REACT JS NOTES

INTRODUCTION

- React is a JavaScript library for building user interfaces.
- React is used to build single-page applications.
- React allows us to create reusable UI components.

What is React?

- React, sometimes referred to as a frontend JavaScript framework, is a JavaScript library created by Facebook.
- React is a tool for building UI components.

- **To get an overview of what React is, you can write React code directly in HTML.**
- **But in order to use React in production, you need npm and** Node.js **installed.**

Installing React App

Create React App

- Create React App is a comfortable environment for learning React, and is the best way to start building a new single-page application in React.

- It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production. You'll need to have Node >= 14.0.0 and npm >= 5.6 on your machine. To create a project, run:

```
npx create-react-app my-app
```

- The `create-react-app` will set up everything you need to run a React application.

Run the React Application

- Run this command to move to the `my-react-app` directory:

```
cd my-app
```

- Run this command to open the app in your text editor

```
code .
```

- Run this command to run your react app

```
Npm start
```

React Render HTML

- React's goal is in many ways to render HTML in a web page.
- React renders HTML to the web page by using a function called `ReactDOM.render()`.

The Render Function

- The `ReactDOM.render()` function takes two arguments, HTML code and an HTML element.
- The purpose of the function is to display the specified HTML code inside the specified HTML element.
- But render where?
- There is another folder in the root directory of your React project, named "public". In this folder, there is an `index.html` file.
- You'll notice a single `<div>` in the body of this file. This is where our React application will be rendered.

The result is displayed in the `<div id="root">` element:

Index.html file

```
<body>

  <div id="root"></div>

</body>
```

App.js file

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

The HTML Code

- The HTML code in this REACT uses JSX which allows you to write HTML tags inside the JavaScript code:

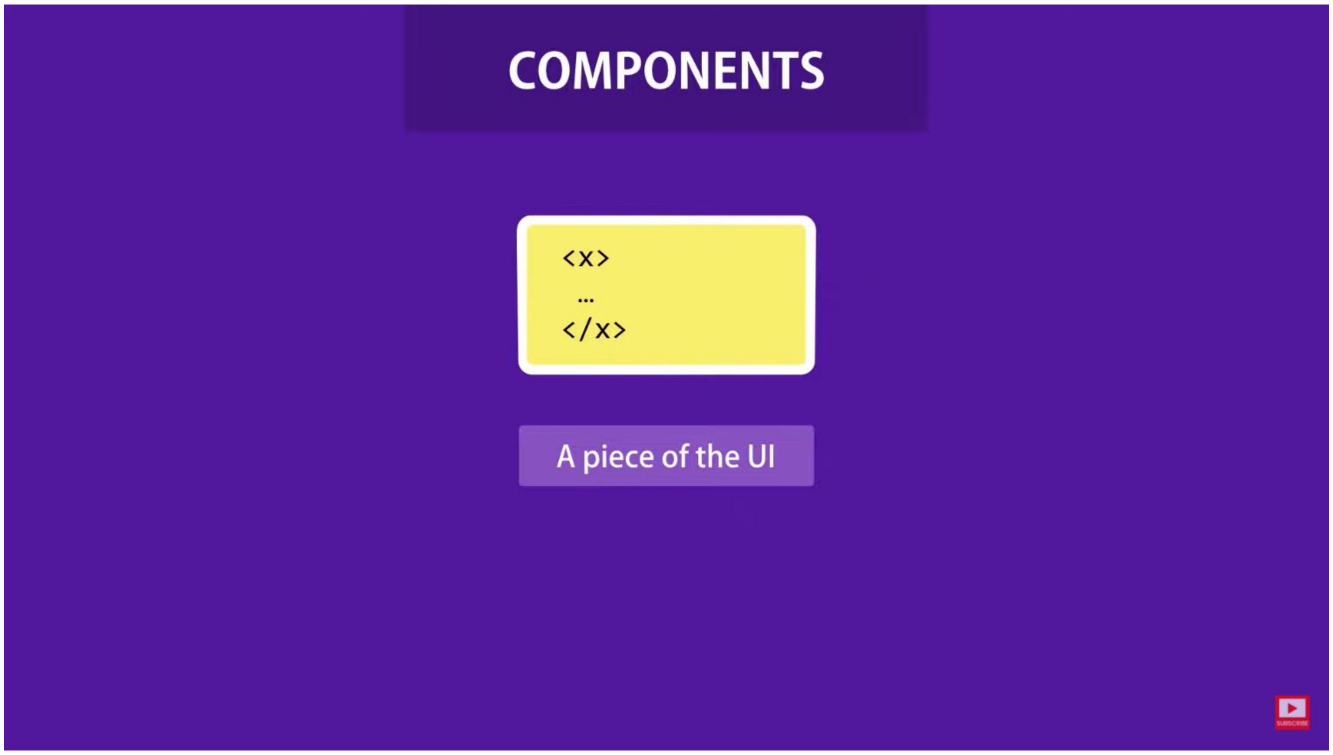
React JSX

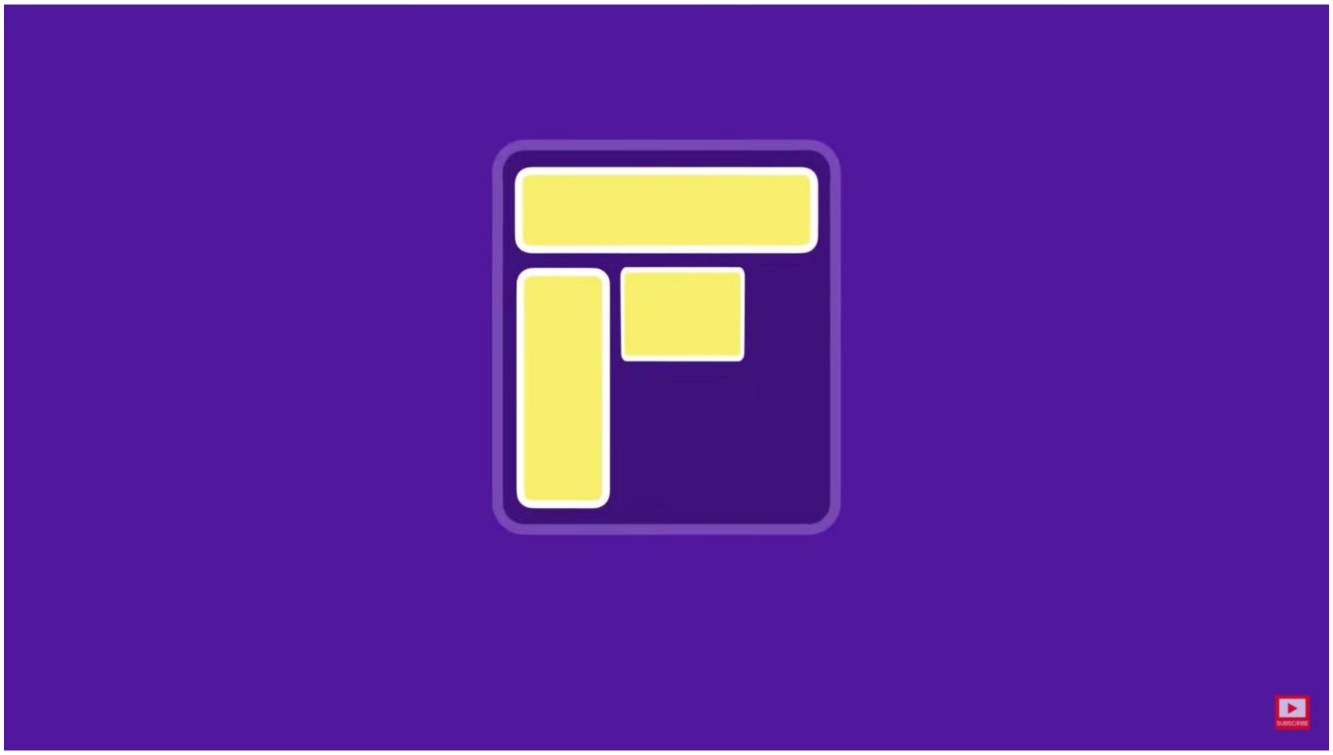
What is JSX?

- JSX stands for JavaScript XML.
- JSX allows us to write HTML in React.
- JSX makes it easier to write and add HTML in React.

React Components

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.



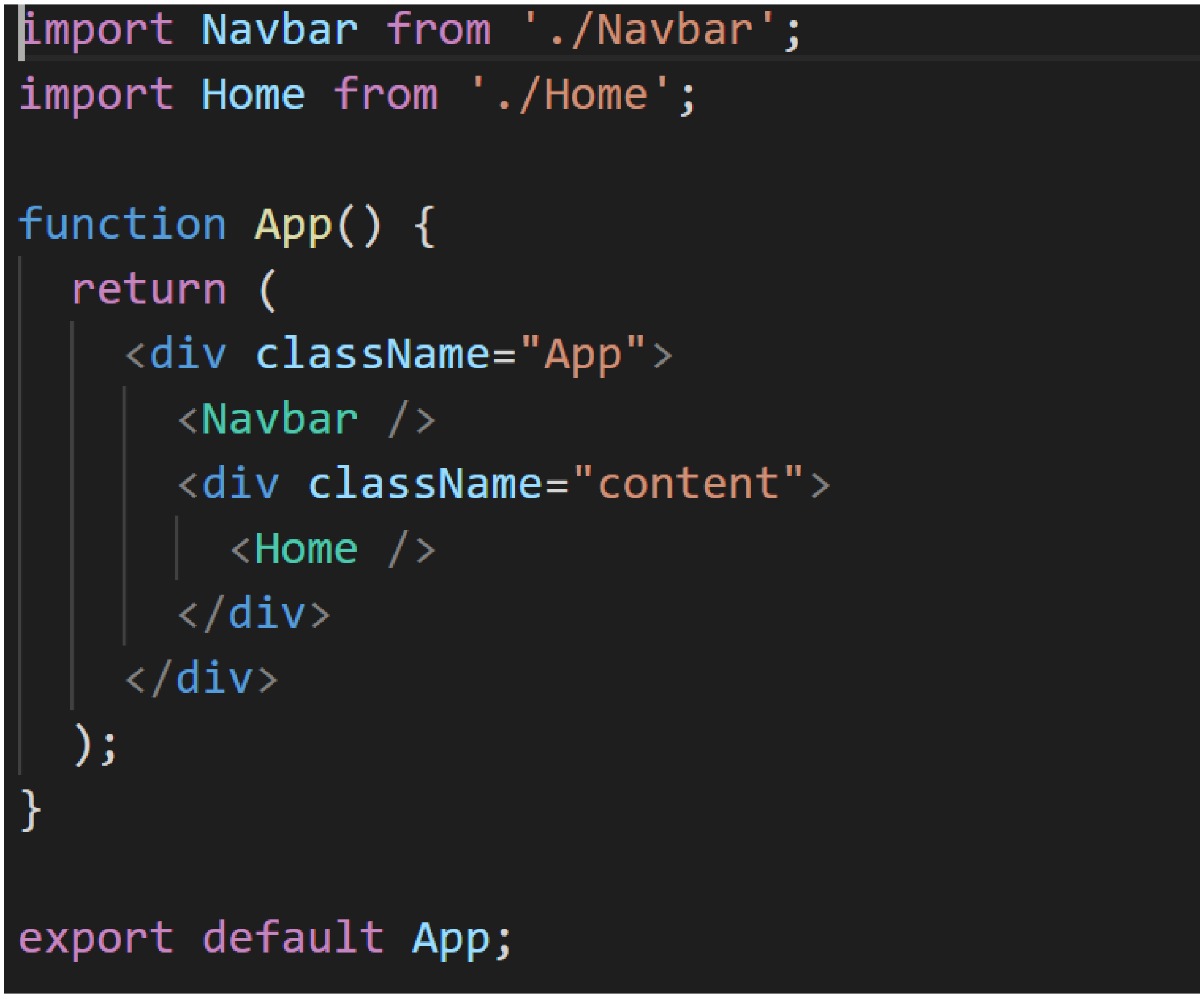


Steps to import and use a component.

1. Create a file.
2. Import that file in your parent component

   Eg: import ComponentName from "…path"
3. Use that file in your JSX code

   Eg: <ComponentName />



```
import Navbar from './Navbar';
import Home from './Home';

function App() {
  return (
    <div className="App">
      <Navbar />
      <div className="content">
        <Home />
      </div>
    </div>
  );
}

export default App;
```

Styling React Using CSS

Inline Styling

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

Example:

Insert an object with the styling information:

```
const Header = () => {
  return (
    <>
      <h1 style={{color: "red"}}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}
```

camelCased Property Names

Since the inline CSS is written in a JavaScript object, properties with hyphen separators, like `background-color`, must be written with camel case syntax:

Example:

Use `backgroundColor` instead of `background-color`:

```
const Header = () => {
  return (
    <>
      <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}
```

JavaScript Object

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

Example:

Create a style object named myStyle:

```
const Header = () => {
  const myStyle = {
    color: "white",
    backgroundColor: "DodgerBlue",
    padding: "10px",
    fontFamily: "Sans-Serif"
  };
  return (
    <>
      <h1 style={myStyle}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}
```

CSS Stylesheet

You can write your CSS styling in a separate file, just save the file with the .css file extension, and import it in your application.

App.css:

Create a new file called "App.css" and insert some CSS code in it:

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

Import the stylesheet in your application:

App.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';

const Header = () => {
  return (
    <>
      <h1>Hello Style!</h1>
      <p>Add a little style!.</p>
    </>
  );
}
```

React Hooks

What is a Hook?

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

Hook Rules

- You must `import` Hooks from `react`.
- Hooks can only be called inside React function components.
- Hooks can only be called at the top level of a component.

React UseState Hook

The React `useState` Hook allows us to track state in a function component.
State generally refers to data or properites that need to be tracking in an application.

Import `useState`

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

Example:
At the top of your component, `import` the `useState` Hook.

```
import { useState } from "react";
```

Initialize `useState`

We initialize our state by calling `useState` in our function component.

`useState` accepts an initial state and returns two values:

- The current state.
- A function that updates the state.

Example:
Initialize state at the top of the function component.

```
import { useState } from "react";

function FavoriteColor() {
  const [name, setName] = useState("");
}
```

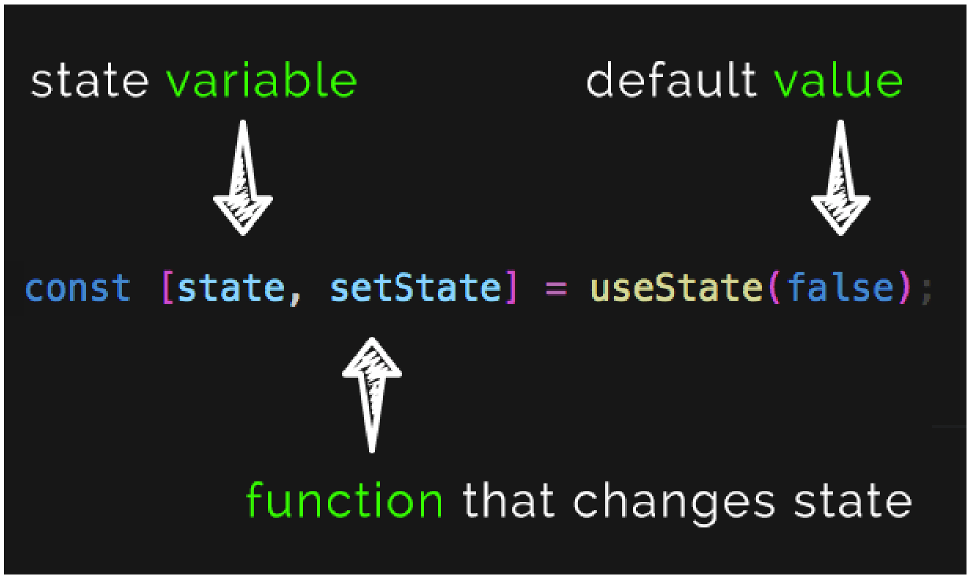
- The first value, `color`, is our current state.
- The second value, `setColor`, is the fuction that is used to update our state.
- Lastly, we set the initial state to an empty string: useState("Ross Geller")

These names are variables that can be named anything you would like.

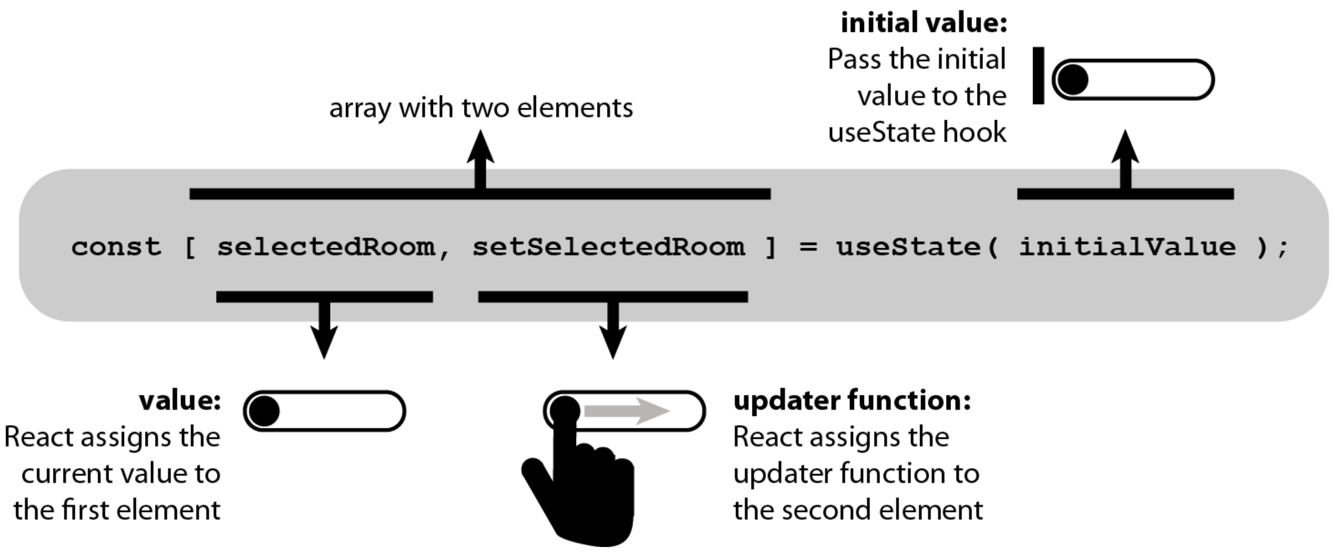
We can now include our state anywhere in our component.
Example:
Use the state variable in the rendered component.

```
import { useState } from "react";

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

       function handleDelete() {
              setName(“Chandler Bing”);
       }
       return (
              <p>{name}</p>
              <button onclick={handleDelete}>Click<button>
       )
```

The useState Hook can be used to keep track of strings, numbers, booleans, arrays, objects, and any combination of these…!



```
const [state, setState] = useState(false);
```


initial value:
Pass the initial value to the useState hook
array with two elements
value:
React assigns the current value to the first element
updater function:
React assigns the updater function to the second element

```
const [ selectedRoom, setSelectedRoom ] = useState( initialValue );
```

React `useEffect` Hook

- The `useEffect` Hook allows you to perform side effects in your components.
- Some examples of side effects are: fetching data, directly updating the DOM, and timers.
- `useEffect` accepts two arguments. The second argument is optional.
- **useEffect(<function>, <dependency>)**

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom";

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

  function handleDelete() {
              setName("Chandler Bing");
       }

  useEffect(() => {
    console.log("Use effect ran…!")
  });

  return (
              <p>{name}</p>
              <button onclick={handleDelete}>Click<button>
       )
}
```

- useEffect runs on every render.
- We should always include the second parameter which accepts an array. We can optionally pass dependencies to useEffect in this array.

1. No dependency passed:

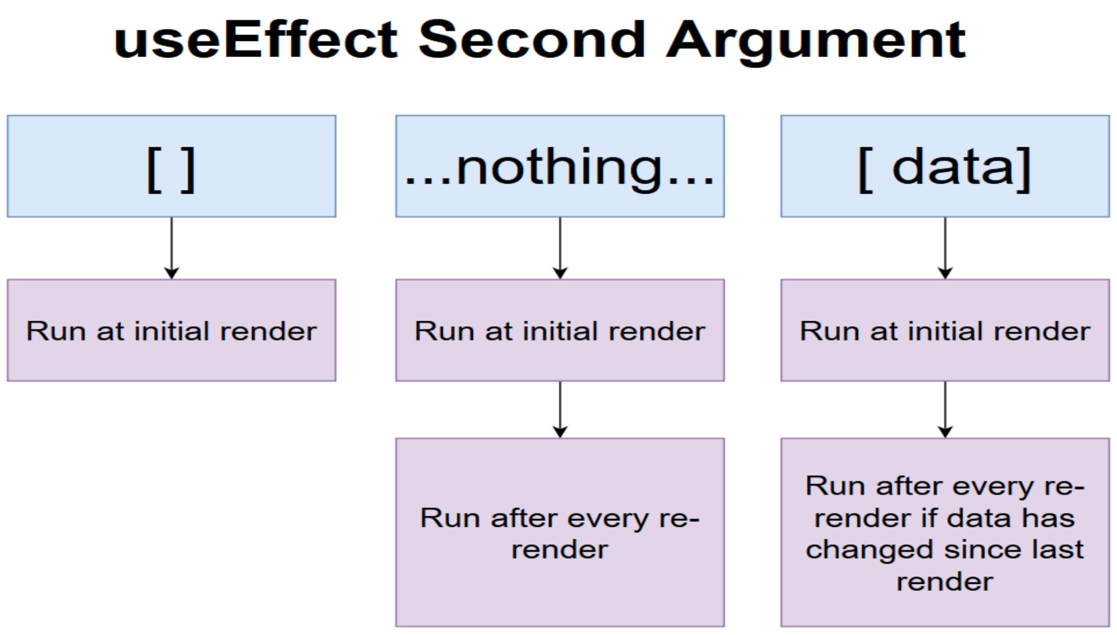
```
useEffect(() => {
  //Runs on every render
});
```

2. An empty array:

```
useEffect(() => {
  //Runs only on the first render
}, []);
```

3. Props or state values:

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [state]);
```


useEffect Second Argument
[ ]
...nothing...
[data]
Run at initial render
Run at initial render
Run at initial render
Run after every re-render
Run after every re-render if data has changed since last render

React Props

- Props are arguments passed into React components.
- Props are passed to components via HTML attributes.

  `props` stands for properties.

- React Props are like function arguments in JavaScript *and* attributes in HTML.
- To send props into a component, use the same syntax as HTML attributes:

Example:
Add a "brand" attribute to the Car element:

```
<BlogList title="All Blogs" />
```

The component receives the argument as a `props` object:

Example:
Use the brand attribute in the component:

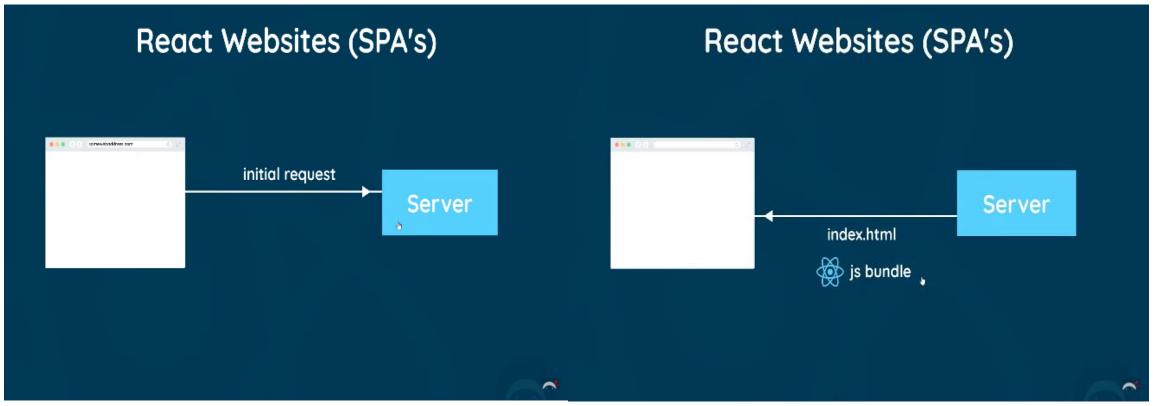
```
function BlogList(props) {
       let title = props.title;
  return (
       <h2>{title} h2>
       );
}
```

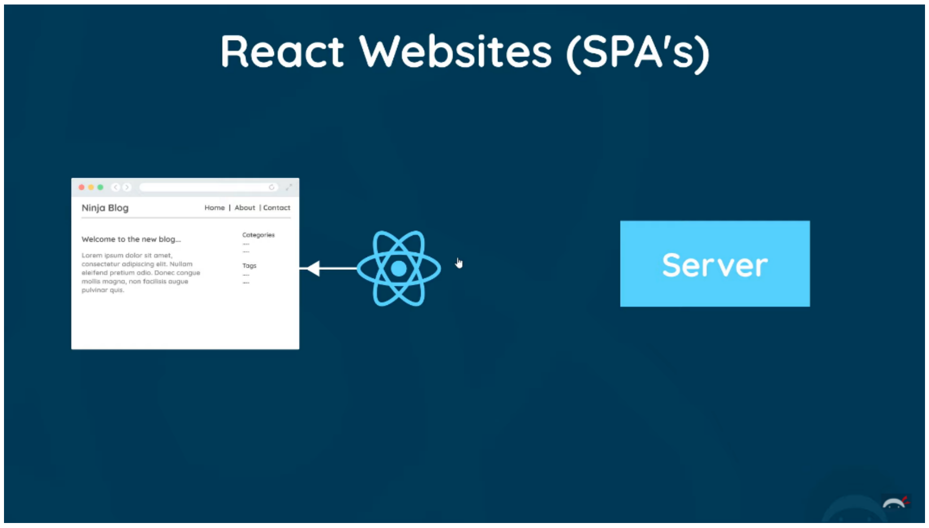
React Routers

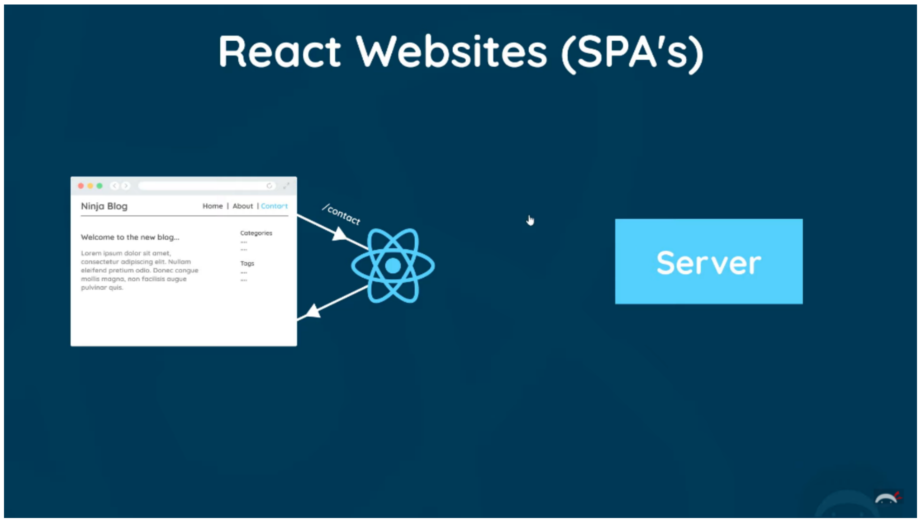
- React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.
- In order to use React Router , we have to install react-router-dom.

Installing React Router: React Router can be installed via npm in your React application.

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







- After installing react-router-dom, add its components to your React application.
- **Adding React Router Components:** The main Components of React Router are:
- **BrowserRouter:** BrowserRouter is a router implementation that uses the HTML5 history API(pushState, replaceState and the popstate event) to keep your UI in sync with the URL. It is the parent component that is used to store all of the other components.
- **Routes:** It's a new component introduced in the v6 and a upgrade of the component. The main advantages of Routes over Switch are:
- Routes are chosen based on the best match instead of being traversed in order.
- **Route:** Route is the conditionally shown component that renders some UI when its path matches the current URL.
- **Link:** Link component is used to create links to different routes and implement navigation around the application. It works like HTML anchor tag.
- To add React Router components in your application, open your project directory in the editor you use and go to app.js file. Now, add the below given code in app.js.

To add React Router components in your application, open your project directory in the editor you use and go to app.js file. Now, add the below given code in app.js.

```
import "./App.css";
import Home from "./home";
import Navbar from "./navbar";
import Create from './create';
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
import BlogDetails from "./blogDetails";
import NotFound from "./NotFound";

function App() {
  return (
    <Router>
      <div className="App">
        <Navbar />
        <hr />
        <div className="content">
          <Switch>
            <Route exact path="/">
              <Home />
            </Route>
            <Route exact path="/create">
              <Create />
            </Route>
            <Route exact path="/blogs/:title">
              <BlogDetails />
            </Route>
            <Route path="*">
              <NotFound />
            </Route>
          </Switch>
        </div>
      </div>
    </Router>
  );
}

export default App;
```

- We wrap our content first with `<BrowserRouter>`.
- Then we define our `<Routes>`. An application can have multiple `<Routes>`. Our basic example only uses one.
- `<Route>`s can be nested. The first `<Route>` has a path of `/` and renders the `Layout` component.
- The nested `<Route>`s inherit and add to the parent route. So the `blogs` path is combined with the parent and becomes `/blogs`.
- The `Home` component route does not have a path but has an `index` attribute. That specifies this route as the default route for the parent route, which is `/`.
- Setting the `path` to `*` will act as a catch-all for any undefined URLs. This is great for a 404 error page.

```
import { Link } from "react-router-dom";

const navbar = () => {
    return (
        <div className="navbar">
            <h1>Blog</h1>
           <div className="links">
                <Link to="/">Home</Link>
                <Link to="/create">Create Blog</Link>
           </div>
        </div>
     );
}

export default navbar;
```

- `<Link>` is used to set the URL and keep track of browsing history.
- Anytime we link to an internal path, we will use `<Link>` instead of `<a href="">`.