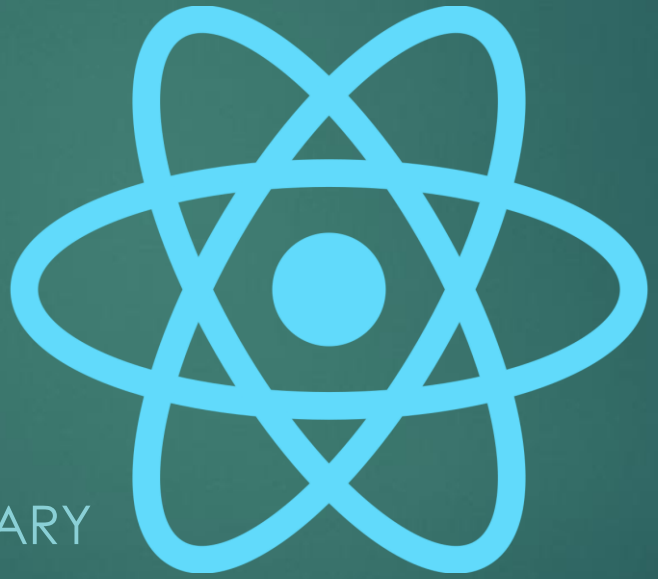
React.js



A WALKTHROUGH OF THE LIBRARY

Zavaar Shah

Compiled by Zavaar Shah https://github.com/thatziv

What is React?

- React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.
- Hello world:

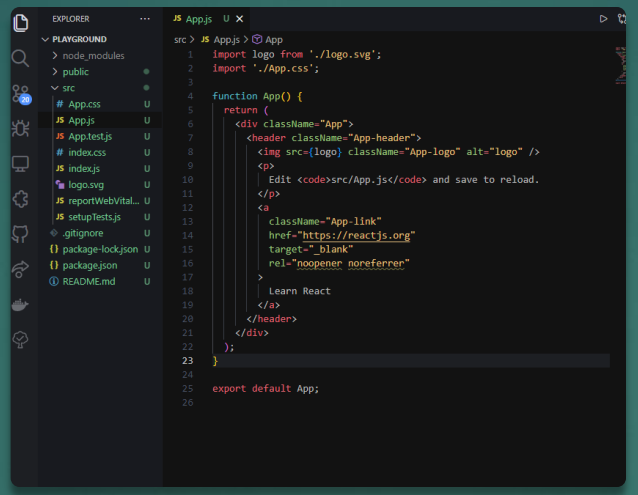
```
function App() {
  return (
    <h1>Hello World</h1>
  );
}

export default App;
```

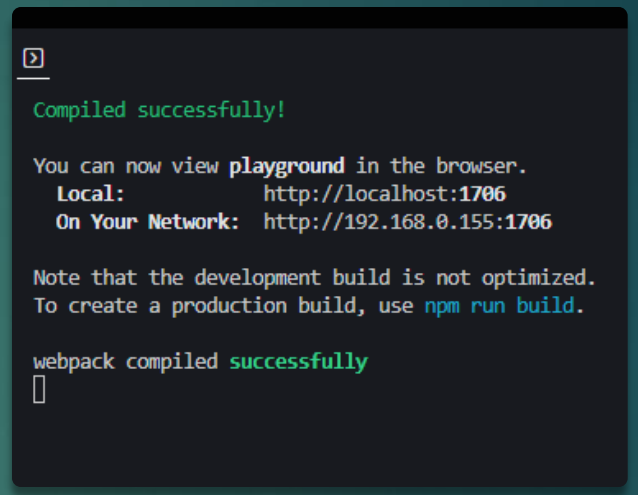
Directly use HTML syntax within JS environment (JSX)

Setup Environment

- Download nodejs
  - https://nodejs.org/en/downloads
- Install NPX globally (terminal)
  - `npm install -g npx`
- Create React app project
  - `npx create-react-app my-app`
  - `cd my-app`
  - `npm start`

Development Environment





Website Preview

https://localhost:3000 by default

Default Code

Console Output

*(after **npm start**)*

Compiled by Zavaar Shah https://github.com/thatziv

React Basics - JSX

Now that we have our project setup, let's learn the basics of React. We can start by understanding the JSX structure of component-based rendering in React.

```
function App() {
  return (
    <div>
      <p>This is a paragraph tag like in HTML!</p>
    </div>
  );
}

export default App;
```

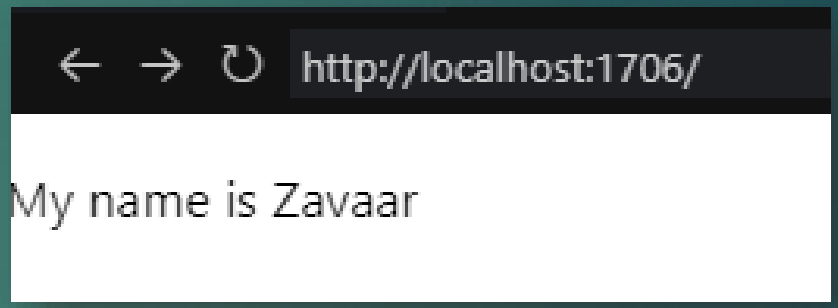
Traditional HTML syntax applies

React Basics - JSX

Embedding Expressions

```jsx
function App() {
  let name = "Zavaar"
  return (
    <div>
      <p>My name is {name}</p>
    </div>
  );
}

export default App;
```

Output



By wrapping your expression around curly braces, the code will execute within.

JSX is an expression too!

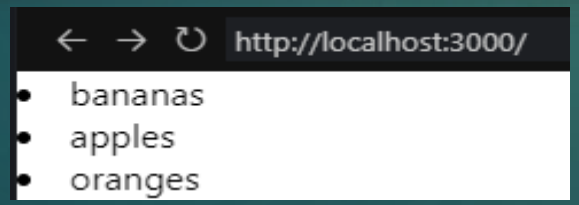
```
function App() {
  let name = <p>My name is Zavaar</p>;
  return (
    <div>
      {name}
    </div>
  );
}

export default App;
```

Compiled by Zavaar Shah https://github.com/thatziv

React Basics - JSX

Iterative rendering

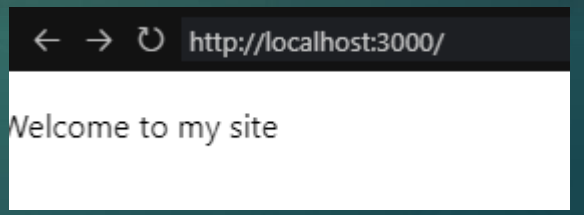
```
function App() {
  let myGroceries = ["bananas", "apples", "oranges"]
  return (
    <div>
      {myGroceries.map((item) => <li>{item}</li>)}
    </div>
  );
}

export default App;
```



Conditional rendering

```
function App() {
  let myAge = 19;
  const Welcome = () => <p>Welcome to my site</p>;
  const Denied = () => <h1>You are not allowed on this site!</h1>;
  return (
    <div>
      {myAge >= 18 ? <Welcome/> : <Denied/>}
    </div>
  );
}

export default App;
```

Functions that return JSX are called components. They allow for code modularity.



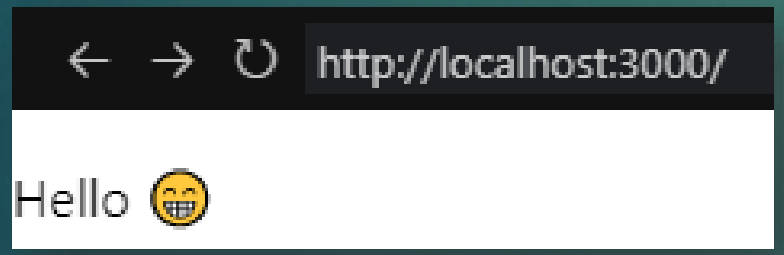
You can "call" those functions with the JSX syntax to render them

Compiled by Zavaar Shah https://github.com/thatziv

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

React Basics - Components

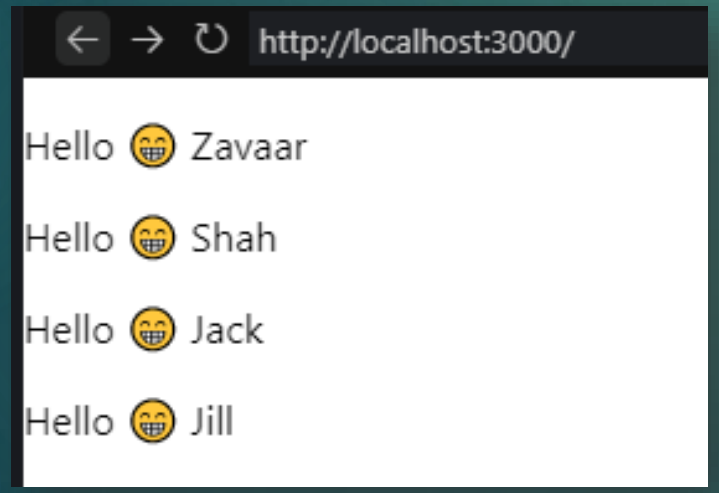
There's a better way to bundle up JSX. We can use functions that return JSX – better known as **components.** Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.




```
function HelloMsg() {
  return <p>Hello 😁</p>
}

function App() {
  return (
    <div>
      <HelloMsg></HelloMsg>
    </div>
  );
}

export default App;
```

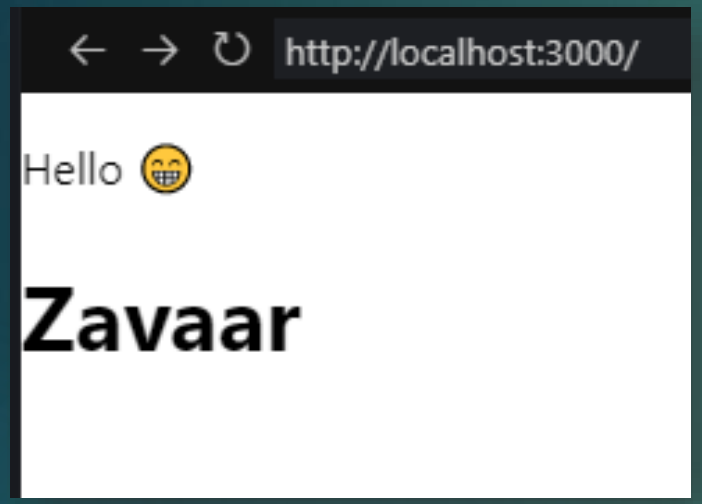
React Basics – Component Props

- When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object "props".




```jsx
function HelloMsg(props) {
  return <p>Hello 😁 {props.name}</p>
}

function App() {
  return (
    <div>
      <HelloMsg name="Zavaar"/>
      <HelloMsg name="Shah"/>
      <HelloMsg name="Jack"/>
      <HelloMsg name="Jill"/>
    </div>
  );
}

export default App;
```

React Basics – Component Children




```
function HelloMsg(props) {
  return <p>Hello 😁 {props.children}</p>
}

function App() {
  return (
    <div>
      <HelloMsg>
        <h1>Zavaar</h1>
      </HelloMsg>
    </div>
  );
}

export default App;
```

So far, all of this has been static.

Let's introduce hooks. Hooks are the "primitives" of the React framework that give access to lower-level features of the frameworks within components.

- Basic Hooks
  - `useState`
  - `useEffect`
  - `useContext`
- Additional Hooks
  - `useReducer`
  - `useCallback`
  - `useMemo`
  - `useRef`
  - `useImperativeHandle`
  - `useLayoutEffect`
  - `useDebugValue`
  - `useDeferredValue`
  - `useTransition`
  - `useId`

How do we use reactive values?

For reactive values, we can use the **useState** hook. Why use a state instead of just declaring a temporary variable and redefining that? In order to show updated content, your component will unmount, remount and now that data is gone. Another problem with this is that our component wouldn't know when to re-render when our temp variable changes. That is why we have states. States allow for the persistence of data within components throughout a component's lifecycle. We will learn the stages of a component's lifecycle later (**useEffect**).

```
import React from "react";

function App() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      {count}
    </div>
  );
}

export default App;
```

Default state value

A couple things worth mentioning:

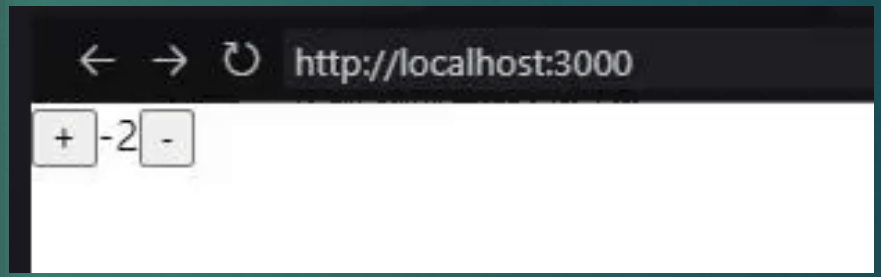
- "useState" returns an array with a "getter" and "setter" function. We can use take advantage of this by destructuring them.
- The "count" constant itself is your "getter" value for that the count state
- The "setCount" is a function that is your "setter" function for the count state. This is the only way to manipulate your count variable state.
- Whenever the "count" variable is changed, those changes will get reflected in realtime.

React Basics – Hooks: **useState**

React Basics – Hooks: **useState**

- Now how would we change the state of our counter? **setCount**

```jsx
import React from "react";

function App() {
  const [count, setCount] = React.useState(0);
  const increment = () => setCount(count+1);
  const decrement = () => setCount(count-1);
  return (
    <div>
      <button onClick={increment}>+</button>
      {count}
      <button onClick={decrement}>-</button>
    </div>
  );
}

export default App;
```



React Basics – Hooks: **useEffect**

- Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.
- The three phases are: **Mounting**, **Updating**, and **Unmounting**.

https://www.w3schools.com/react/react_lifecycle.asp

React Basics – Hooks: **useEffect**

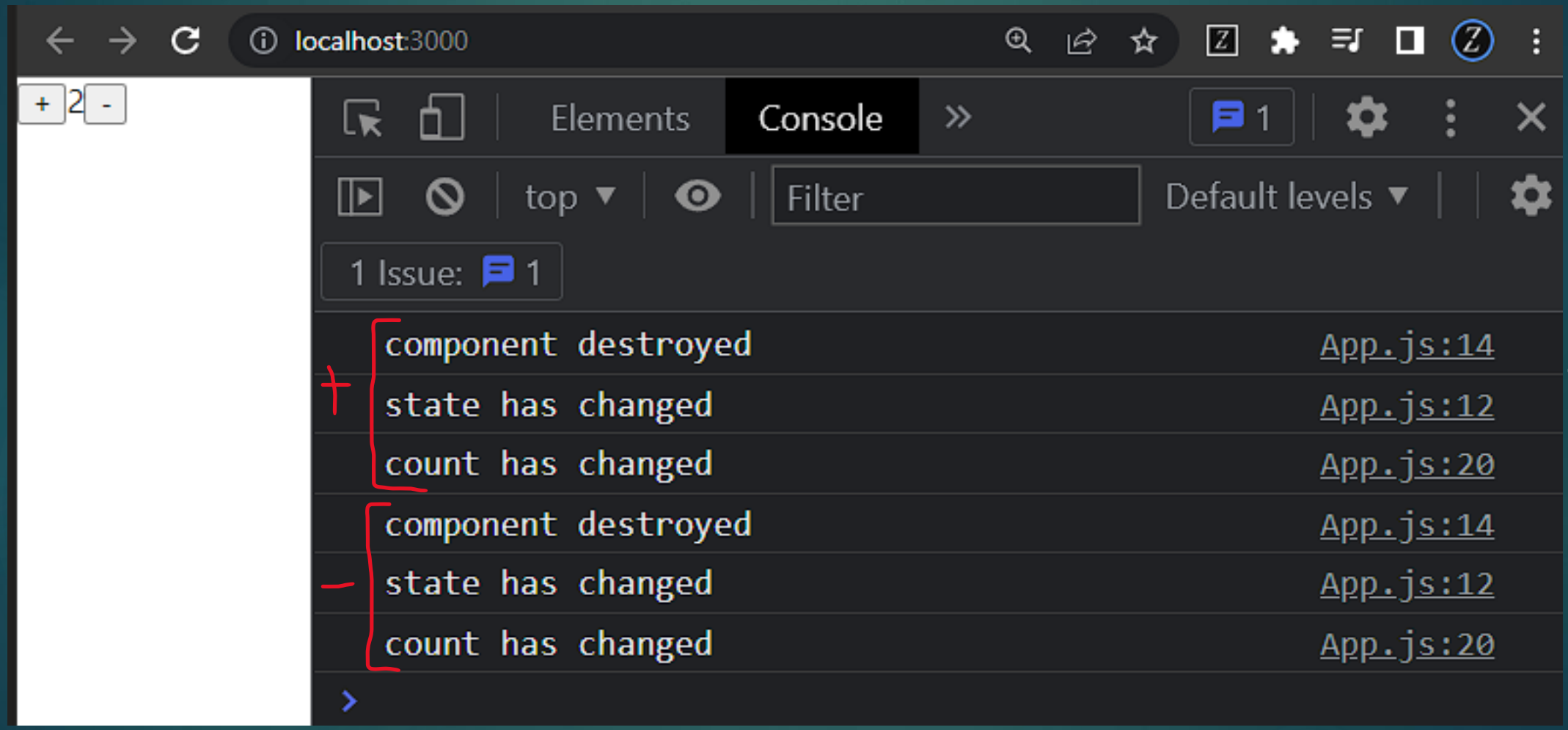
```jsx
import React from "react";

function App() {
  // useState
  const [count, setCount] = React.useState(0);
  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);

  // useEffect
  React.useEffect(
    () => {
      console.log("state has changed")
      // this will run when the component is first mounted and when the state changes
      return () => console.log("component destroyed")
      // this is our teardown function and will run when our component is unmounted
      // think of this as like a destructor in c++
    }
  )
  React.useEffect(() => {
    console.log("count has changed")
    // this will run when the count state has updated
  }, [count]) // the last array argument are our dependencies

  return (
    <div>
      <button onClick={increment}>+</button>
      {count}
      <button onClick={decrement}>-</button>
    </div>
  );
}

export default App;
```

Triggers when **mounted & [any state] updates**

Triggers when **unmounted**

Triggers when “count” state **updates**

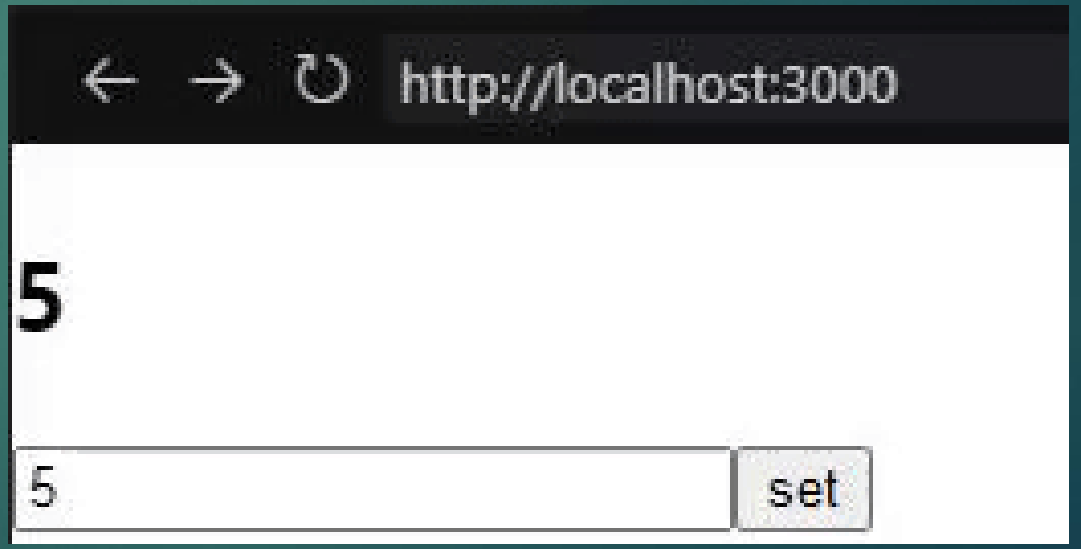
React Basics – Hooks: **useEffect**



React Basics – Hooks: **useRef**

```jsx
import React from "react";

function App() {
  // useState
  const [count, setCount] = React.useState(0);
  // useRef
  const myInput = React.useRef(null);
  const handleClick = () => {
    let myInputValue = myInput.current?.value;
// get the current input value (if it exists)
    if (myInputValue) {
      setCount(myInputValue);
    }
  }
  return (
    <div>
      <h2>{count}</h2>
      <input type="number" ref={myInput}/>
      <button onClick={handleClick}>set</button>
    </div>
  );
}

export default App;
```

- **useRef** allows us to interact with the JS DOM more easily. This let's us take values other elements to parse user input.



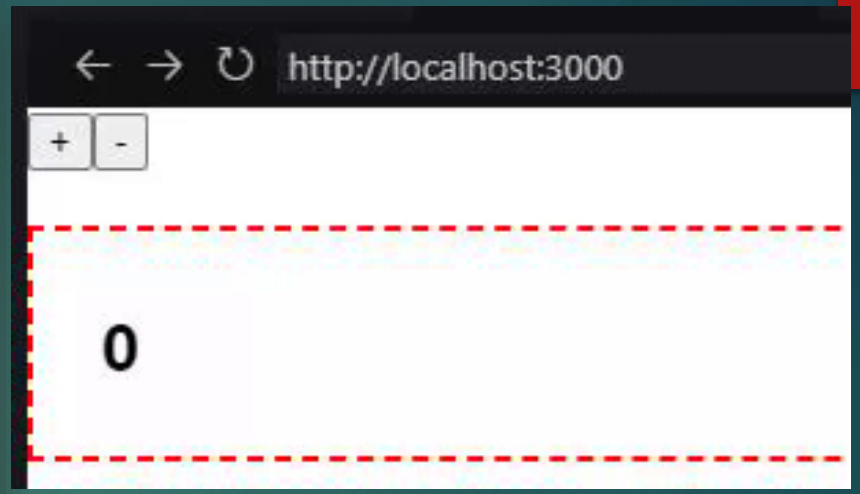
React – Hooks: **useContext**

- Contexts can be a bit tricky but very important when scaling your application up
- Contexts allow sharing data between components without using props.
  - **Context Providers** allow for the use of **useContext** and must be the child of a provider to use its values

React – Hooks: **useContext**

Initialize the context

```jsx
import React from "react";

const defaultCount = 0;
const CounterContext = React.createContext(defaultCount);
// creates the context with the default value of zero

function App() {
  // useState
  const [count, setCount] = React.useState(defaultCount);
  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);

  return (
    <div>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
      <CounterContext.Provider value={count}>
        <CounterDisplay />
      </CounterContext.Provider>
    </div>
  );
}

function CounterDisplay() {
  const count = React.useContext(CounterContext);
  // read the state of the parent component using ctx
  return <h2 style={{ padding: 25, border: "2px dashed red" }}>{count}</h2>
}

export default App;
```



Pass the value of the context provider to be our counter state's value

Take the context's value from its parent (the provider).

React Advanced – Hooks: ***Contexts & Reducers**

- You can also change their value from within nested components.
- This can be done with Reducers.
  - Reducers allow for changes to specific sub-values within the state
  - To change the values of these "sub-values" we would use a dispatch (similar to React-redux).
- ***This topic is beyond the scope of this session.***
  - To learn more about reducers - https://reactjs.org/docs/hooks-reference.html#usereducer

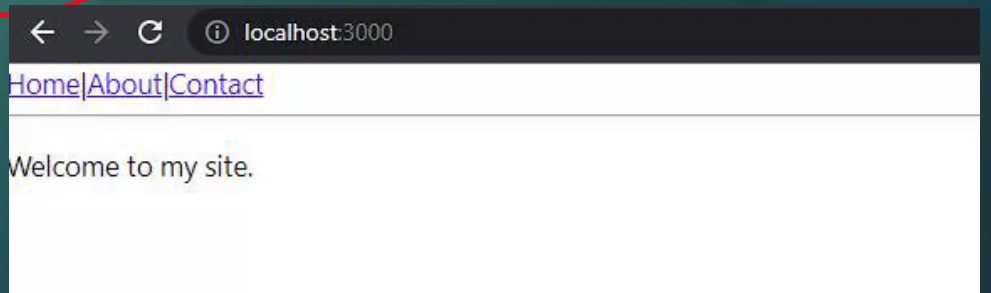
React Advanced – **Routing**

```jsx
import React from "react";
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";

function Home() { // home page
  return <p>Welcome to my site.</p>;
}

function About() { // about page
  return <p>Hey! my name's Zavaar Shah. I'm a CS student
    at <a href="https://wayne.edu">Wayne State</a> University.</p>;
}

function Contact() { // contact page
  return <p>Contact me at https://github.com/thatziv</p>
}

function NotFound() { // 404 page
  return <h1>404 - Not Found</h1>
}

function App() {
  return (
    <div>
      <BrowserRouter>
        <div className="nav"> {/* Nav bar */}
          <Link to="/">Home</Link>
          |{/* <- pipe for style */}
          <Link to="/about">About</Link>
          |
          <Link to="/contact">Contact</Link>
        </div>
        <hr />
        <Routes> {/* Declared routes */}
          <Route element={<Home />} path="/" />
          <Route element={<About />} path="/about" />
          <Route element={<Contact />} path="/contact" />
          <Route element={<NotFound/>} path="*" />
        </Routes>
      </BrowserRouter>
    </div>
  );
}

export default App;
```

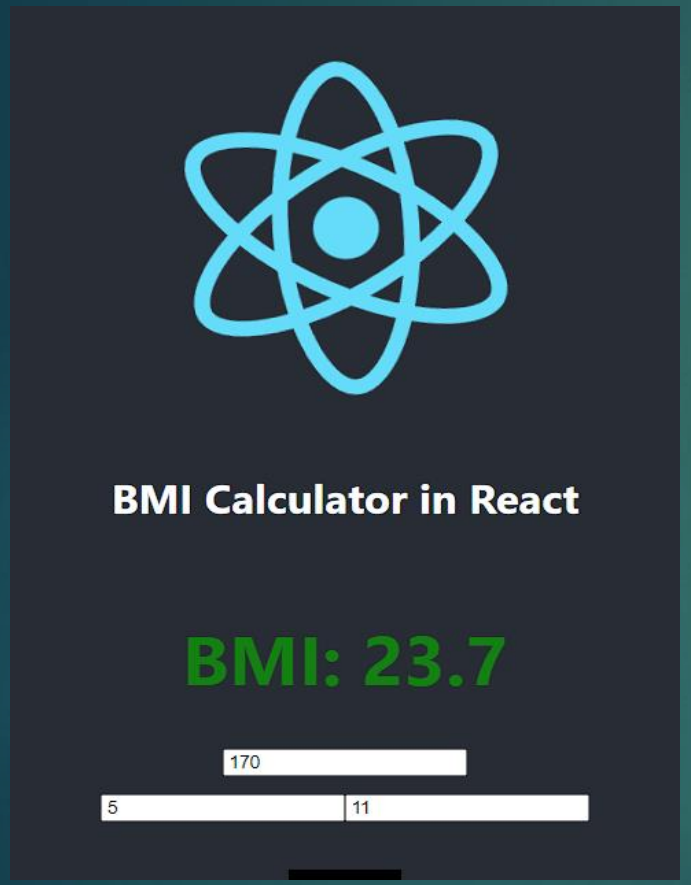
These 4 components represent pages

These "Links" in our navbar allow for the user to navigate our site. It's purposefully outside of any of the pages so that it will render regardless of what route we are on.

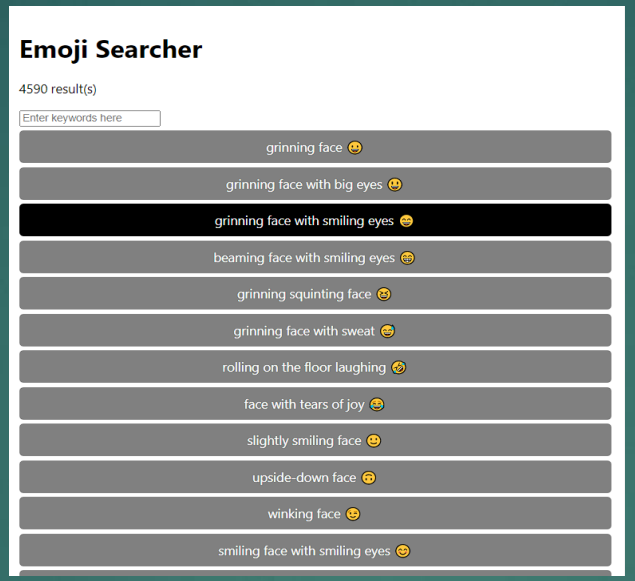
Declare routing for each component to render whenever a user is at that location.



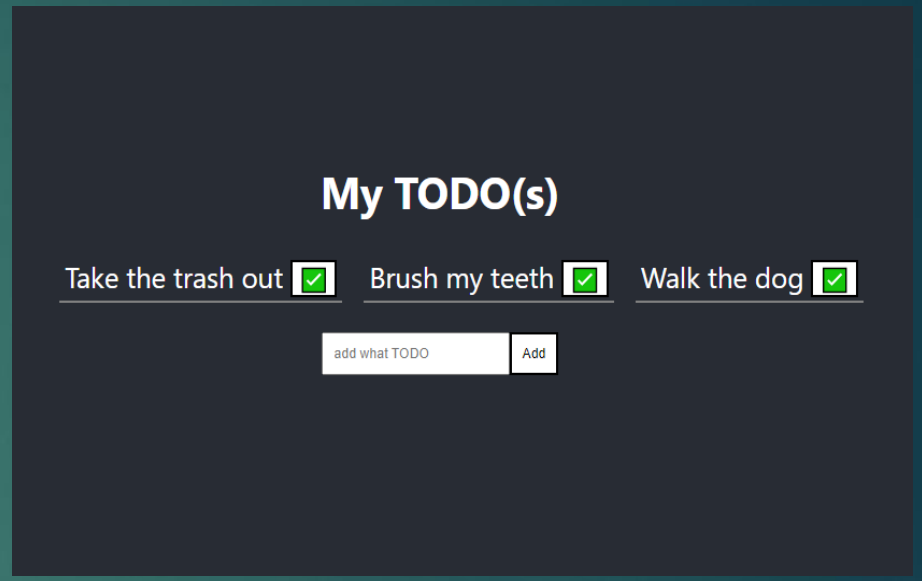
Now, let's try to build a...



BMI Calculator



Emoji Searcher



TODO/Reminder app

https://github.com/WSU-Society-of-Computer-Developers/workshops/tree/main/react_nov28
OR https://s.zavaar.net/9uL9