNumber] = useState(0);
  return (
    <div>
      <h2>{number}</h2>
      <button onClick={() => setNumber(number + 1)}> + </button>
      <button onClick={() => setNumber(number - 1)}> - </button>
    </div>
  );
};


```

useReducer

The `useReducer` hook is another built-in React hook that allows for more complex state management. It works similarly to the `useState` hook, but uses a reducer function to update the state based on dispatched actions. The `useReducer` hook is useful when the state has complex logic, and when the state update depends on the previous state.



UseReducer.tsx

```tsx
const InitialState = {
    number: 0,
  };

  const reducer = (state, action) => {
    switch (action.type) {
      case "Increment":
        return { number: state.number + 1 };
      case "decrement":
        return { number: state.number - 1 };
      default:
        throw new Error("Invalid Action Type");
    }
  };

  const UseReducer = () => {
    const [state, dispatch] = useReducer(reducer, InitialState);
    return (
      <div>
        <h2>{state.number}</h2>
        <div>
          <button onClick={() => dispatch({type:"Increment"})}>+</button>
          <button onClick={() => dispatch({type: "decrement"})}>-</button>
        </div>
      </div>
    );
  };
```

Shopping Cart

When building a shopping cart in React, the useState hook can become complex and difficult to maintain due to the many possible actions that can modify the cart's state. In this scenario, the useReducer hook is a better choice because it allows for more efficient and less boilerplate code. With useReducer, a single reducer function can handle all the different actions that modify the state of the cart, and each action can be represented by an object with a type property and additional data needed to perform the action.

The code of shopping cart in this link that describe everything about states with examples.
https://react-presentation-five.vercel.app/#/state

useMemo

The useMemo hook is used for optimizing performance by memoizing expensive computations. It accepts a function and an array of dependencies and only re-computes the result when the dependencies change. The useMemo hook is useful when the computation is expensive and has to be called several times.

UseMemo.tsx

```tsx
export const UseMemo = () => {
  const [number, setNumber] = useState(100);

  const data = Array.from({ length: 40 },
  () =>Math.floor(Math.random() * 40));

  const processedData = useMemo(() => {
    return data.sort((a, b) => a - b);
  }, [data]);

  return (
     <div>
       <div>
         <h2>Some Random numbers</h2>
         <ul>
           {processedData.map((item, index) => (
             <li key={index}>{item}</li>
           ))}
         </ul>
       </div>
       <div>
         <h2>Another State</h2>
         <h2>{number}</h2>
         <button onClick={() => setNumber(number + 1)}>+</button>
       </div>
     </div>
   );
};
```

Some Random numbers

2 4 5 5 6 7 8 8 9 10 11 11 11 12 12 16 16 16 17 17 17 18 19 22 22 22 23 24 26 26 27 27 28 28 30 30 30 30 34 36 38 42 42 42 43 43 46

47 49 50 51 53 53 53 53 55 56 56 56 57 57 58 58 59 59 60 60 62 62 64 66 68 69 71 72 73 75 76 78 79

Another State

100

useCallback

The useCallback hook is also used for performance optimization by memoizing functions. It accepts a function and an array of dependencies and only re-creates the function when the dependencies change. This is useful when the function needs to be passed down to child components, and the child components should only re-render when the function is updated.

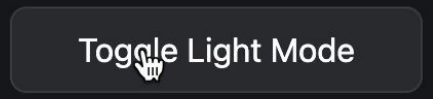
UseCallback Syntax

```
useCallback( () => { /* function to be memoized */ } , [/* dependcies */])

```

useContext

useContext

The useContext hook is used for accessing context values in nested components. It accepts a context object created by the createContext function and returns the current value of the context. Context is a way of passing down data through the component tree without having to pass props manually at every level.
When a component calls `useContext(ContextObject)`, React will look up the nearest matching `ContextObject.Provider` in the component tree hierarchy and return its value prop. If there is no matching provider, the default value passed to createContext will be returned instead.



App.tsx (Creating Context)

```
const DarkModeContext = createContext({
  darkMode: false,
  toggleDarkMode: () => undefined,
});

const App = () => {
  const [darkMode, setDarkMode] = useState(false);

  const toggleDarkMode = (dark) => {
    setDarkMode(dark);
    localStorage.setItem("theme", dark ? "dark" : "light");
    if (dark)
document.documentElement.classList.add("dark");
    else document.documentElement.classList.remove("dark");
  };

   useEffect(() => {
    if (localStorage.getItem("theme") === "dark")
 toggleDarkMode(true);
    else if (localStorage.getItem("theme") === "light")
 toggleDarkMode(false);
    else {
      localStorage.setItem("theme", "dark");
      toggleDarkMode(true);
    }
  }, []);
```

StateAndProps.tsx (using Context)

```
import { DarkModeContext } from "../App";
import { useContext } from "react";
...
const { darkMode, toggleDarkMode } =
useContext(DarkModeContext);
```

App.tsx (Creating Context)

```
    else {
      localStorage.setItem("theme", "dark");
      toggleDarkMode(true);
    }
  }, []);

  return (
    <div>
      <DarkModeContext.Provider value={{ darkMode,
toggleDarkMode }}>
        <NavBar />
        <StateAndProps />
      </DarkModeContext.Provider>
    </div>
  );
};
```

StateAndProps.tsx (using Context)

```
import { DarkModeContext } from "../App";
import { useContext } from "react";
...
const { darkMode, toggleDarkMode } =
useContext(DarkModeContext);
```

Data Fetching

Data fetching is a common use case in React applications. There are various libraries and methods for fetching data, including the `fetch` API, `axios`, and the `useEffect` hook. Data fetching is usually done in the `useEffect` hook, which is a built-in React hook for handling side effects. The `useEffect` hook allows for running code after the component is mounted, and for cleaning up after the component is unmounted.

Fetch Api

- Built-in to modern browsers, no additional dependencies needed
- Lightweight and easy to use
- Supports both simple and complex requests
- Returns a Proxmise which is easily integrated with async/await

DataFetching.tsx

```tsx
export const DataFetching = () => {
  const [quote, setQuote] = useState({
    author: "",
    id: 0,
    quote: "",
  });
  const [isLoading, setIsLoading] = useState(true);

  const fetchQuote = async () => {
    setIsLoading(true);
    const quoteRes = await
fetch("https://dummyjson.com/quotes/random");
    const quote = await quoteRes.json();
    setIsLoading(false);
    setQuote(quote);
  };

  useEffect(() => {
    fetchQuote();
  }, []);

```

Fetch Api

- Built-in to modern browsers, no additional dependencies needed
- Lightweight and easy to use
- Supports both simple and complex requests
- Returns a Proxmise which is easily integrated with async/await

DataFetching.tsx

```tsx
  return (
    <div>
      {isLoading ? (
        <AiOutlineLoading />
      ) : (
        <blockquote>
          <div>
            <FaQuoteLeft />
          </div>
          <div>
            <p>{quote.quote}</p>
            <p>- {quote.author}</p>
          </div>
        </blockquote>
      )}
      <Btn onClick={() => fetchQuote()}>
        Fetch Another One
      </Btn>
    </div>
  );
};
```

I attribute my success to this: I never gave or took any excuse.

- Florence Nightingale

Fetch Another One

Creating Custom Hooks

Custom hooks are reusable functions that encapsulate common logic in a component. They are a way to share stateful logic between components without having to use higher-order components, render props, or other patterns. Custom hooks allow you to abstract complex logic into a reusable and testable module that can be shared across multiple components.

- Reusability: Custom hooks are reusable functions that can be shared between components, reducing code duplication.
- Abstraction: Custom hooks abstract away complex logic, making it easier to reason about and test.
- Encapsulation: Custom hooks encapsulate stateful logic, making it easier to manage and debug.

UseFetch.tsx

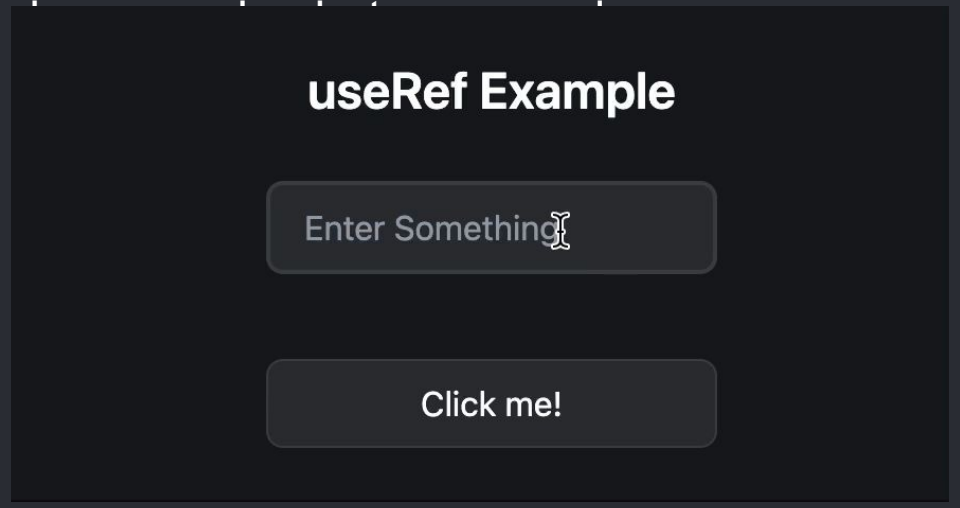
```tsx
import { useState, useEffect, useCallback } from
"react";
export const useFetch = (url) => {
  const [data, setData] = useState();
  const [error, setError] = useState();
  const [isLoading, setIsLoading] = useState(false);
  const fetchData = useCallback(async () => {
    setIsLoading(true);
    try {
      const response = await fetch(url);
      const jsonData = await response.json();
      setData(jsonData);
      setIsLoading(false);
    }
    catch (error) {
      setError(error);
    }
  }, [url]);
  useEffect(() => {
    fetchData();
  }, [fetchData]);
  return {
    data: [data, error, isLoading],
    refetch: fetchData,
  };
};
```

DataFetching.tsx

```tsx
/* const [quote, setQuote] = useState<quote>({
author: "", id: 0, quote: "",});
const [isLoading, setIsLoading] = useState(true);
const fetchQuote = async () => {
  setIsLoading(true);
  const quoteRes = await
  fetch("https://dummyjson.com/quotes/random");
  const quote = (await quoteRes.json());
  setIsLoading(false);
  setQuote(quote);
}; */
const {
  data: [quote, error, isLoading],
  refetch: fetchQuote,
} = useFetch("https://dummyjson.com/quotes/random");
```

useRef

The useRef hook is used for accessing DOM elements or storing mutable values between renders. It returns a mutable ref object with a current property that can be updated. This is useful when the component needs to access a DOM element or when the component needs to



UseRef.tsx

```tsx
export const UseRefExample = () => {
  const inputRef = useRef(null);
  const [message, setMessage] = useState("");

  const handleClick = (e: FormEvent) => {
    e.preventDefault();
    const inputValue = inputRef.current?.value as
string;
    setMessage("User Entered : " + inputValue);
  };

  return (
    <form>
      <input
        type="text"
        ref={inputRef}
      />
      <p>{message}</p>
      <Btn rectangle onClick={handleClick}>
        Click me!
      </Btn>
    </form>
  );
};
```

React Router

React Router is a popular library for adding routes to React applications. It allows for declarative routing, meaning that routes are defined in a structured and readable way. React Router also provides various components for handling common routing tasks, such as rendering different components based on the current route or passing route parameters.

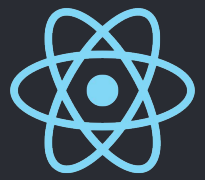
It works by changing the component shown on the screen with component provided in the routes

App.tsx

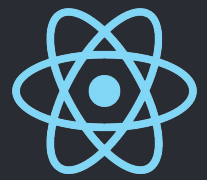
```tsx
import {BrowserRouter as Router, Route, Routes} from "react-router-dom";

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/stateprops" element={<StateAndProps />} />
        <Route path="/routing" element={<Routing />} />
      </Routes>
    </Router>
  )
}
```

Adding Links

Links can be added using built in Link component



useEffect

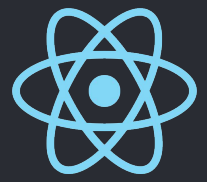


useEffect

- **useEffect** is a hook that allows you to perform side effects in functional components.

- useEffect returns a cleanup function that can be used to clean up after the effect.

```
useEffect(
effectFunction,dependencyArray);
```



Recap on Javascript Arrow Functions

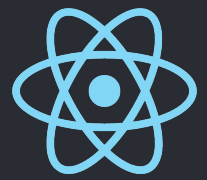
- Arrow functions are a concise way to write functions in JavaScript. They use the "=>" syntax and are often used in modern JavaScript frameworks like React.

Normal Function

```
function add(a, b) {
    return a + b;
}
```

Arrow Function

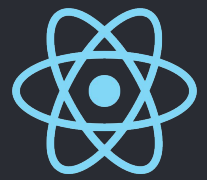
```
const add = (a, b) => a + b;

const add = (a, b) =>  {
    return a + b;
}
```



useEffect Syntax

useEffectsyntax

```
function MyComponent() {

    useEffect(() => {
        // effect function

        return () => {
            // cleanup function
        };

    }, [...dependencies]);

    // rest of component code

}
```

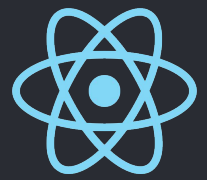


useEffect

1. **Effect with dependencies It gets called every time any of the dependencies changes.**

3-useEffect

```
1  const users = [
2      {
3          name:'Ahmed',
4          age:20,
5      },
6      {
7          name:'Hamdy',
8          age:20
9      }
10 ];
11
12 function UserInfo({ userId }) {
13    const [user, setUser] = useState(null);
14
```

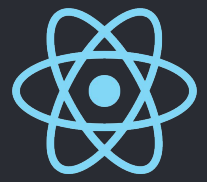


useEffect

1. **Effect with dependencies It gets called every time any of the dependencies changes.**

3-useEffect

```
12 function UserInfo({ userId }) {
13    const [user, setUser] = useState(null);
14
15     useEffect(() => {
16         setUser(users[userId]);
17     }, [userId]);
18
19     if (!user) {
20       return <div>Loading...</div>;
21     }
22
23     return <div>{user.name}</div>;
24   }
25
```

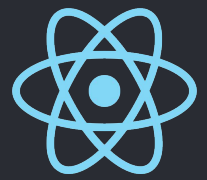


useEffect

2 . **Effect with empty dependencies It gets called one time when the component is mounted.**

2-useEffect

```
function MyComponent() {
    useEffect(() => {
        console.log("Component mounted");
        return () => console.log("Component Unmounted");
    }, []);
    // rest of component code
}

```



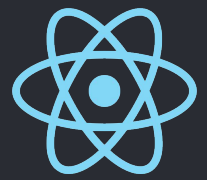
useEffect

3. Effect with no dependencies It gets called every time the component changes.

1-useEffect

```
1 function MyComponent() {
2     useEffect(() => {
3         console.log("Component updated");
4     });
5 }
6
```

- **Be careful while using it, as it might lead to infinity loop if changing any state inside it .**

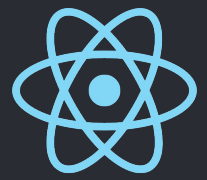


useEffect

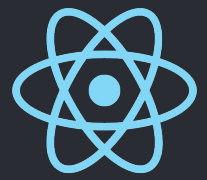
4 . Effect with clean-up function

3-useEffect

```
1  function MyComponent() {
2    useEffect(() => {
3      const handleKeyPress = (event) => {
4        if (event.code === 'F1') {
5          console.log('F1 pressed!');
6        }
7      };
8
9      window.addEventListener('keydown', handleKeyPress);
10
11     return () => {
12       window.removeEventListener('keydown', handleKeyPress);
13     };
14   }, []);
15
16   return (<h1>Press F1 to log to console</h1>);
17 }
```



Class-based vs Functional Components



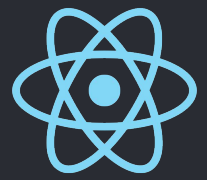
Class-based vs Functional Components

Class-based

```
class App extends React.Component {
    constructor(props){
        super(props);
    }

    render() {
     return (<h1>Hello From app</h1>);
    }
}

export default App;
```

Function-based

```
import React from "react";
function App() {
  return (<h1>Hello From app</h1>);
}
export default App;
```



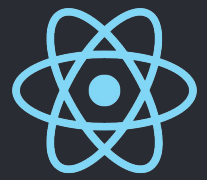
State Change

Class-based

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
    this.handleIncrement =
this.handleIncrement.bind(this);
  }

  handleIncrement() {
    this.setState({
     ...this.state, count: this.state.count
      + 1 }
    );
  }

}
```

Function-based

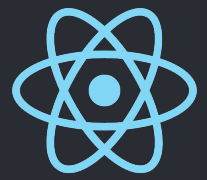
```
function Counter() {
 const [count, setCount] = useState(0);

  function handleIncrement() {
    setCount(count + 1);
  }
}

```



Class-based vs Functional Components

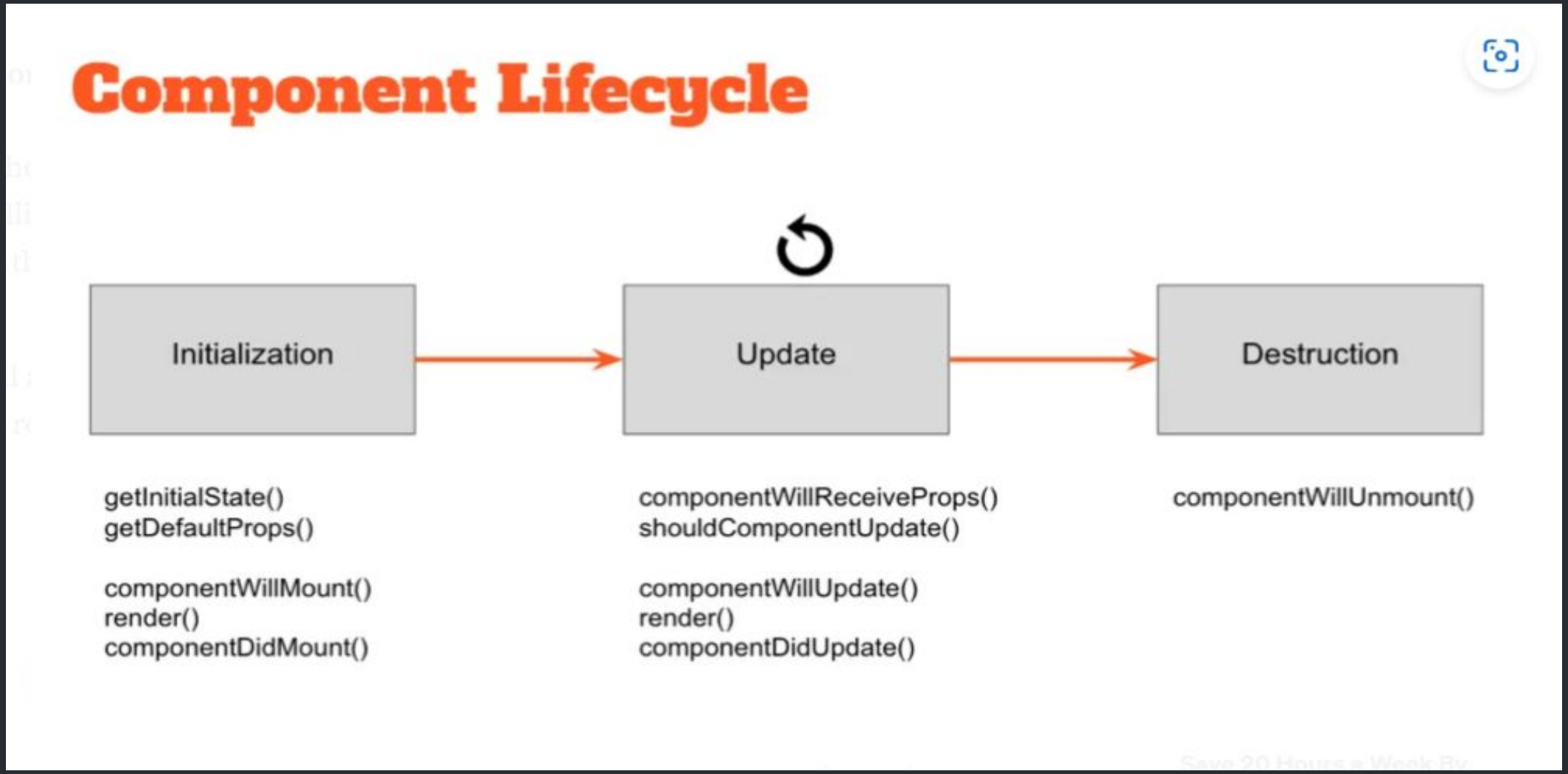
Why to use Functional over Class-based?

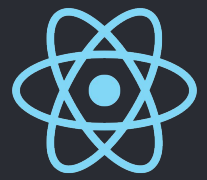
–Simplicity
- Performance
- Hooks
- Better integration with other libraries (Redux, React-Native, …etc)



Component Lifecycle

Component Lifecycle





Lifecycle

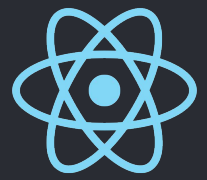
1. **Constructor**

Class-based

```
class MyComponent extends React.Component {
    constructor(props) {
      super(props);
      this.state = { count: 0 };
      // other initializations
    }
}
```

Function-based

```
function MyComponent(props) {
    const [count, setCount] = useState(0);
}
```



Lifecycle

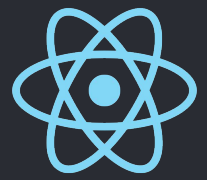
2. **Render**

Class-based

```
class MyComponent extends React.Component {
    render() {
      return (
        <div>
          <h1>Count:{this.state.count}</h1>
        </div>
      );
    }
}
```

Function-based

```
function MyComponent(props) {
  return (
    <div>
      <h1>Count: {count}</h1>
    </div>
  );
}
```



Lifecycle

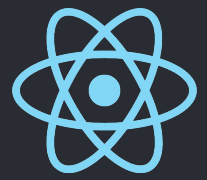
3. Component Mounted

Class-based

```
class MyComponent extends React.Component {
    componentDidMount() {
      fetch("https://api.example.com/data")
        .then(response => response.json())
        .then(
          data => this.setState({ data })
        );
    }
}
```

Function-based

```
function MyComponent() {
   const [data, setData] = useState(null);
    useEffect(() => {
      fetch("https://api.example.com/data")
        .then(response => response.json())
        .then(data => setData(data));
    }, []);

}

```



Lifecycle

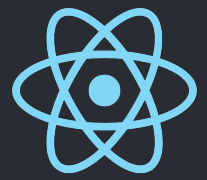
4. Component Did Update

Class-based

```
1 class MyComponent extends React.Component {
2   componentDidUpdate(prevProps, prevState){
3     if
4      (this.state.count !== prevState.count)
5 {
6           console.log("Count has changed!");
7      }
8   }
9 }
```

Function-based

```
1 function MyComponent() {
2     const [count, setCount] = useState(0);
3     useEffect(() => {
4         console.log("Count has changed!");
5     }, [count]);
6 }
```



Lifecycle

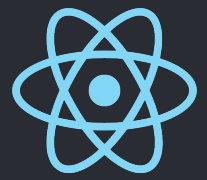
5. Component Will Unmount

Class-based

```
1 class MyComponent extends React.Component {
2     componentWillUnmount() {
3       window.removeEventListener(
4         "resize", this.handleResize);
5       clearInterval(this.timerID);
6     }
7 }
8
9
```

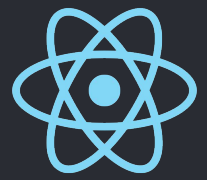
Function-based

```
1 function MyComponent() {
2     useEffect(() => {
3         // add events & timers
4         return () => {
5             window.removeEventListener(
6                 "resize", handleResize);
7             clearInterval(timerID);
8         };
9     }, []);
10}
11
```



React Native

Learn once, write anywhere!



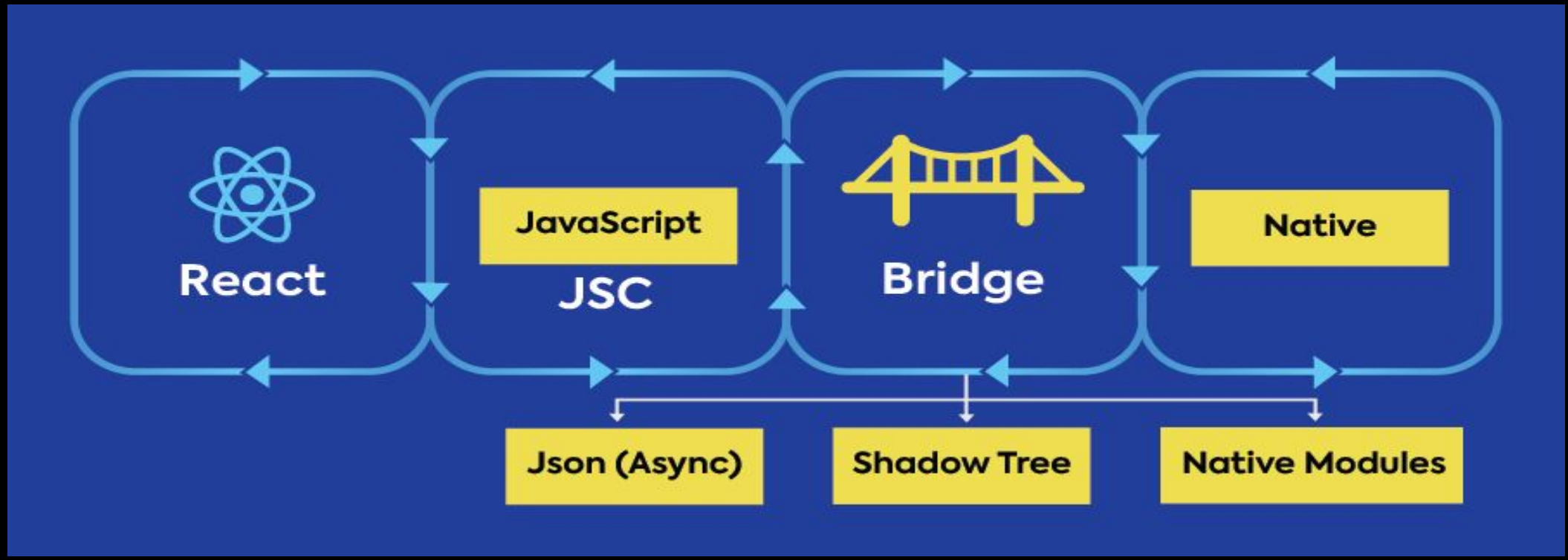
What is React Native?

- Open-source JavaScript Mobile Framework created by Facebook
- Comes out of React
- Written in JavaScript, rendered with native code
- Can be used in existing apps or to create a whole new app
- Some apps like Facebook, Instagram, and Discord uses React Native.

How does React Native works?

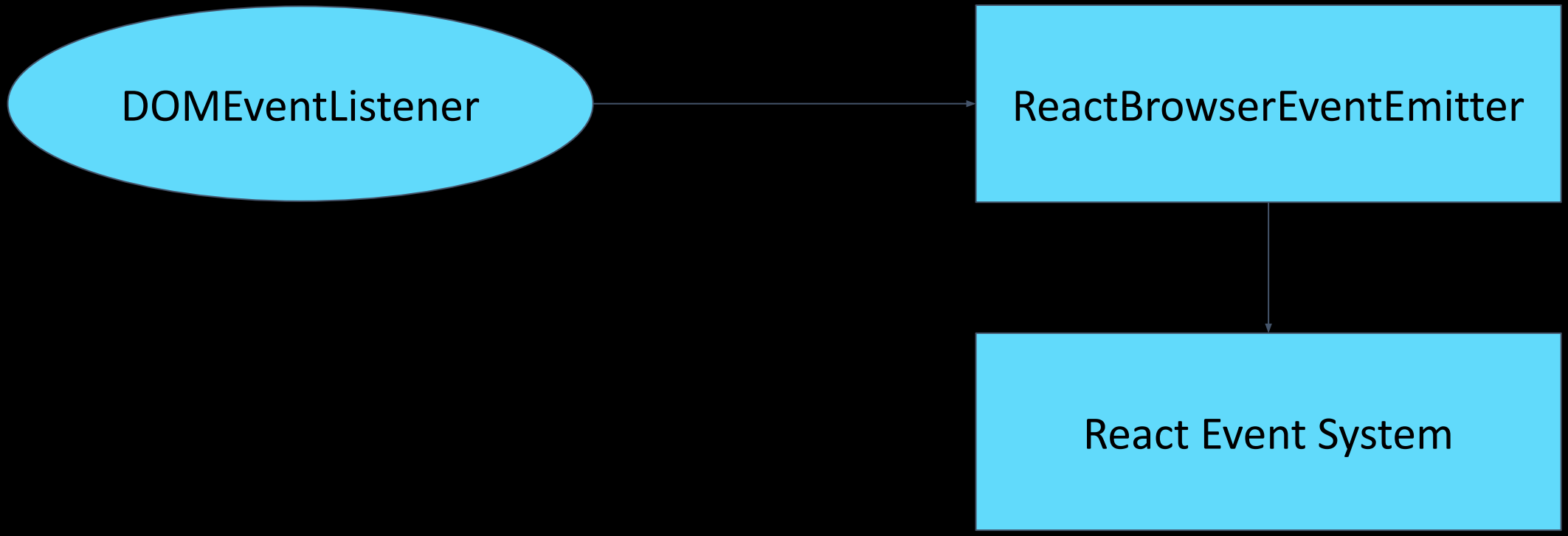
React Native has three Threads

- **UI Thread:-** This is the main thread of an application that has access the UI of the app
- **Shadow Thread:-** This is the background thread that uses React library for the calculation of the layout of the application
- **JavaScript Thread:-** The JavaScript thread that executes the React (JavaScript code) code

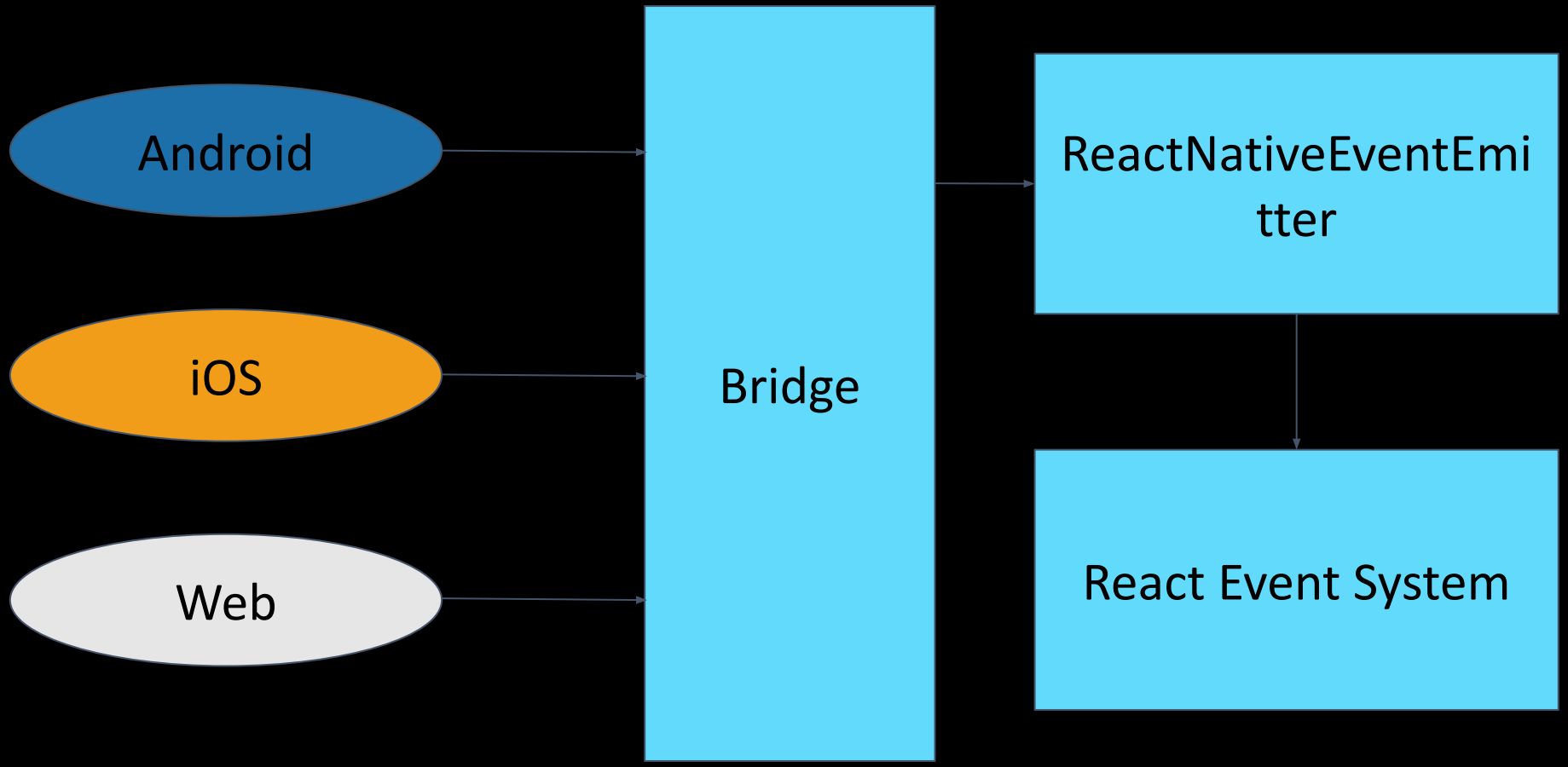


- first of all, a layout is created using the codes in the JavaScript thread. This layout calculation is then offloaded to the shadow thread which constructs a tree of this layout using a layout engine called Yoga.
- The communication between the two threads happens through a React Native bridge which serializes and transfers it to the UI (main) thread after receiving the rendered markup from Yoga.
- The main thread then renders the UI after deserialization and completes the mapping from the browser to React Native.

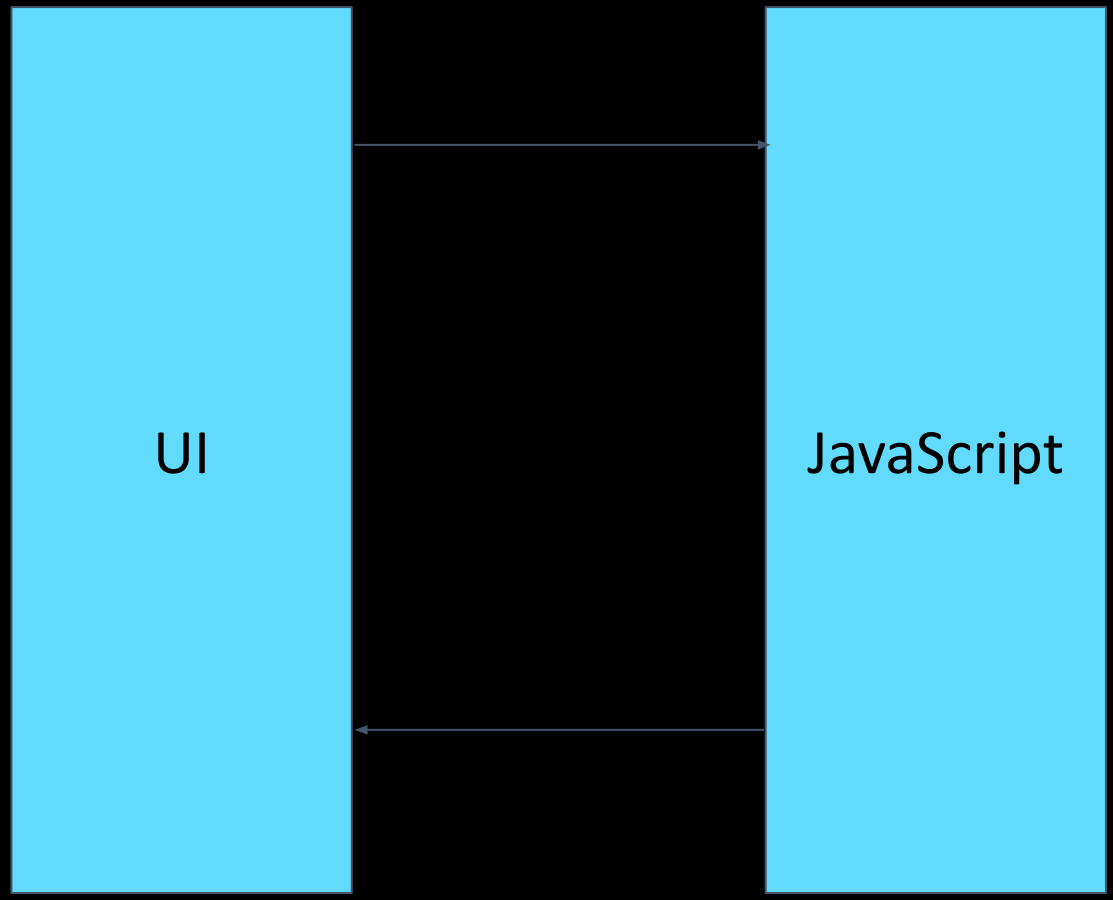
Events in React



Events in React Native



Events in React Native



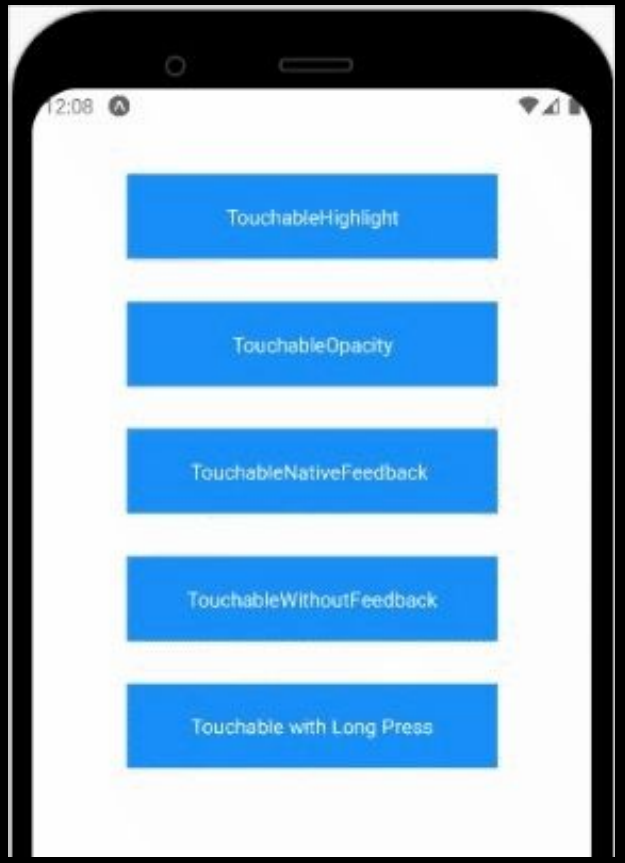
Events in React Native

- In React every web component has an interaction
- In React native, however, this is no the case
- All events are reduced to “Touches”
- “Touchable” component may return different events according to how they were interacted with
- Not limited to just Touchables buttons (Google maps embedded in a page can be pinched to zoom/ photos can be press and held to bring up a menu)

Touches and Touchables

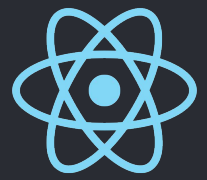
```
export default class Touchables extends Component {
  _onPressButton() {
    Alert.alert('You tapped the button!');
  }

  _onLongPressButton() {
    Alert.alert('You long-pressed the button!');
  }

  render() {
    return (
      <View style={styles.container}>
        <TouchableHighlight onPress={this._onPressButton} underlayColor="white">
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableHighlight</Text>
          </View>
        </TouchableHighlight>
        <TouchableOpacity onPress={this._onPressButton}>
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableOpacity</Text>
          </View>
        </TouchableOpacity>
        <TouchableNativeFeedback
          onPress={this._onPressButton}
          background={
            Platform.OS === 'android'
              ? TouchableNativeFeedback.SelectableBackground()
              : undefined
          }>
          <View style={styles.button}>
            <Text style={styles.buttonText}>
              TouchableNativeFeedback{' '}
              {Platform.OS !== 'android' ? '(Android only)' : ''}
            </Text>
          </View>
        </TouchableNativeFeedback>
        <TouchableNativeFeedback
          onPress={this._onPressButton}
          background={
            Platform.OS === 'android'
              ? TouchableNativeFeedback.SelectableBackground()
              : undefined
          }>
          <View style={styles.button}>
            <Text style={styles.buttonText}>
              TouchableNativeFeedback{' '}
              {Platform.OS !== 'android' ? '(Android only)' : ''}
            </Text>
          </View>
        </TouchableNativeFeedback>
        <TouchableWithoutFeedback onPress={this._onPressButton}>
          <View style={styles.button}>
            <Text style={styles.buttonText}>TouchableWithoutFeedback</Text>
          </View>
        </TouchableWithoutFeedback>
        <TouchableHighlight
          onPress={this._onPressButton}
          onLongPress={this._onLongPressButton}
          underlayColor="white">
          <View style={styles.button}>
            <Text style={styles.buttonText}>Touchable with Long Press</Text>
          </View>
        </TouchableHighlight>
      </View>
    );
  }
}
```

Touches and Touchables



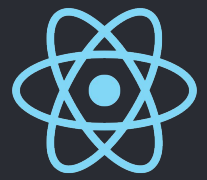
Touches and Touchables

- For more interactivity use Gesture Responders to be able to determine what the user intends to do
- Other touchables include ScrollView to list of items



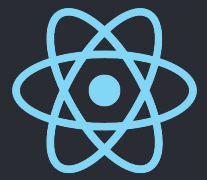
Create a React Native App

1. Set up the environment
2. Create your project
3. Test your project



Setting up the environment

Expo Go CLI	React Native CLI
- Set of tools and services built Native apps - Provides simplified app development workflow - Has a lot of utility features to do common tasks - Less flexible - Much simples	- Pure React Native app - Requires more effort and configurations to get up running - Consists of the very essentials only - More flexible - More Complex



Create a project

To install Expo CLI, just write the following command in your terminal after installing node.js

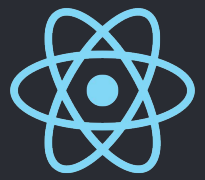
```
npm install --global expo-cli
```

Then to create a project write the following commands to your terminal

```
npx create-expo-app MyFirstProject
cd MyFirstProject
npx expo start
```

You can test and preview your app using

- Your own phone
- An emulator, e.g., Android Studio



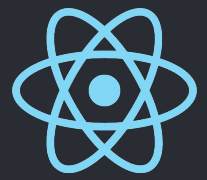
C:\WINDOWS\system32\cmd.

```
C:\Users\Owner\OneDrive\Desktop\trash-can\sandbox>npx expo start
Starting project at C:\Users\Owner\OneDrive\Desktop\trash-can\sandbox
Starting Metro Bundler
```

```
› Metro waiting on exp://192.168.1.3:19000
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Press a │ open Android
› Press w │ open web

› Press j │ open debugger
› Press r │ reload app
› Press m │ toggle menu

› Press ? │ show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
› Detected a change in babel.config.js. Restart the server to see the new results. You may need to clear the bundler cache with the --clear flag for
 your changes to take effect.
```

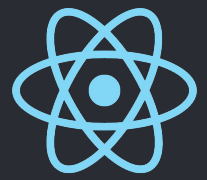
App.jsx

```jsx
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});

```



React Native Components

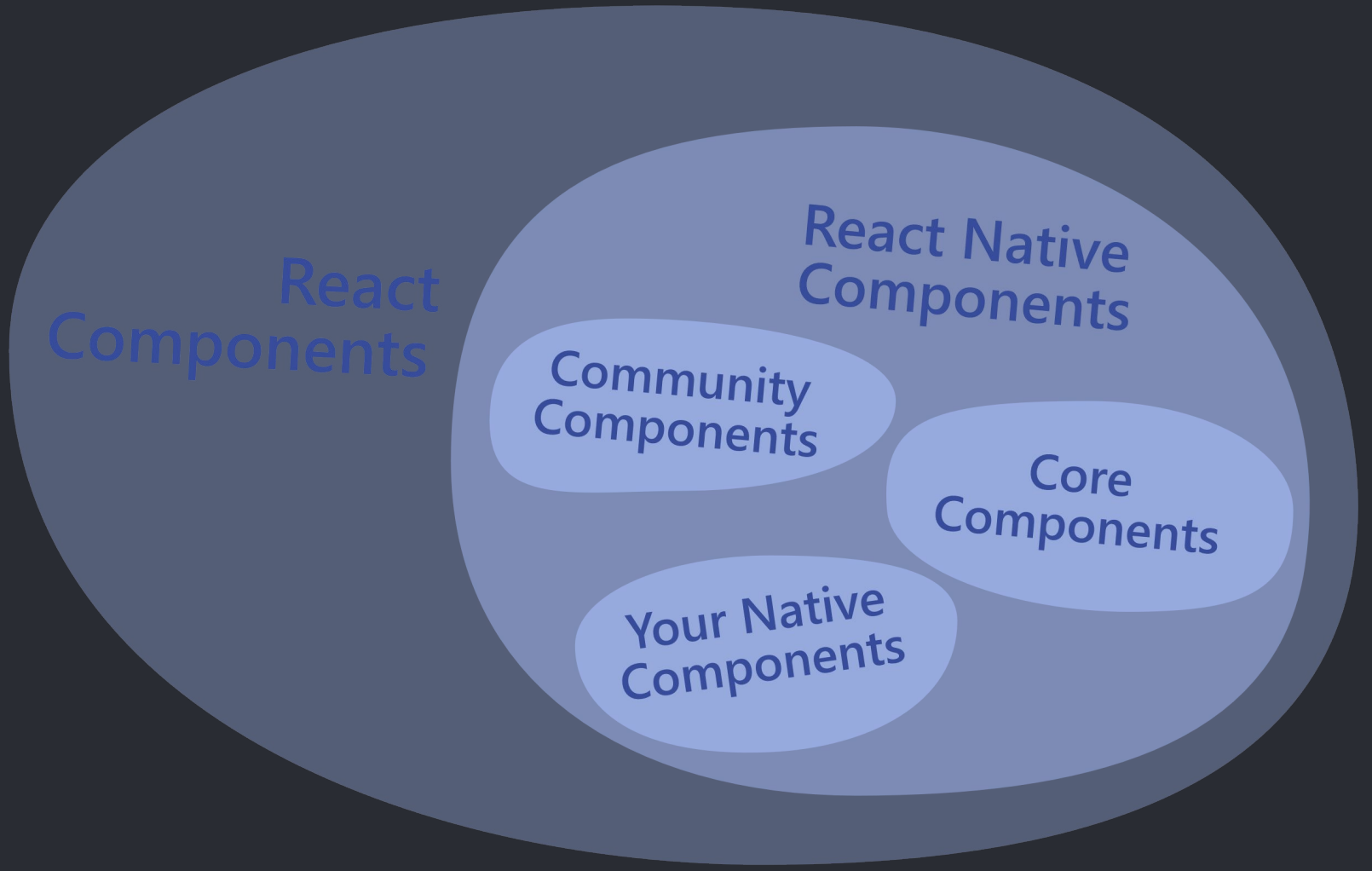
- In Android development, you write views in Kotlin or Java; in iOS development, you use Swift or Objective-C. But with React Native, you can invoke these views with JavaScript using React components
- At runtime, React Native creates the corresponding Android and iOS views for those components
- We call these components Native Components, because they’re backed by the same view as Android and iOS

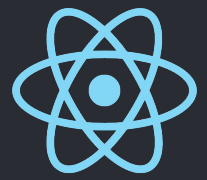


React Native Components

- Core Components: A set of essential, ready-to-use Native Components you can use to start building your app
- You can also build your own native components for Android and iOS to suit your app’s unique needs
- There’s also a thriving ecosystem of theses community-contributed components that are ready to use

React Native Components





React Native UI Component	Android View	iOS View	Web Analog	Description
<View>	<ViewGroup>	<UIView>	A non-scrolling	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
<Text>	<TextView>	<UITextView>	<p>	Displays, styles, and nests strings of text and even handles touch events
<Image>	<ImageView>	<UIImageView>	<img>	Displays different types of images
<ScrollView>	<ScrollView>	<UIScrollView>		A generic scrolling container that can contain multiple components and views
<TextInput>	<EditText>	<UITextField>	<input type=”text”>	Allow the user to enter text

sandbox.js

```
import React, { useState } from 'react';
import { StyleSheet, View, Text, Image, ScrollView, TextInput } from
'react-native';

const App = () => {
  const [name, setName] = useState('Hamdy');

  const [age, setAge] = useState('21');

  return (
    <View style={styles.container}>
      <Text style={styles.header}>This is the header!</Text>
      <ScrollView>
        <View>
          <Text>This is an image!</Text>
          <Image
            source={{
              uri: 'https://reactnative.dev/img/logo-og.png',
            }}
            style={styles.image}
```

sandbox.js

```
  const [age, setAge] = useState('21');

  return (
    <View style={styles.container}>
      <Text style={styles.header}>This is the header!</Text>
      <ScrollView>
        <View>
          <Text>This is an image!</Text>
          <Image
            source={{
              uri: 'https://reactnative.dev/img/logo-og.png',
            }}
            style={styles.image}
          />
        </View>
        <View>
          <Text>Enter your name:</Text>
          <TextInput
            style={styles.input}
            placeholder='e.g. Hamdy'
            onChangeText={(val) => setName(val)}
```

sandbox.js

```
        source={{
          uri: 'https://reactnative.dev/img/logo-og.png',
        }}
        style={styles.image}
      />
    </View>
    <View>
      <Text>Enter your name:</Text>
      <TextInput
        style={styles.input}
        placeholder='e.g. Hamdy'
        onChangeText={(val) => setName(val)}
      />
      <Text>Enter your age:</Text>
      <TextInput
        style={styles.input}
        placeholder='e.g. 21'
        keyboardType='numeric'
        onChangeText={(val) => setAge(val)}
      />
      <Text>name: {name}, age: {age}</Text>
```

sandbox.js

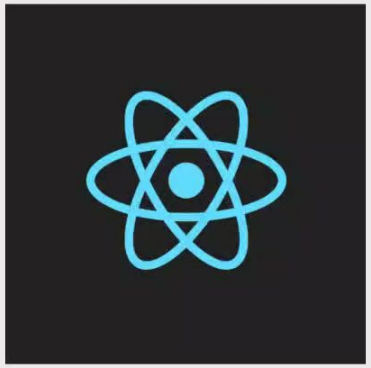
```
            <Text>name: {name}, age: {age}</Text>
          </View>
        </ScrollView>
      </View>
    );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 24,
    backgroundColor: '#eaeaea',
    alignItems: 'center',
    justifyContent: 'center',
  },
  header: {
    margin: 12,
    paddingVertical: 8,
    borderWidth: 3,
    borderColor: 'rgb(33, 35, 41)',
    borderRadius: 6,
```

```
    header: {
      margin: 12,
      paddingVertical: 8,
      borderWidth: 3,
      borderColor: 'rgb(33, 35, 41)',
      borderRadius: 6,
      backgroundColor: 'rgb(130, 215, 247)',
      fontWeight: 'bold',
      fontSize: 24,
      padding: 20,
    },
    image: {
      width: 200,
      height: 200,
      padding: 20,
      margin: 12,
    },
    input: {
      height: 40,
      padding: 8,
      margin: 10,
      width: 200
```

10:53

This is the header!

This is an image!



Enter your name:

e.g. Hamdy

Enter your age:

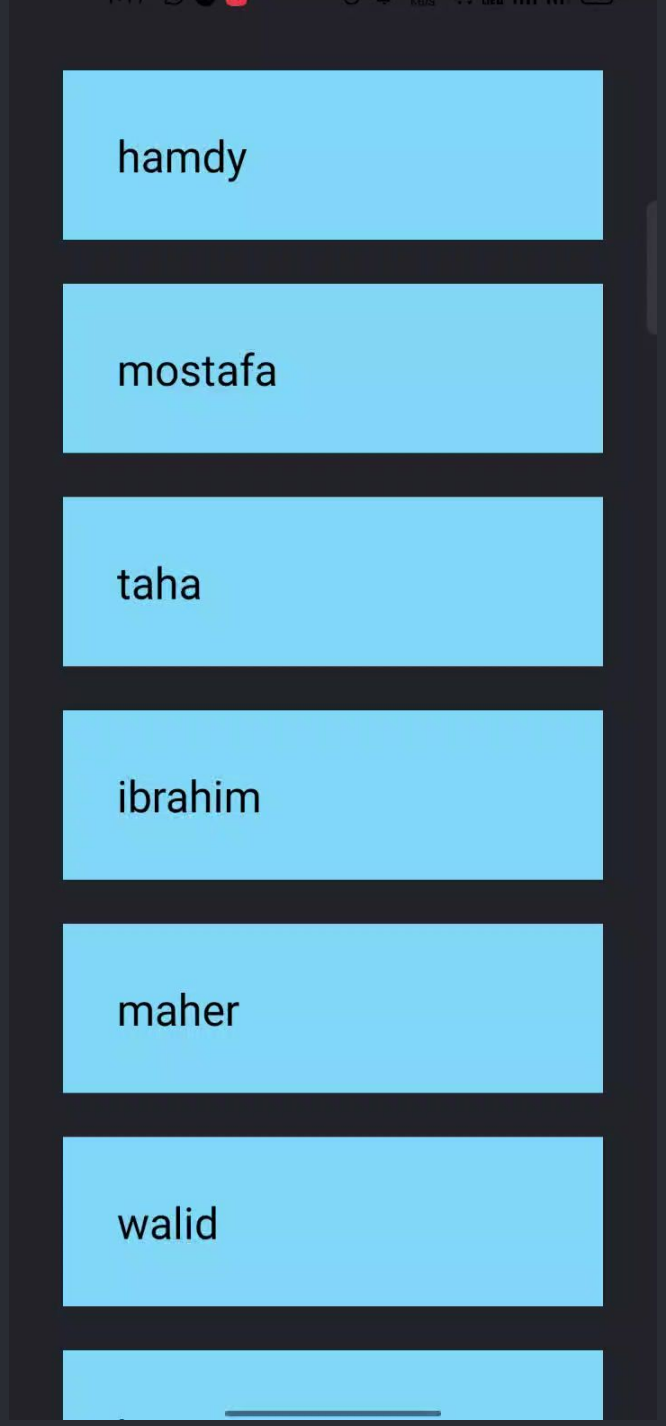
e.g. 21

name: Hamdy, age: 21

FlatList_Comp.js

```
import React, { useState } from 'react';
import { StyleSheet, Text, View, FlatList } from 'react-native';
export default function App() {
  const [people, setPeople] = useState([
    { name: 'hamdy', id: '1' },
    { name: 'taha', id: '2' },
    { name: 'ibrahim', id: '3' },
    { name: 'kore', id: '4' },
    { name: 'sayed', id: '5' },
    { name: 'maher', id: '6' },
    { name: 'walid', id: '7' },
  ]);
  return (
    <View style={styles.container}>
      <FlatList
        numColumns={2}
        keyExtractor={(item) => item.id}
        data={people}
        renderItem={({ item }) => (
          <Text style={styles.item}>{item.name}</Text>
        )}
```

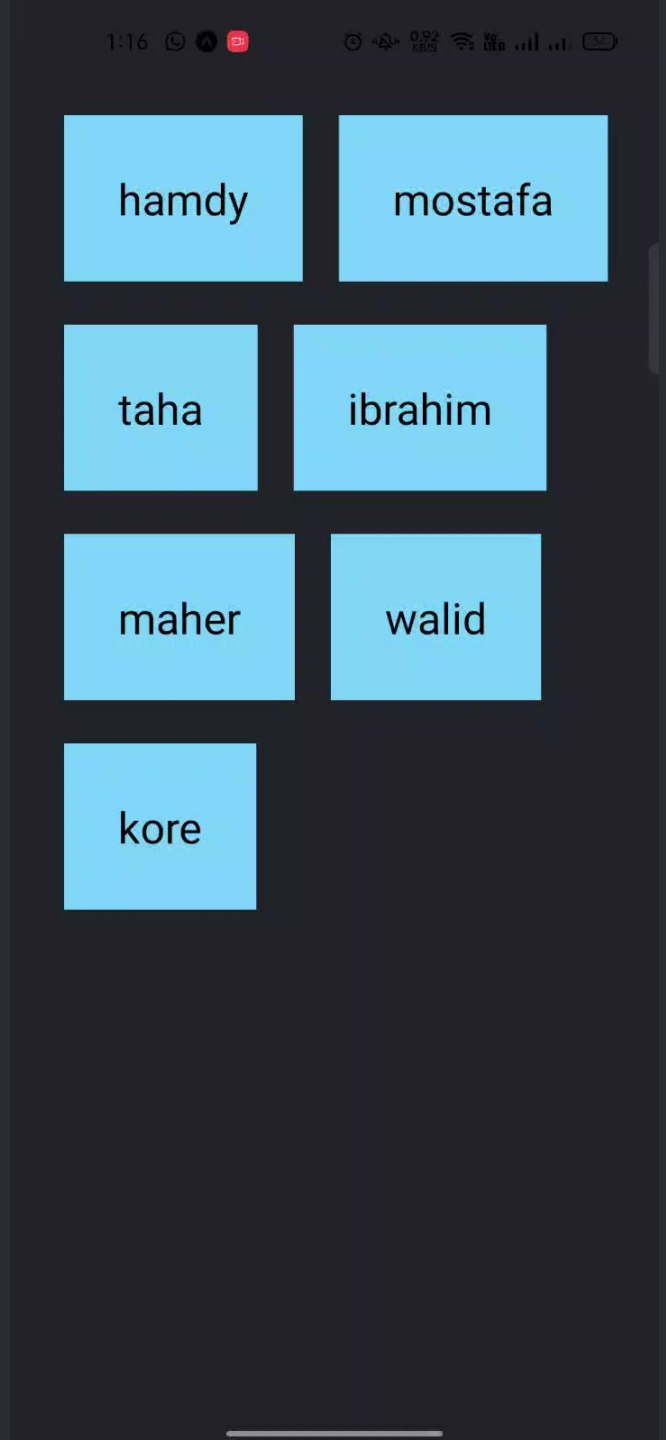
FlatList

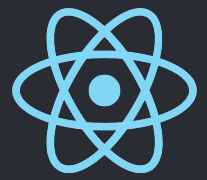


TouchableOpacity_Comp.js

```
  const pressHandler = (id) => {
    console.log(id);
    setPeople((prevPeople) => {
      return prevPeople.filter(person => person.id != id);
    });
  };

  return (
    <View style={styles.container}>
      <FlatList
        numColumns={2}
        keyExtractor={(item) => item.id}
        data={people}
        renderItem={({ item }) => (
          <TouchableOpacity onPress={() => pressHandler(item.id)}>
            <Text style={styles.item}>{item.name}</Text>
          </TouchableOpacity>
        )}
      />
    </View>
```

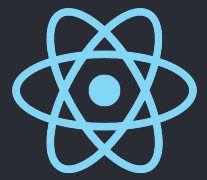
TouchableOpacity





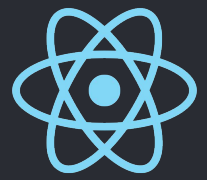
Platform Specific Code

- When building a cross-platform app, you'll want to re-use as much code as possible. Scenarios may arise where it makes sense for the code to be different, for example you may want to implement separate visual components for Android and iOS.
- React Native provides two ways to organize your code and separate it by platform:
  - Using the `platform module`
  - Using `platform-specific file extensions`
- Certain components may have properties that work on one platform only.



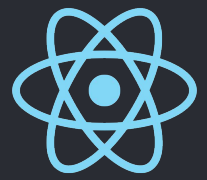
Platform module

```
1 import { Platform, StyleSheet } from 'react-native';
2
3 const styles = StyleSheet.create({
4   height: Platform.OS === 'ios' ? 200 : 100,
5 });
6
7
8
```



Platform module

```
import { Platform, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
  container: {
    flex: 1,
    ...Platform.select({
      ios: {
        backgroundColor: 'red',
      },
      android: {
        backgroundColor: 'green',
      },
      default: {
        // other platforms, web for example
        backgroundColor: 'blue',
      },
    }),
  },
});
```



Platform-specific extensions

```
BigButton.ios.js
BigButton.android.js

import BigButton from './BigButton';

```

Styling

Styling

Agenda:

- Inline Styling
- CSS Stylesheets
- CSS modules Styling

Inline Styling

main.js

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  const Header = () => {
5  return (
6  <>
7  <h1 ›Hello Style!‹/h1>
8  <p›Add a little style!‹/p>
9  </>
10 )
11 };
12 const root = ReactDOM. createRoot
13 (document. getelementById('root'));
14 root.render (‹Header />);
```

main.js

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  const Header = () => {
5  return (
6  <>
7  <h1 style={{color: "red"}} >Hello Style!</h1>
8  <p›Add a little style!‹/p>
9  </>
10 )
11
12 };
13 const root = ReactDOM. createRoot
14 (document. getelementById('root'));
15 root.render (<Header />);
```

main.js

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  const Header = () => {
5  return (
6  <>
7  <h1 style={{color: "red"}} >Hello Style!</h1>
8  <p›Add a little style!‹/p>
9  </>
10 )
11 };
12 const root = ReactDOM. createRoot
13 (document. getelementById('root')));
14 root.render (‹Header />);
```

To style an element with the inline style attribute, the value must be a JavaScript object:

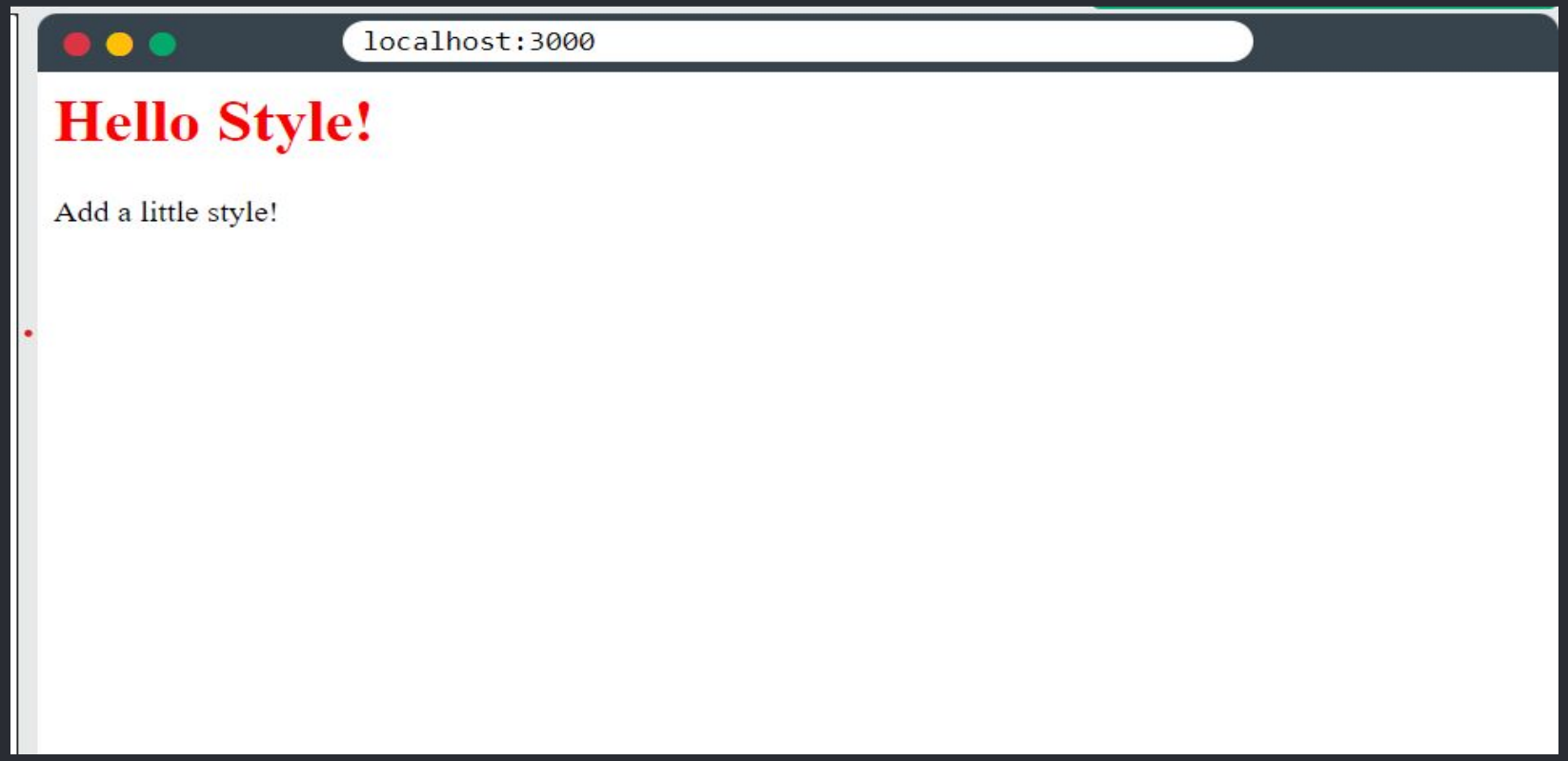
main.js

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3
4  const Header = () => {
5  return (
6  <>
7  <h1 style={{color: "red"}} >Hello Style!</h1>
8  <p›Add a little style!‹/p>
9  </>
10 )
11 };
12 const root = ReactDOM. createRoot
13 (document. getelementById('root')));
14 root.render (‹Header />);
```

To style an element with the inline style attribute, the value must be a JavaScript object:

Note: In JSX, JavaScript expressions are written inside curly braces and since JS objects also use curly braces the styling is done in 2 curly braces

{{...}}


localhost:3000
Hello Style!
Add a little style!

What if we want to add multiple styles for an element ?

What if we want to add multiple styles for an element ?

Well, there are several ways.

•Java Script Object:

create an object with styling information, and refer to it in the style attribute:

main.js

```
1  const mystyle = {
2    color: "white"
3
4  backgroundColor: "DodgerBlue"
5  padding: "10px" ,
6  fontFamily: "Arial"
7  };
```

main.js

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client'
3
4  const Header = () => {
5  const myStyle = {
6  color: "white",
7   backgroundColor: "DodgerBlue",
8    padding: "10px",
9     fontFamily: "Sans-Serif"
10 };
11 return (
12   <>
13 <h1 style={myStyle} >Hello Style!</h1>
14 <p>Add a little style!</p>
15 </>
16 )
17 };
```

- *A java script object with all styling information required.*

main.js

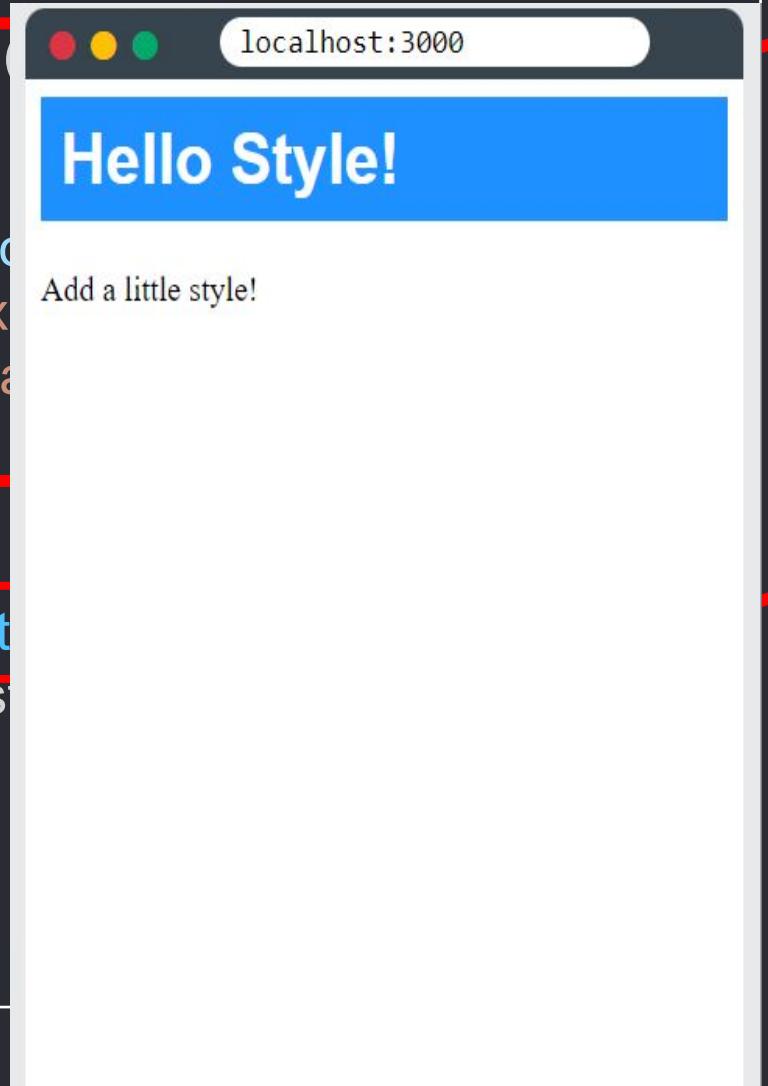
```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client'
3
4  const Header = () => {
5  const myStyle = {
6  color: "white",
7   backgroundColor: "DodgerBlue",
8    padding: "10px",
9     fontFamily: "Sans-Serif"
10 };
11 return (
12   <>
13 <h1 style={myStyle} >Hello Style!</h1>
14 <p>Add a little style!</p>
15 </>
16 )
17 };
```

- *A java script object with all styling information required.*

- *Referring the object with the '=' operator*
- *It's necessary to put the object between*
- *{..} as it's a java script object.*

main.js

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client'
3
4  const Header = (
5  const myStyle =
6  color: "white",
7   backgroundColo
8    padding: "10px
9     fontFamily: "Sa
10 };
11 return (
12   <>
13 <h1 style={myS
14 <p>Add a little s
15 </>
16 )
17 };
```



- *A java script object with all styling information required.*

- *Referring the object with the '=' operator*
- *It's necessary to put the object between*
- *{..} as it's a java script object.*

What's the problem with inline styling?

- We know that inline styling works really well when we’re starting a new application, and we can apply it everywhere we want in our code. But if this concept works fine, why should we stop using it?
- The main disadvantage of the inline styling is the reusability.
- The inline styling concept might not help you to build the best React components in your app. If you’re planning to build a very performant, scalable, and rich application inline styling is not the right option for you.

CSS Stylesheet

Before diving into, Let’s take a quick look at CSS Syntax

Intro to CSS

What is CSS ?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media

Intro to CSS

What is CSS ?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media Is CSS a Programming Language ?

Intro to CSS

What is CSS ?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- Is CSS a Programming Language ?
- ANS: NO, CSS was introduced by the World Wide Web Consortium (W3C) as a **STYLING LANGUAGE**.

Why use CSS ?

- HTML was created to describe the content of a web page, like:
- <h1>This is a heading</h1>
- This is a paragraph.
- <When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers.
- Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.

CSS Selectors

CSS Selector

```
Selector{
Property1:  value;
Property2:  value;
}
```

CSS Selector

```
H1{
    color: blue;
    Padding: 10x;

    }
```

Selects all H1 elements

CSS Selector

```
header {
    color: blue;
}
```

CSS Selector

```
div {
    color: blue;
}
```

CSS Selector

CSS Selectors are divided into :

- Element Selector
- Class Selector
- ID Selector

Class Selectors

- All HTML elements can have attributes such as the href in anchor tag

```
<a href =“index.html”>Home</a>
```

- A class is just an attribute that all HTML elements can have and is used with CSS to distinguish.

Class Selectors

- To select an element with a class we need to use a period ‘ . ’ before the class selector.

```
.class-name {
property: value;
}
```

```
.btn{
background-color: orange;
padding: 10px 20px;
}
```

Class Selectors

- A class attribute can also have multiple different classes in the same attribute as long as they are separated by space.

```
<div class =“red square” >
Content
</div>
```

App.css

```
.btn{
    background-color: orange;
    padding: 10px 20px;
}
```

App.css

```css
.btn{
    background-color: orange;
    padding: 10px 20px;
}
```

App.css

```css
    .btn-1{
    background-color: green;
            }
.btn-2{
    background-color: blue;
        }
.btn-3{
    background-color: purple;
        }
```

App.css

```
.btn{
    background-color:
orange;
    padding: 10px 20px;
}
```

- ***Base button class that all have the same style but differ at colors.***

App.css

```
.btn-1{
    background-color:
green;
}
.btn-2{
    background-color:
blue;
}
.btn-3{
    background-color:
purple;
}
```

- ***Each class child class specify the Color of each buttons.***

App.js

```
1 ‹button className= "btn btn-1"›button1</button>
2 <button className ="btn btn-2">button2</button>
3 <button className= "btn btn-3">button3</button>
```

App.js

```
1 ‹button className= "btn btn-1"›button1</button>
2 <button className ="btn btn-2">button2</button>
3 <button className= "btn btn-3">button3</button>
```

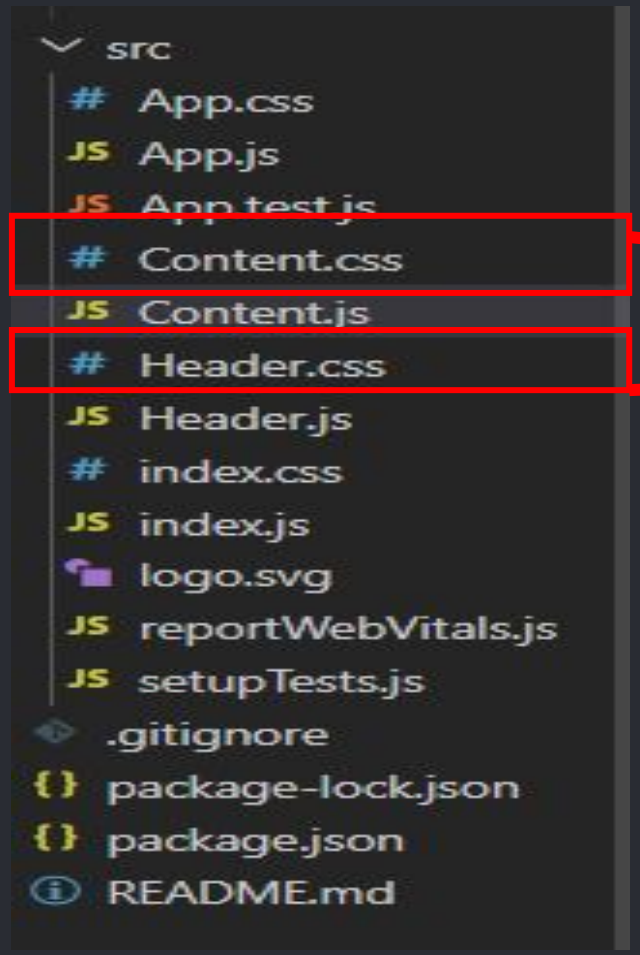
ID Selectors

- They are very similar to the class selector that it's an HTML attribute, but an element can have only ***one id*** and multiple classes.

```
#id {
Property: value;
}
```

CSS Stylesheet

CSS Stylesheet



- *write the CSS in separate file, just save the file in "**.**css" extension and import it in your application.*

App.css

```css
body {
  background-color: #282c34;
  color: white;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}
```

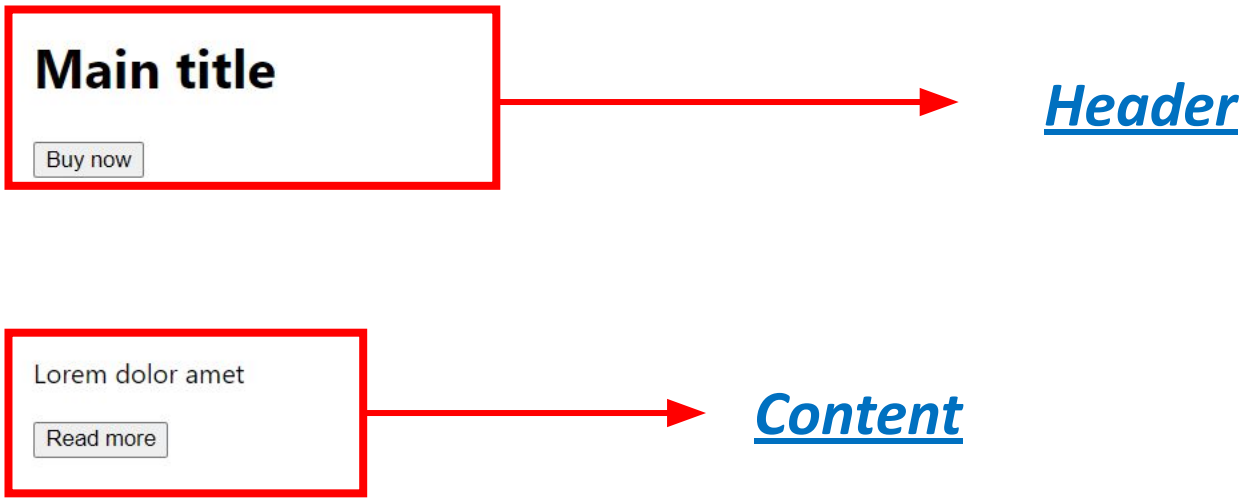
App.css

```css
body {
  background-color: #282c34;
  color: white;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}
```

App.js

```jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import  '. /App.css';
const Header = () => {
return (
< >
<h1>Hello Style!</h1>
<p>Add a little style!.</p>
</>
);
}
```

Let’s do a quick styling



Header

Content

index.css

```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: burlywood;
}
```

index.css

```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: antialias
  -moz-osx-font-smoothing: grays

background-color: burlywood;
}
```

Let’s try to style the buttons

content.css

```
.btn{

    padding: 10px;
    color: royalblue;
}


```

Header.css

```
.btn{
    background-color: orange;
    padding: 10px 20px;
}


```

content.css

```
1   .btn{
2
3       padding: 10px;
4       color: royalblue;
5   }
6
7
```

Header.css

```
1   .btn{
2       background-color: orange;
3       padding: 10px 20px;
4   }
5
6
7
```

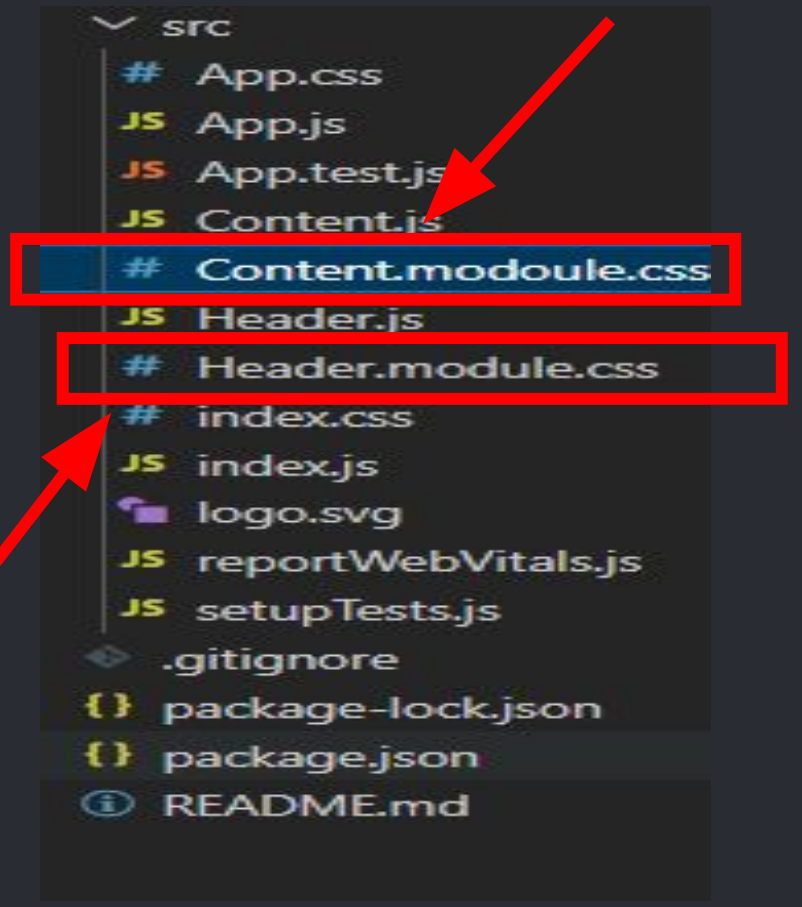
What will happen ??

CSS Modules

- CSS Modules is popular in React because of the following reasons:

1. **Scoped**: CSS Modules are scoped when you use them the right way.
2. **Highly composable**: You can compose different styles in a lot of ways.

- For each component you add a CSS styling module with the name convention *.module.css



Content.module.css

```
1   .btn{
2
3       padding: 10px;
4       color: royalblue;
5   }
6
7
```

Header.module.css

```
1   .btn{
2       background-color: orange;
3       padding: 10px 20px;
4   }
5
6
7
```

What will happen ??