React - Udemy Course!

Lean and focused component - based UI library.
React is all about components.

Why components?

1. **Reusability - Don't repeat yourself.**
2. **Separation of concerns - Don't do too many things in one and in the same place (function).**

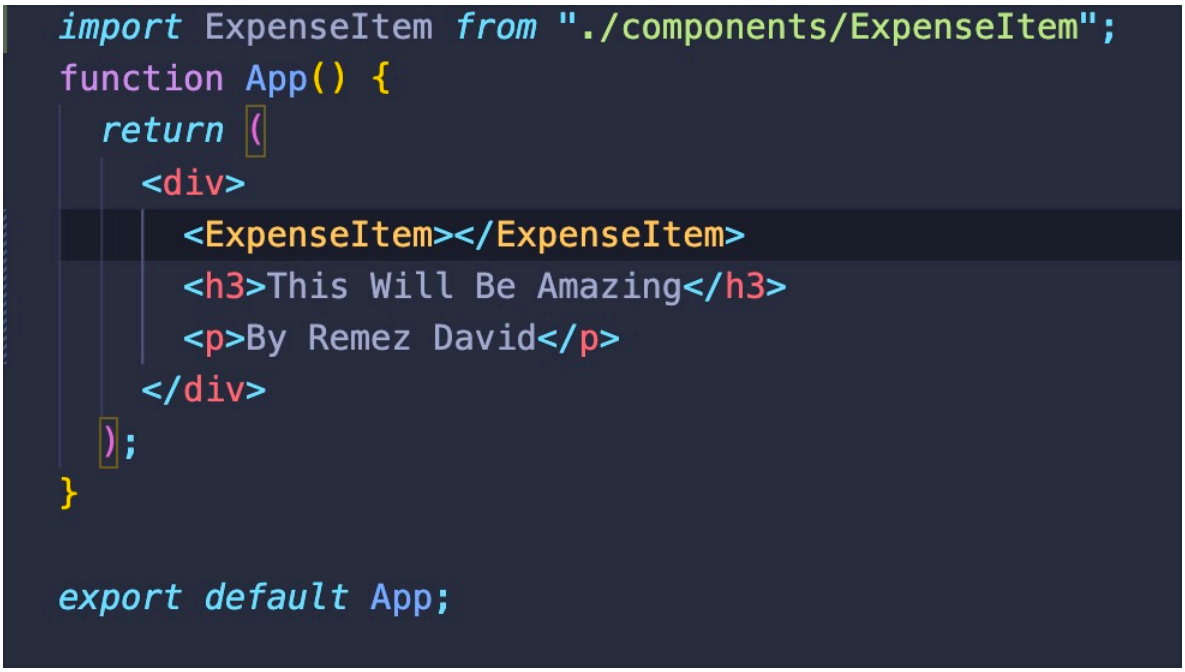
How To Create React App?

Use the command - "npx create-react-app [project_name]".
Run - "npm start" in order to start a development server.

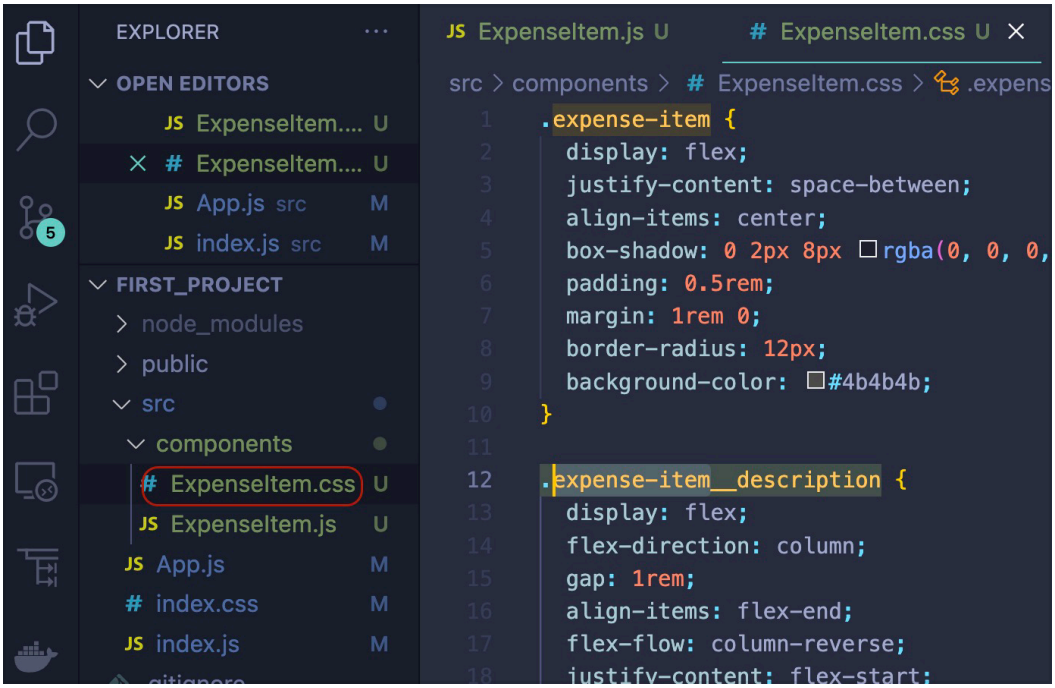
Building a First Custom Component:

File naming convention - Starting with a capital letter (Camel Case).
A component in real its a javaScript function.

Using the custom component inside another component.

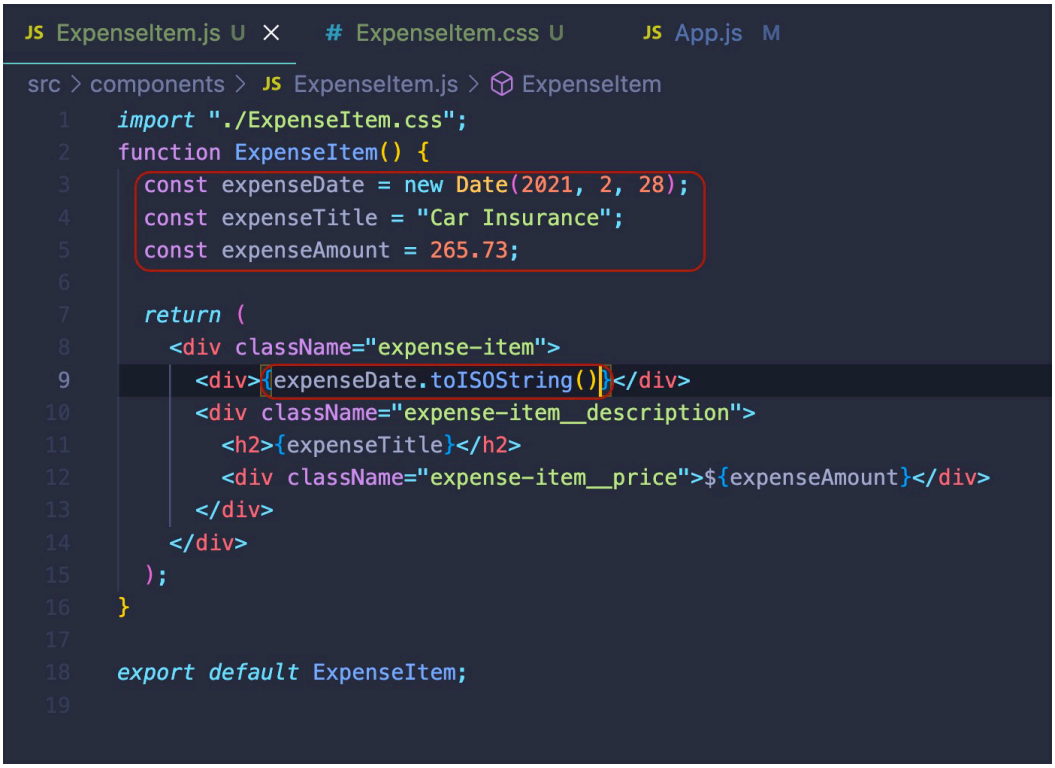


Adding Basic CSS Styling:

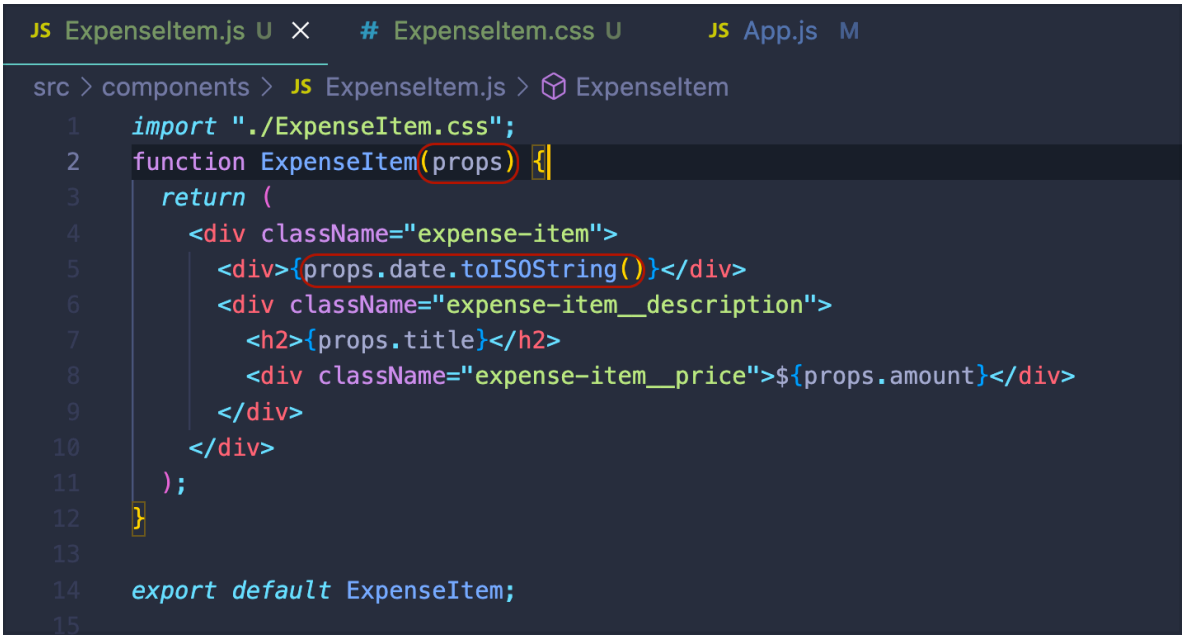


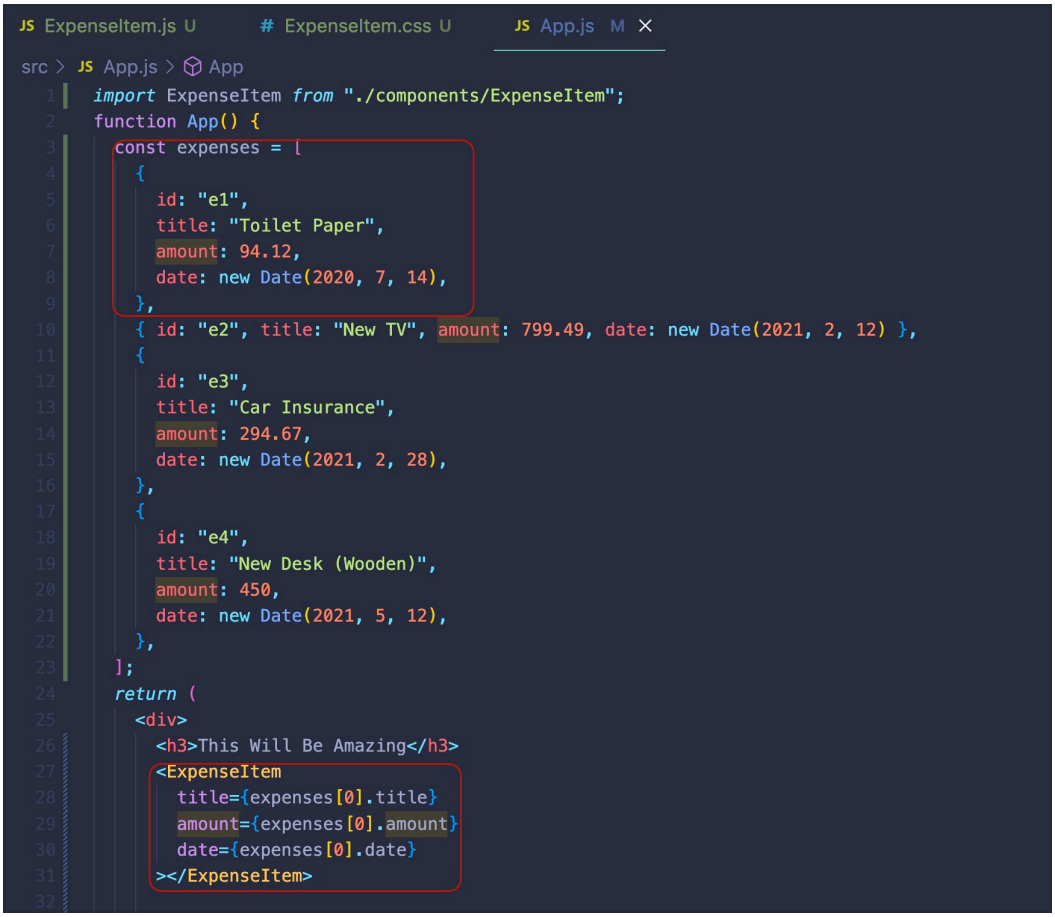
Outputting Dynamic Data & Working with Expressions in JSX:

In order to run basic javaScript we need to use the -> “{}” and inside them we can write javaScript Code



Passing Data via "props":





Splitting Components Into Multiple Components:

We pass data from component A to B component with the props.

The Concept of "Composition" ("children props"):

JS Expenses.js U JS Card.js U # Card.css U

src > components > # Card.css > ...

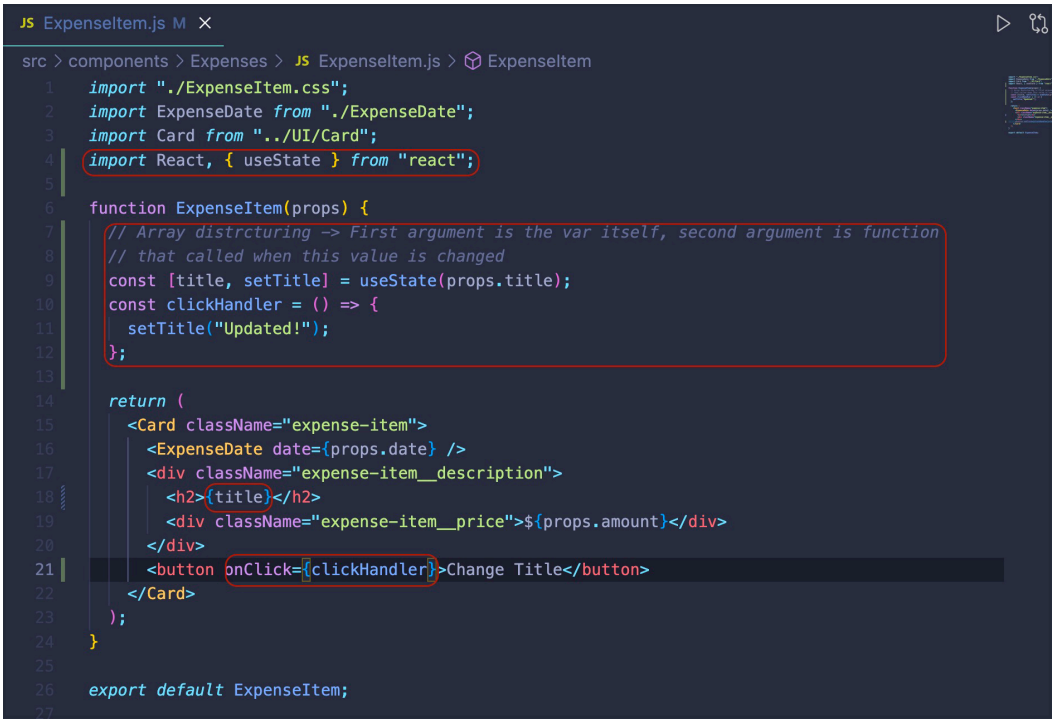
```css
.card {
  border-radius: 12px;
  box-shadow: 0 1px 8px rgba(0, 0, 0, 0.25);
}

```

JS Expenses.js U JS Card.js U

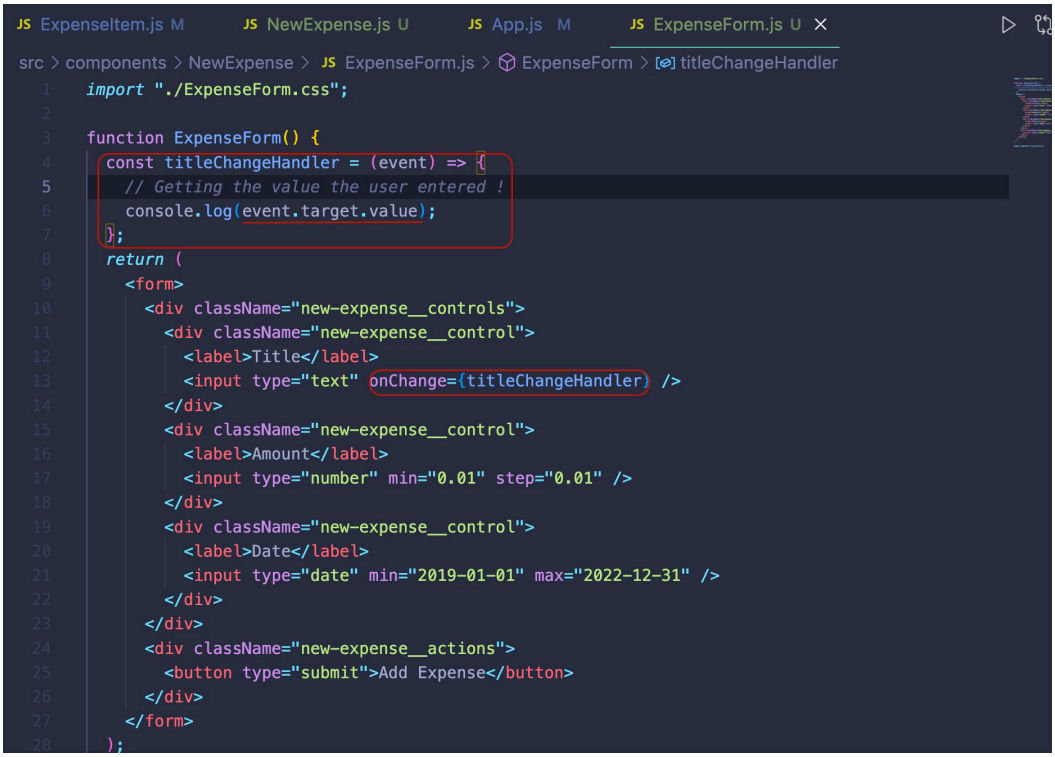
src > components > JS Expenses.js > Expenses

```jsx
import ExpenseItem from "./ExpenseItem";
import "./Expenses.css";
import Card from "./Card";
function Expenses(props) {
  return (
    <Card className="expenses">
      <ExpenseItem
        title={props.items[0].title}
        amount={props.items[0].amount}
        date={props.items[0].date}
      />
      <ExpenseItem
        title={props.items[1].title}
        amount={props.items[1].amount}
        date={props.items[1].date}
      />
      <ExpenseItem
        title={props.items[2].title}
        amount={props.items[2].amount}
        date={props.items[2].date}
      />
      <ExpenseItem
        title={props.items[3].title}
        amount={props.items[3].amount}
        date={props.items[3].date}
      />
    </Card>
  );
}

export default Expenses;

```

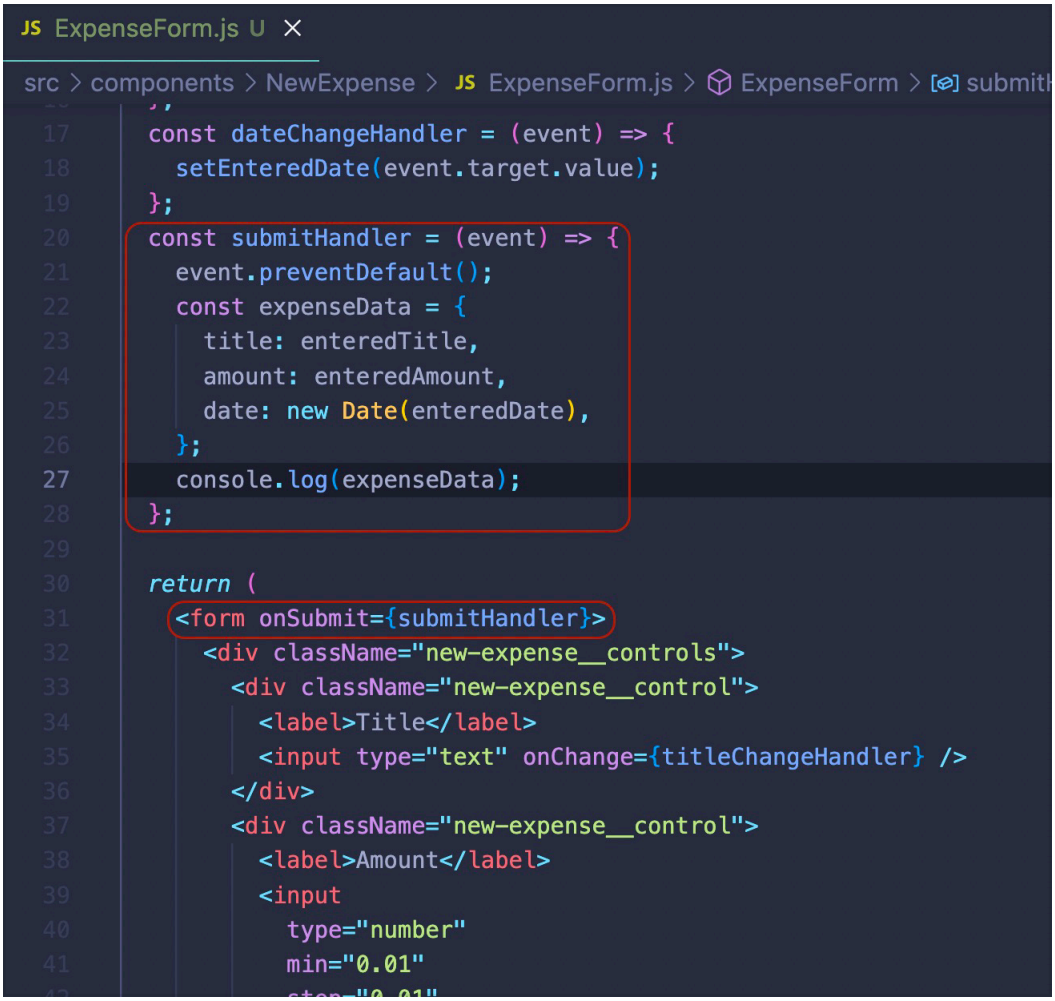
Working with "State":



Listening to User Input:



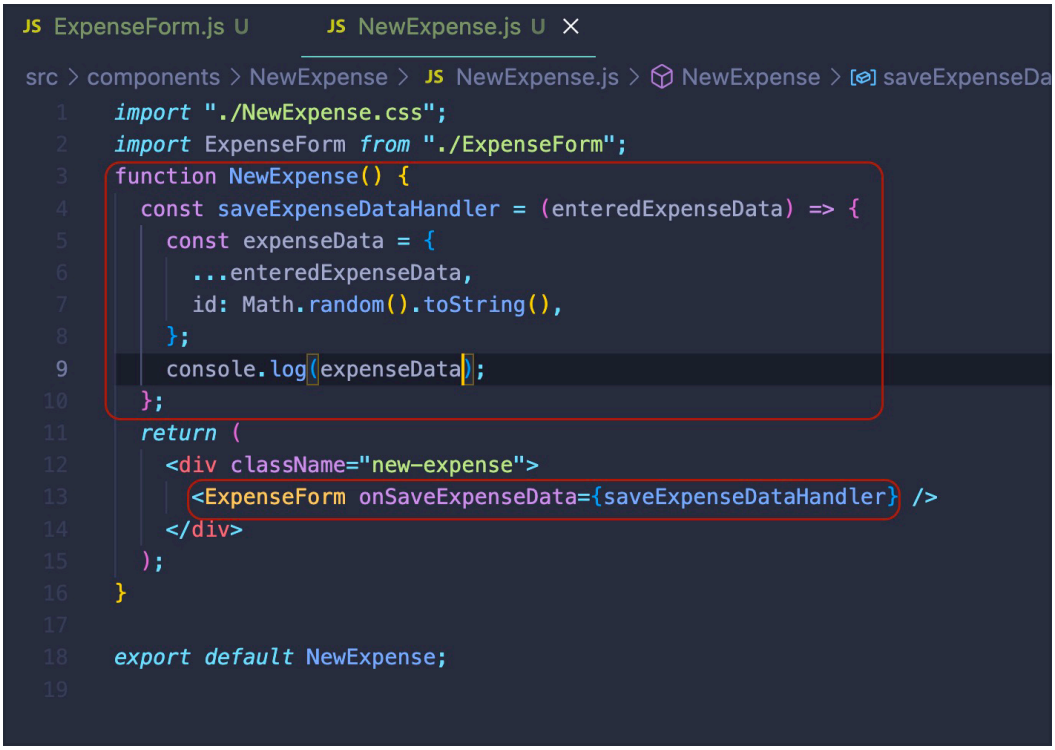
Handling Form Submission:



Adding Two-Way Binding:

With this we can clear the value in the fields when the form is submitted.

Child-to-Parent Component Communication (Bottom-up): Also called “Lifting The State Up”

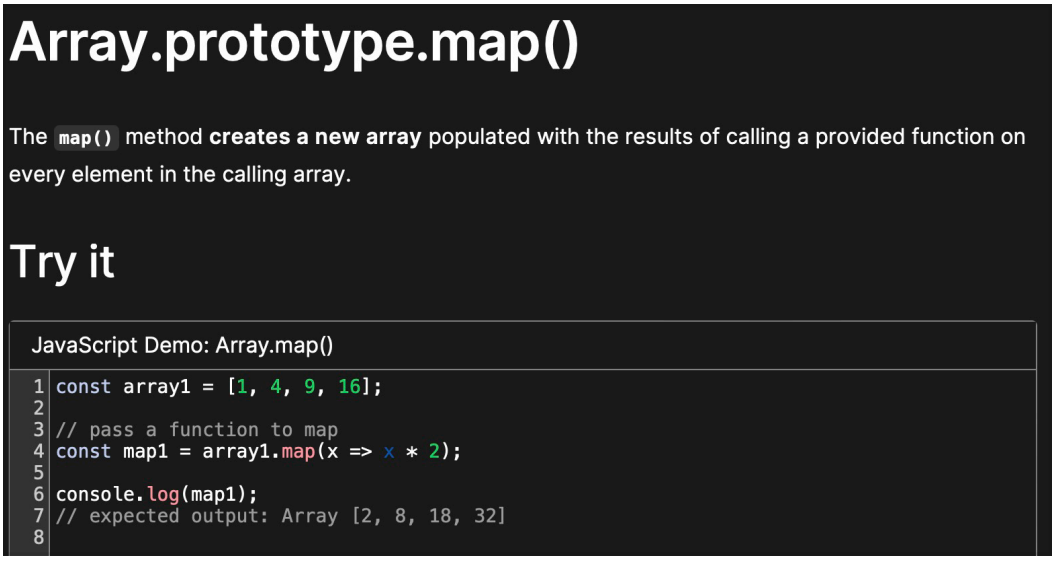

ExpenseForm.js U
NewExpense.js U
src > components > NewExpense > NewExpense.js > NewExpense > saveExpenseDa
import "./NewExpense.css";
import ExpenseForm from "./ExpenseForm";
function NewExpense() {
const saveExpenseDataHandler = (enteredExpenseData) => {
const expenseData = {
...enteredExpenseData,
id: Math.random().toString(),
};
console.log(expenseData);
};
return (

<ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />

);
}
export default NewExpense;

Rendering Lists of Data:

The map function documentation.



Using Stateful Lists:

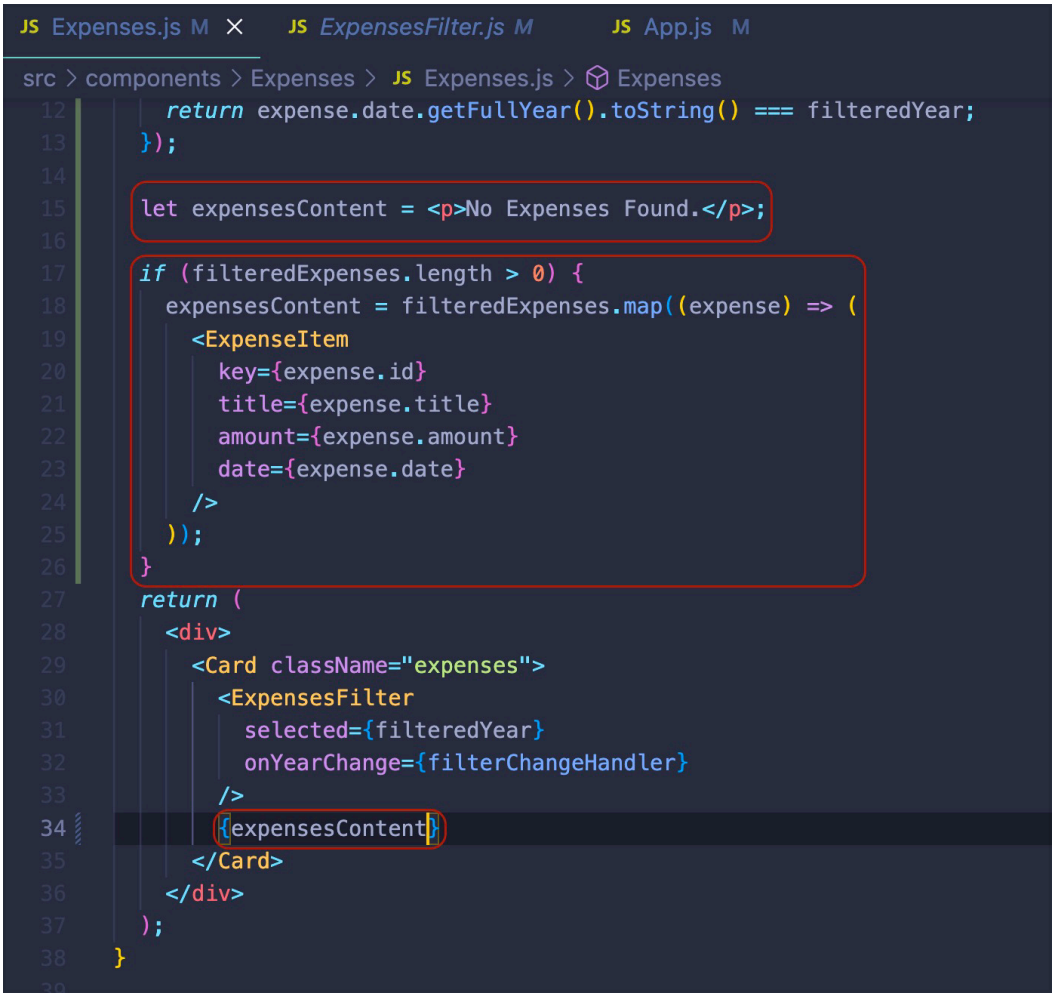
Understanding "Keys":

We start by going to the place which we put out our list.
There we add to the component a special prop named “key” (we can add this prop to any component we build).
By doing this we help React to identify the individual items.
For that we need to provide a unique value for every list item (id).

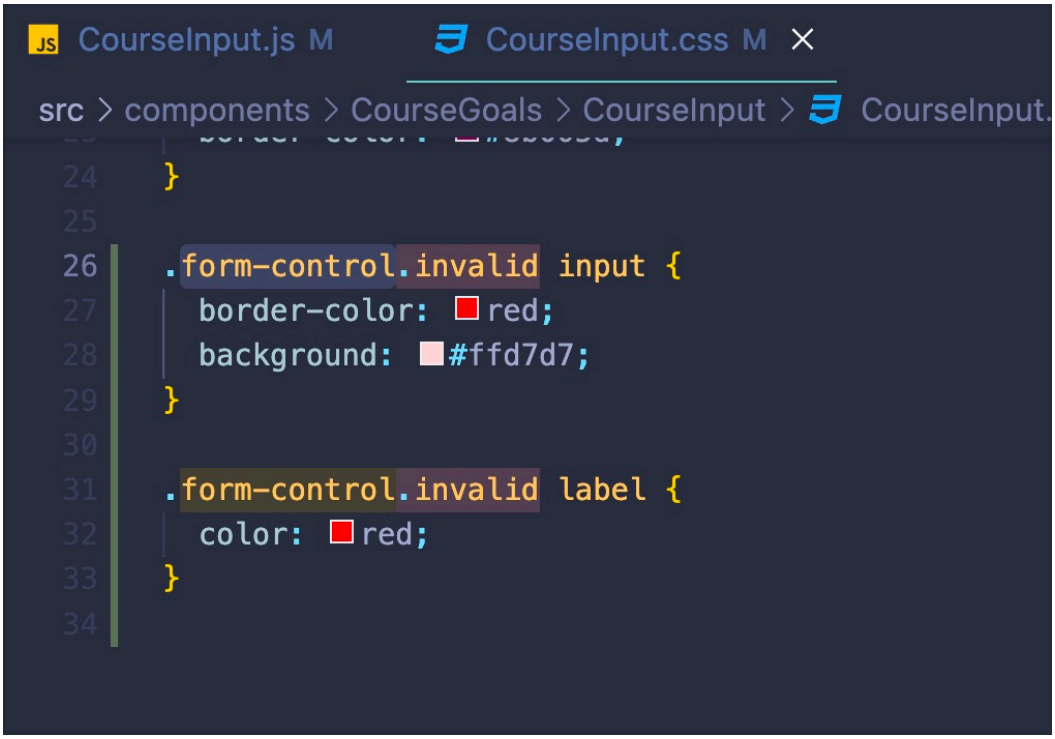
Outputting Conditional Content:

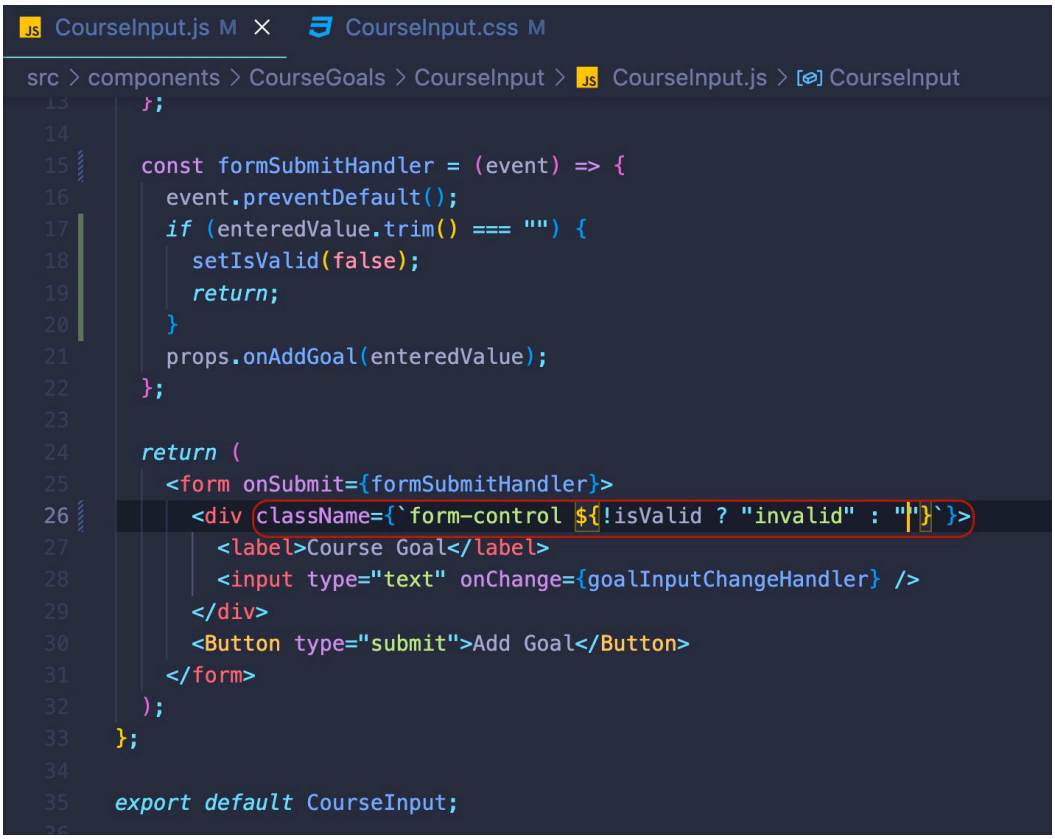
Cannot use a regular if but we can use a ternary expression -> “condition ? True:False”

An alternative way (if we want to avoid ternary expressions) is to use variables like such:



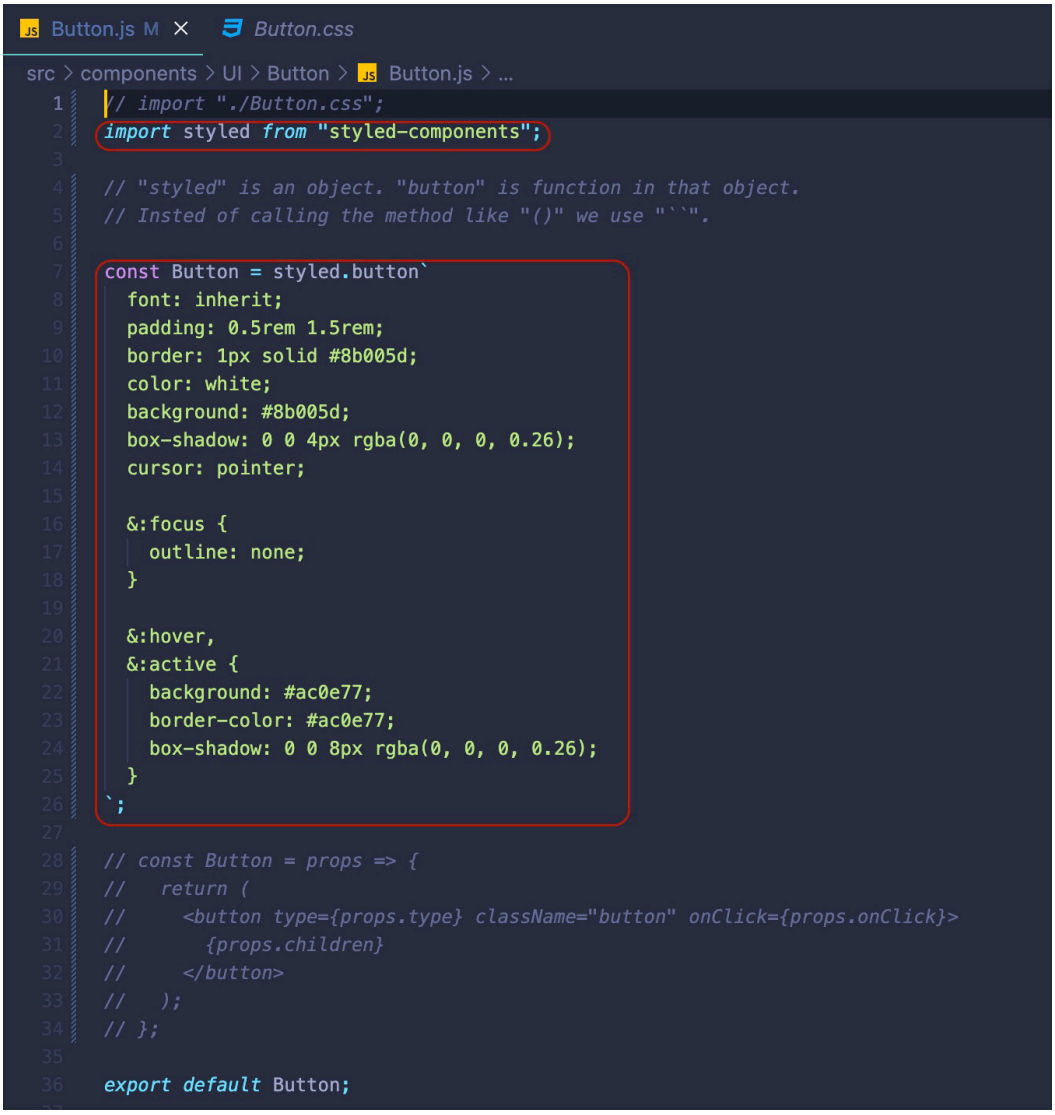
Setting CSS Classes Dynamically:

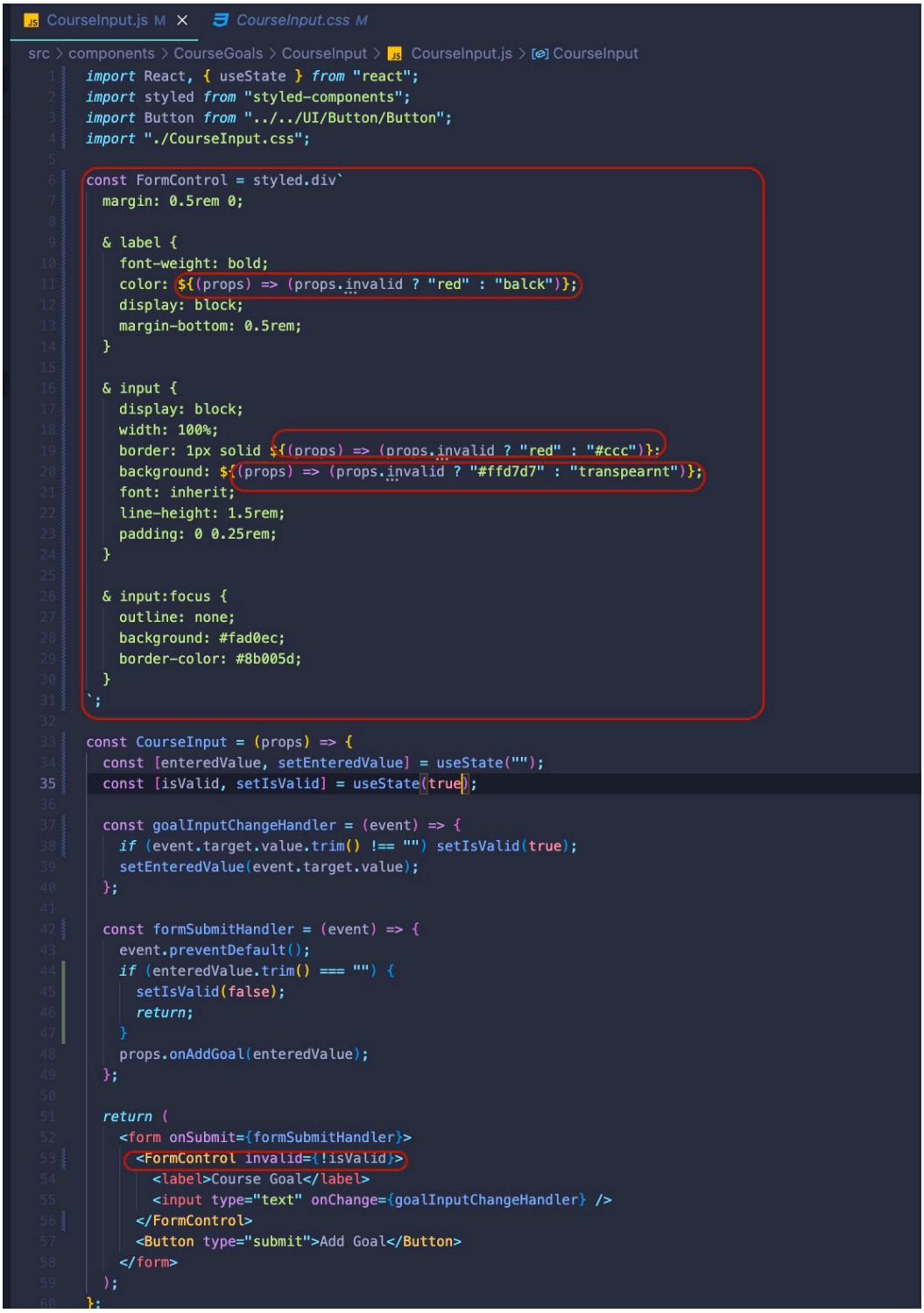




Introducing Styled Components:

In order to use this library we need to install it by running **“npm install --save styled-components”** in our project folder.


Button.js M
Button.css
src > components > UI > Button > Button.js > ...
// import "./Button.css";
import styled from "styled-components";
// "styled" is an object. "button" is function in that object.
// Insted of calling the method like "()" we use "``".
const Button = styled.button`
font: inherit;
padding: 0.5rem 1.5rem;
border: 1px solid #8b005d;
color: white;
background: #8b005d;
box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
cursor: pointer;
&:focus {
outline: none;
}
&:hover,
&:active {
background: #ac0e77;
border-color: #ac0e77;
box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
}
`;
// const Button = props => {
// return (
// <button type={props.type} className="button" onClick={props.onClick}>
// {props.children}
// </button>
//);
// };
export default Button;



Styled Components & Media Queries:

Adding the media to our styles like this:

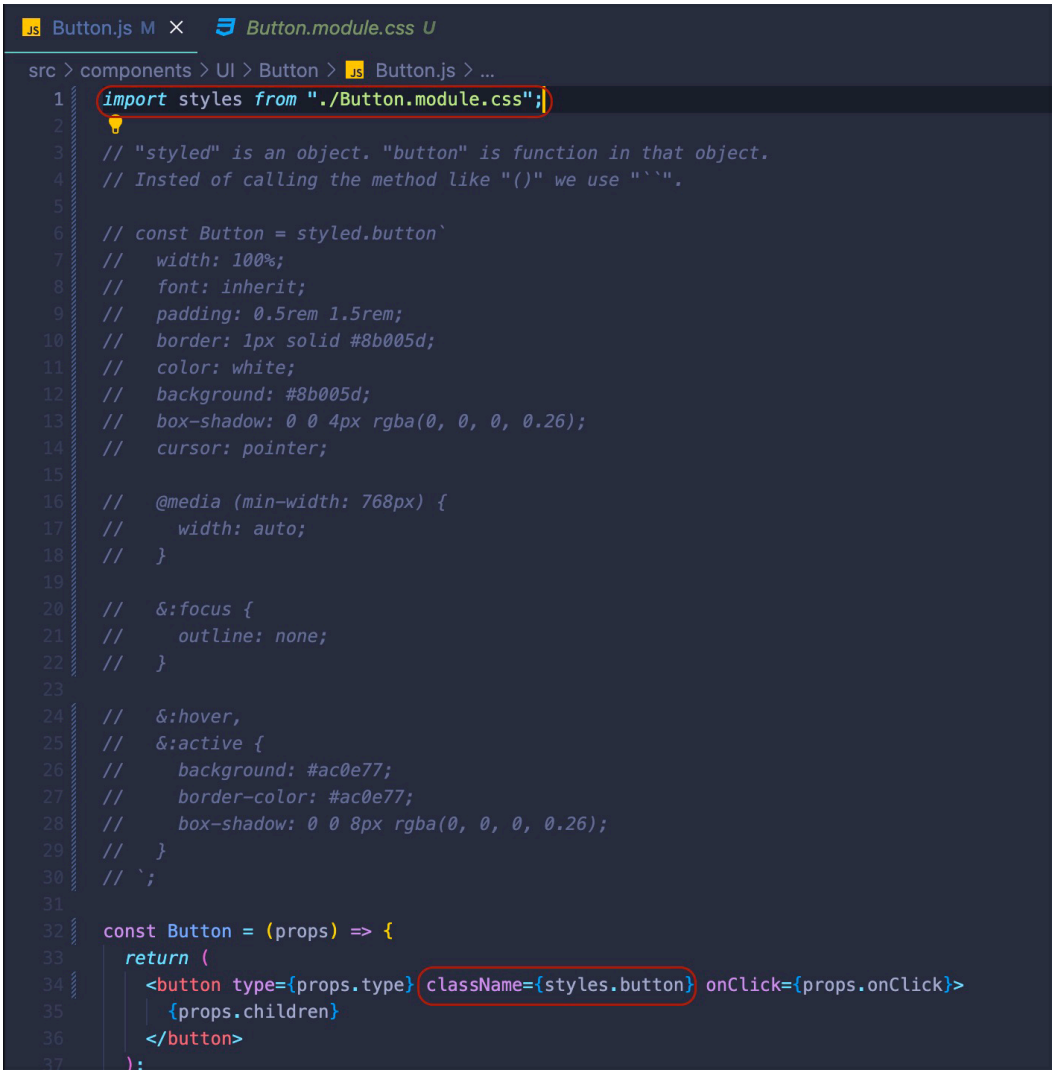
```
@media (min-width: 768px) {
   width: auto;
 }
```

Using CSS Modules:

Starting with renaming the css file we want to use -> “ourFileName.module.css” - we add “.module” to our files.

Than we can import them by -> “Import styles from “./path_to_css_file”.

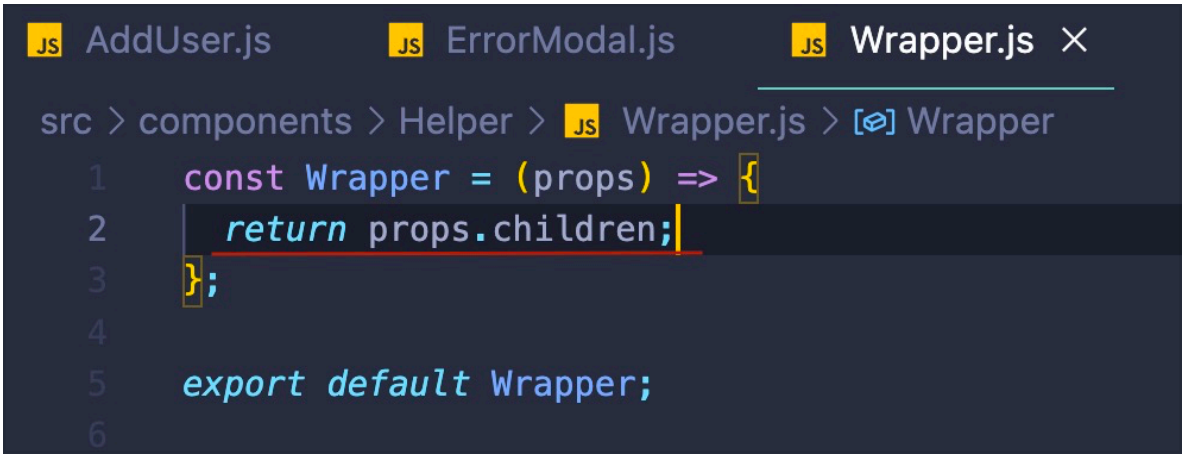
In my opinion it is the best way to style components !




```
  return (
    <form onSubmit={formSubmitHandler}>
      <div
        className={`${styles["form-control"]} ${!isValid && styles.invalid}`}
      >
        <label>Course Goal</label>
        <input type="text" onChange={goalInputChangeHandler} />
      </div>
      <Button type="submit">Add Goal</Button>
    </form>
  );
};

export default CourseInput;
```

Creating a Wrapper Component:

With this approach we just return the content inside that so-called “Wrapper” component. (We end up without div’s soup).



React Fragments:

The wrapper concept is so common that we can simply use "<></>" to wrap our JSX code or we can use "<React.Fragment></React.Fragment>"

```
return (
  <>
    <AddUser onAddUser={addUserHandler}></AddUser>
    <UsersList userList={userList}></UsersList>
  </>
);
}
```

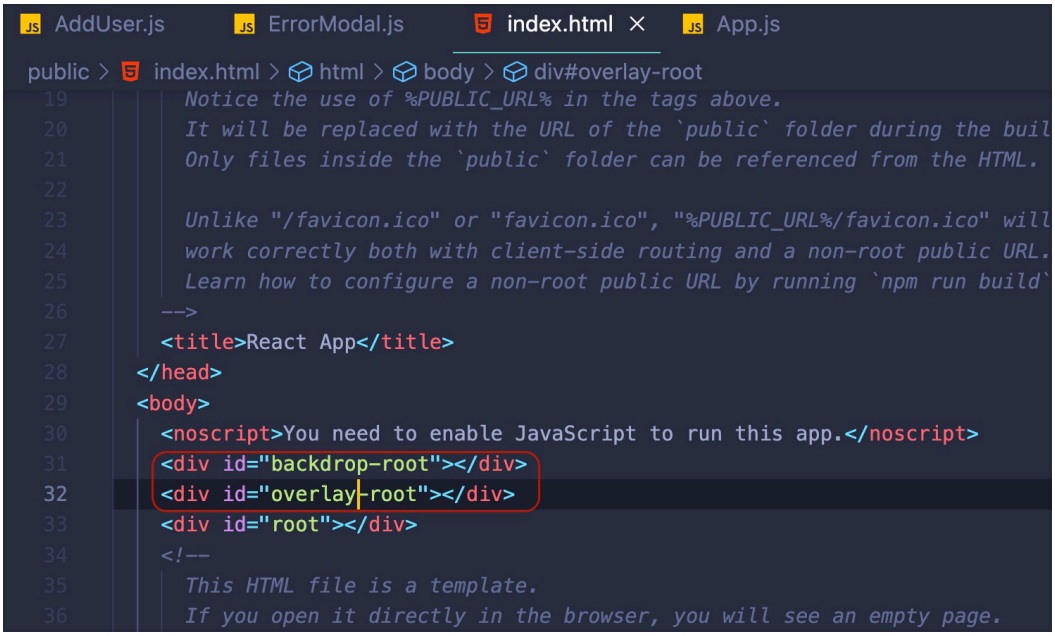
```
return (
  <React.Fragment>
    <AddUser onAddUser={addUserHandler}></AddUser>
    <UsersList userList={userList}></UsersList>
  </React.Fragment>
);
}
```

Introducing React Portals:

With portals we need two thing:

1. Place to "port" the component to.
2. Let the component know that it should have a portal to that place.

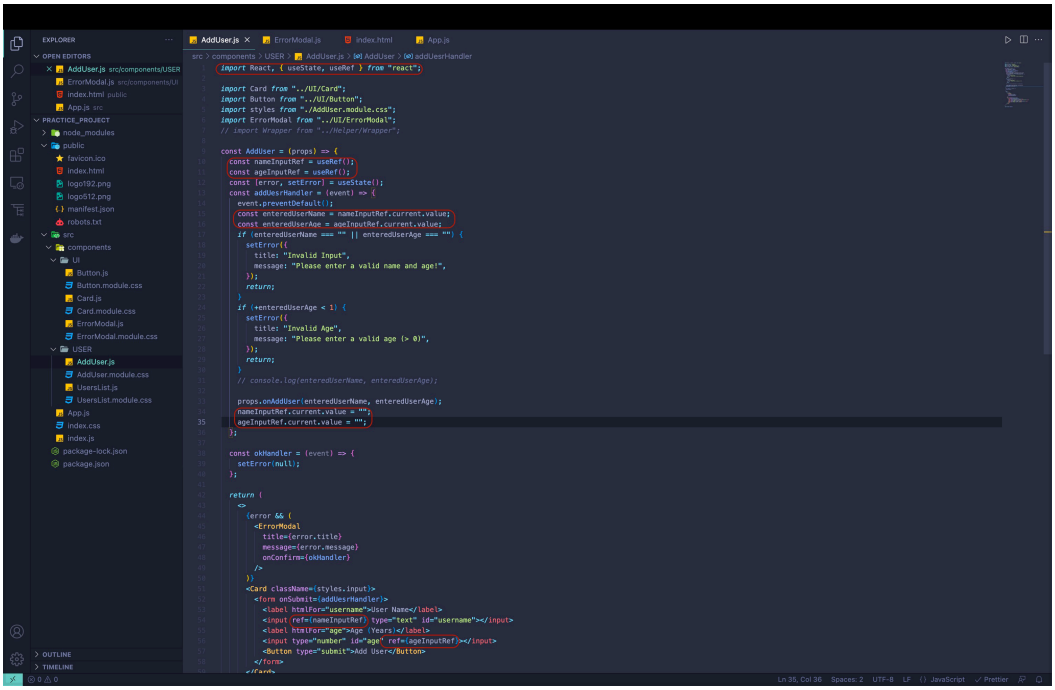
We use the id to identify the place we want the component to "go".



Then we use a special function (from react-dom) - **ReactDOM.createPortal(The_component_we_want, the_place_which_the_component_go)**

Working with "ref"s:

With we setup a connection between html element and a js code.



refs

What are "Side Effects" & Introducing useEffect:

Store data in browser storage, sending http requests to backend server, set and menage timers and more...

These tasks must happen outside of the normal component evaluation and render cycle - especially since they might block/delay rendering (e.g. Http requests).

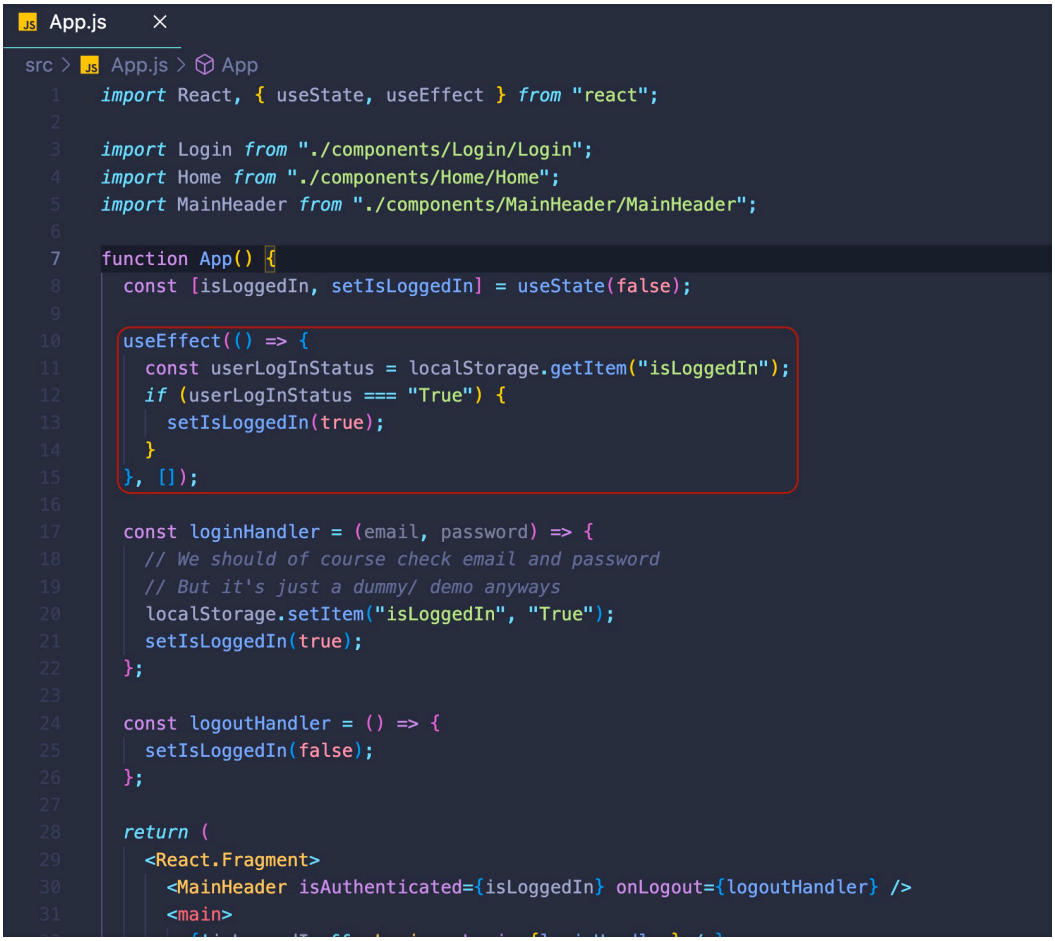
Using the useEffect() Hook:

The useEffect Hook function gets two parameters useEffect(function_to_excute, array_of_dependencies).

The use effect function only called in case the dependencies changed.

In this example the useEffect function only called when the App is first called than the dependencies not changing (empty array).

Therefor we are not in infinite loop. (useEffect than useState every time).

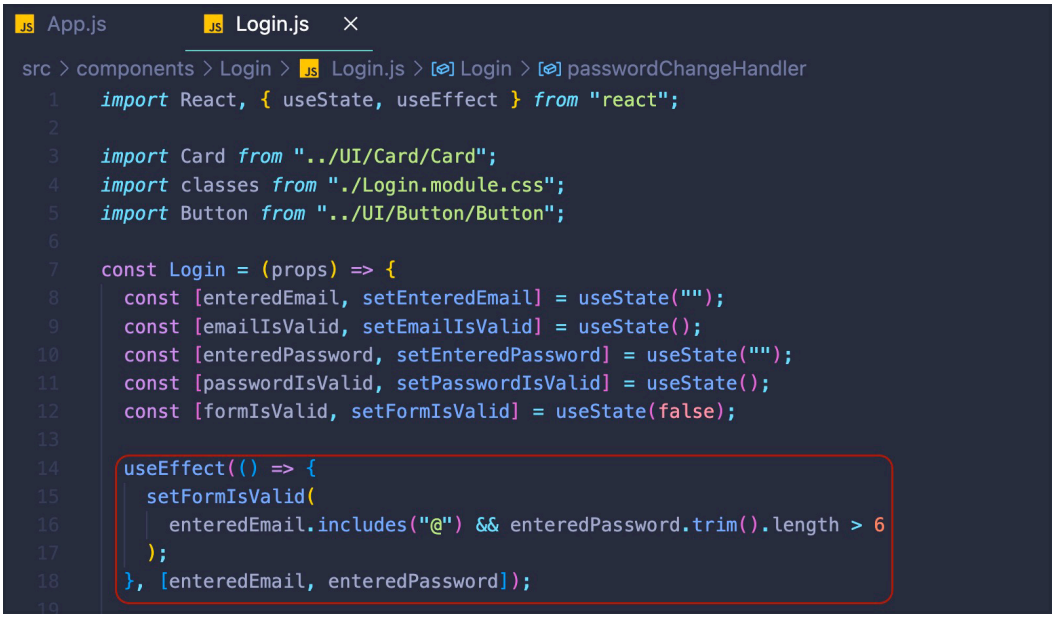


useEffect & Dependencies:

In this example the useEffect function will only run whenever the "enteredEmail" or the "enteredPassword"
Value is changed.

useEffect helps you deal with code that should be executed in response to something!

Whenever we have a an action that should be executed in response to other action we should use useEffect!



Using the useEffect Cleanup Function:

introducing useReducer & Reducers In General:

We want to use "useReducer" in cases where we have states that deepens on one another such as email_value state and email_valid state.
The email_valid state defense on the email_value state.

Using the useReducer() Hook:

Login.js

src > components > Login > Login.js > Login

```jsx
import React, { useState, useEffect, useReducer } from "react";

import Card from "../UI/Card/Card";
import classes from "./Login.module.css";
import Button from "../UI/Button/Button";

const emailReducer = (state, action) => {
  if (action.type === "USER_INPUT") {
    return { value: action.val, isValid: action.val.includes("@") };
  }
  if (action.type === "INPUT_BLUR") {
    return { value: state.value, isValid: state.value.includes("@") };
  }
  return { value: "", isValid: false };
};

const Login = (props) => {
  // const [enteredEmail, setEnteredEmail] = useState("");
  // const [emailIsValid, setEmailIsValid] = useState();
  const [enteredPassword, setEnteredPassword] = useState("");
  const [passwordIsValid, setPasswordIsValid] = useState();
  const [formIsValid, setFormIsValid] = useState(false);
  const [emailState, dispatchEmail] = useReducer(emailReducer, {
    value: "",
    isValid: null,
  });
  // useEffect(() => {…

  const emailChangeHandler = (event) => {
    dispatchEmail({ type: "USER_INPUT", val: event.target.value });

    setFormIsValid(emailState.isValid && enteredPassword.trim().length > 6);
  };

  const passwordChangeHandler = (event) => {
    setEnteredPassword(event.target.value);

    setFormIsValid(emailState.isValid && event.target.value.trim().length > 6);
  };

  const validateEmailHandler = () => {
    dispatchEmail({ type: "INPUT_BLUR" });
  };

  const validatePasswordHandler = () => {
    setPasswordIsValid(enteredPassword.trim().length > 6);
  };

  const submitHandler = (event) => {
```

Using the React Context API:

Creating a context object

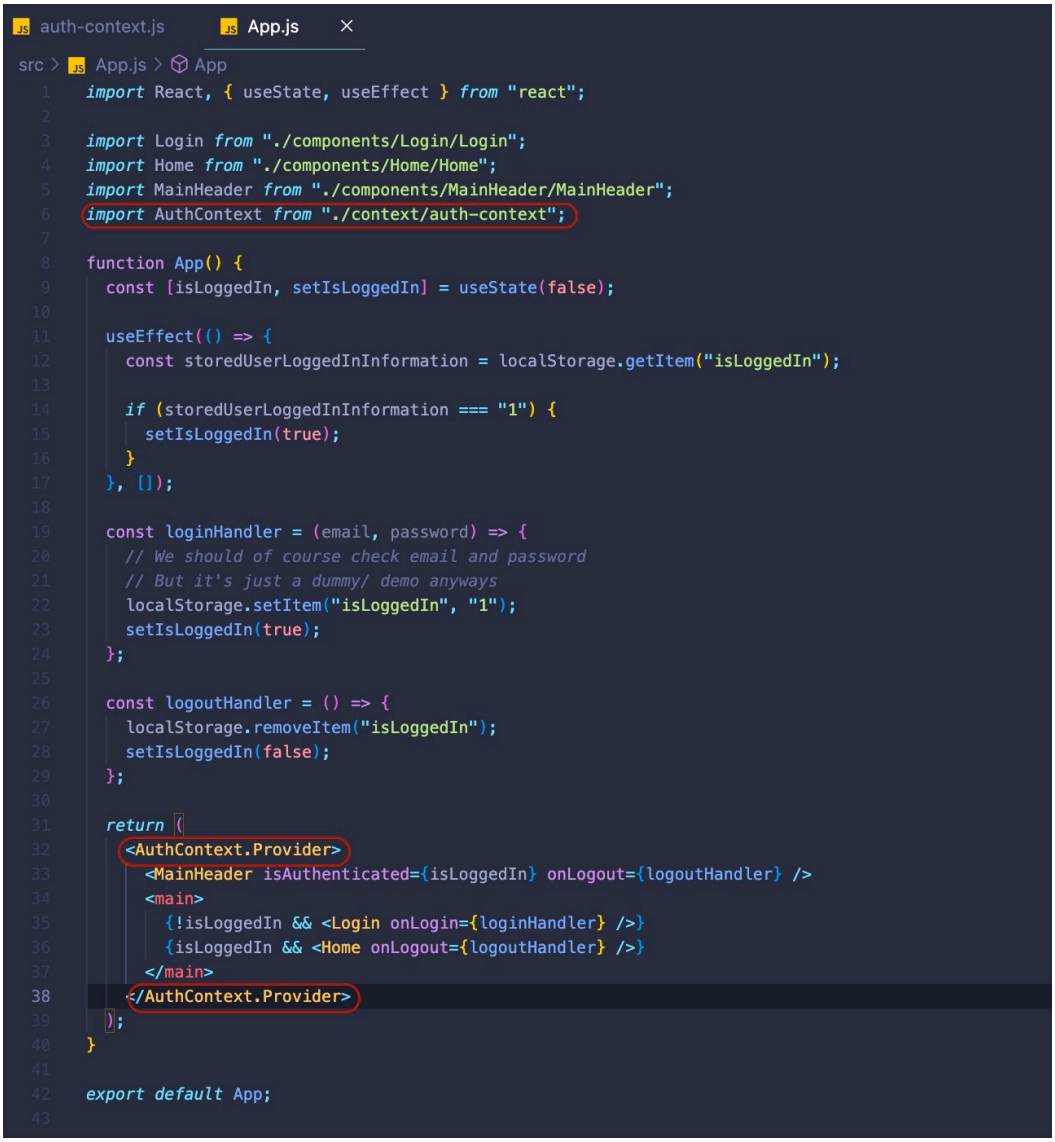
auth-context.js

src > context > auth-context.js > ...

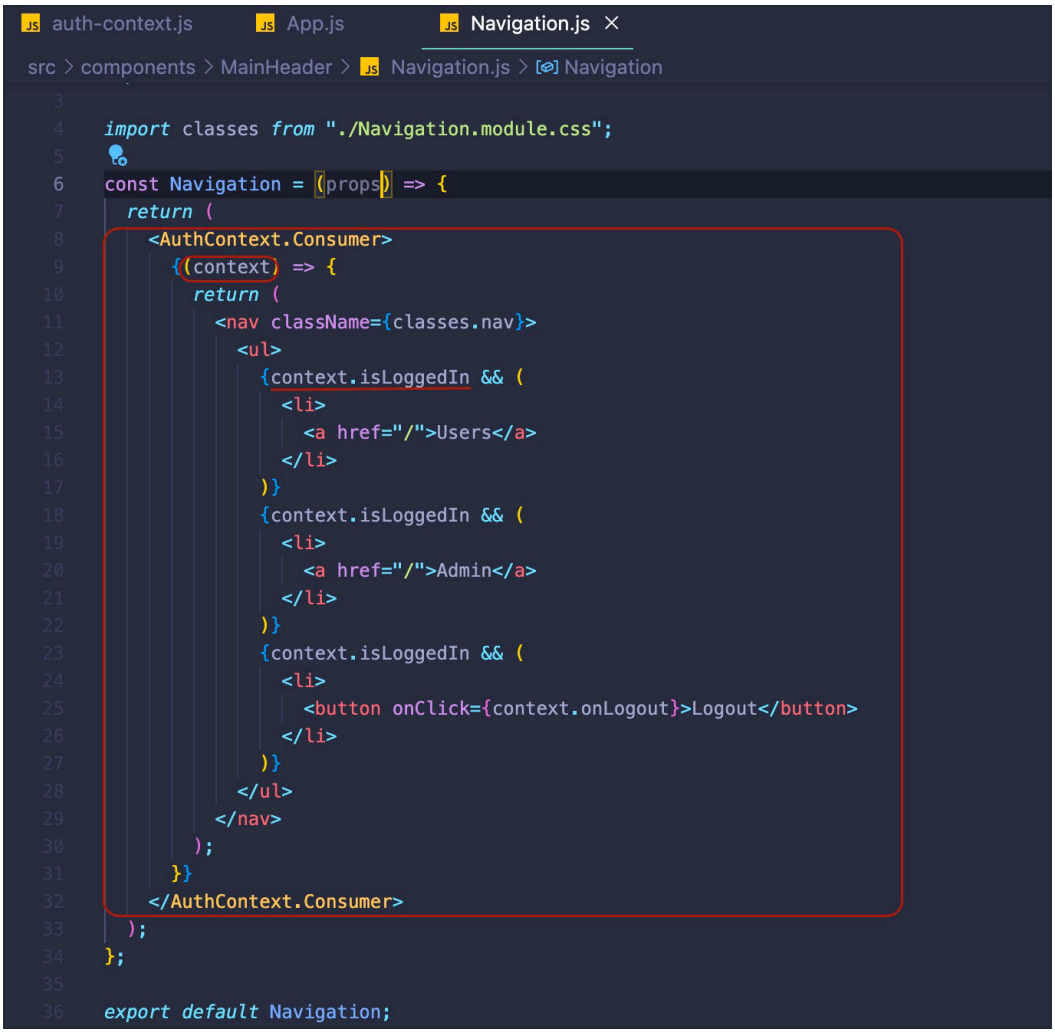
```jsx
const AuthContext = React.createContext({
  isLoggedIn: false,
});
export default AuthContext;
```

Every component that is “nested” inside our “Context object” will have access to data we set in our context object.
Also, every component child of the components that are “nested” inside our “Context object” will have access to data we set in our context object and there

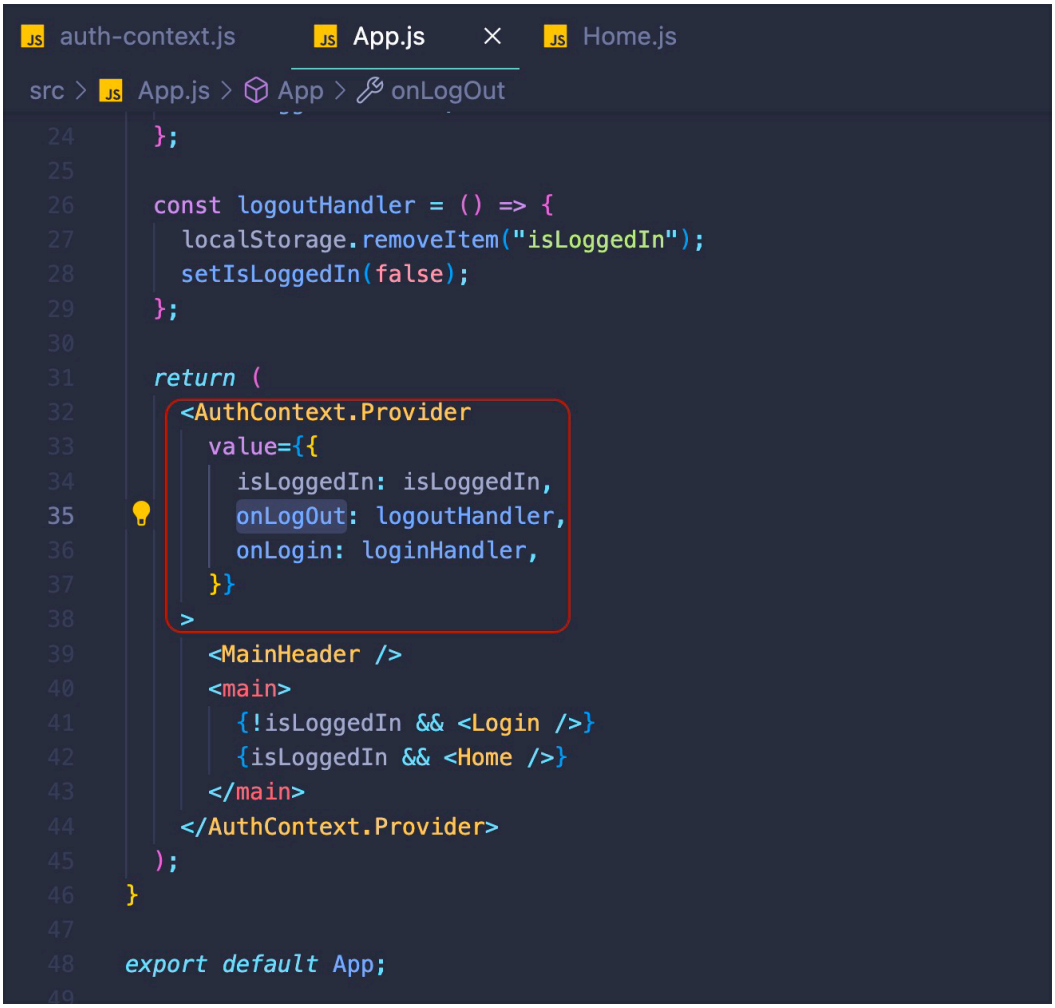
children components too.

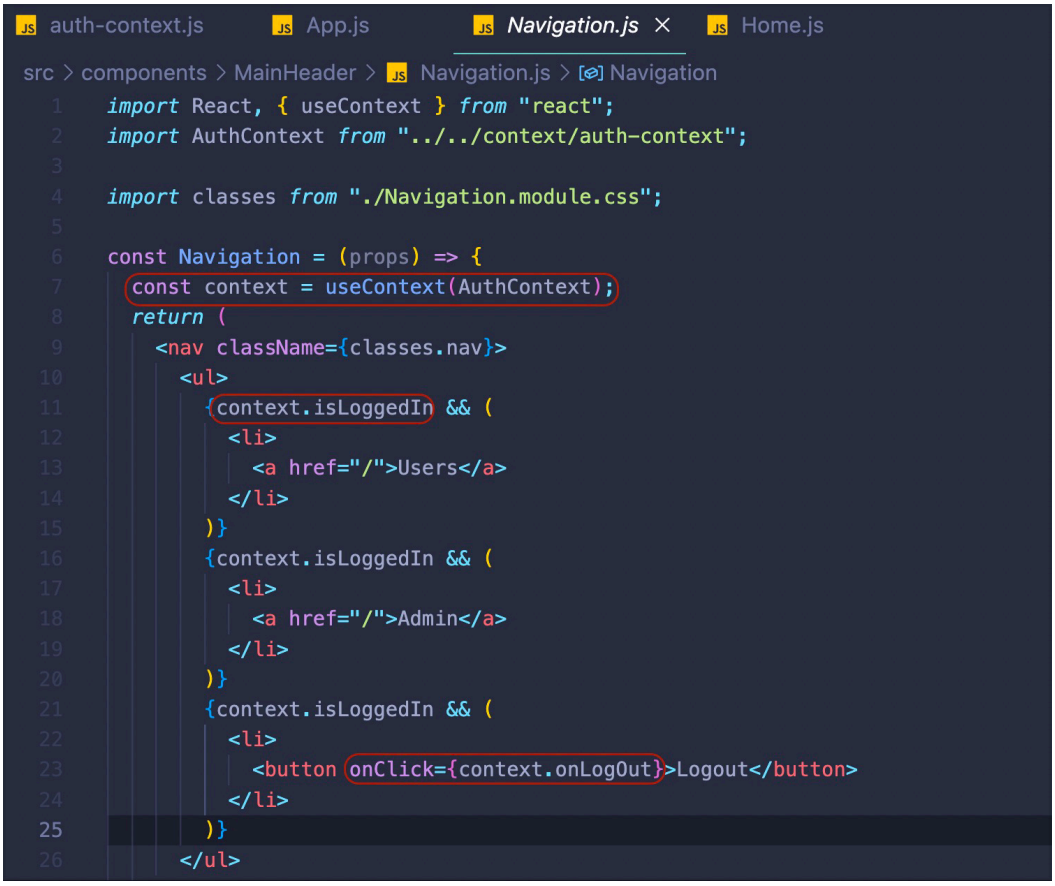


One approach to get the data and use it in other components is as follow (Using the Consumer):



The more elgant solution is to use the “useContext” Hook as follows:


auth-context.js
App.js
Home.js
src > App.js > App > onLogOut
};
const logoutHandler = () => {
localStorage.removeItem("isLoggedIn");
setIsLoggedIn(false);
};
return (
<AuthContext.Provider
value={{
isLoggedIn: isLoggedIn,
onLogOut: logoutHandler,
onLogin: loginHandler,
}}
>
<MainHeader />
<main>
{!isLoggedIn && <Login />}
{isLoggedIn && <Home />}
</main>
</AuthContext.Provider>
);
}
export default App;



Building & Using a Custom Context Provider Component:

auth-context.js | App.js | index.js | Home.js

src > context > auth-context.js > ...

```jsx
import React, { useState, useEffect } from "react";
const AuthContext = React.createContext({
  isLoggedIn: false,
  onLogOut: null,
  onLogin: (email, password) => {},
});

export const AuthContextProvider = (props) => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  useEffect(() => {
    const storedUserLoggedInInformation = localStorage.getItem("isLoggedIn");

    if (storedUserLoggedInInformation === "1") {
      setIsLoggedIn(true);
    }
  }, []);

  const logOutHandler = () => {
    localStorage.removeItem("isLoggedIn");
    setIsLoggedIn(false);
  };
  const logInHandler = () => {
    localStorage.setItem("isLoggedIn", "1");
    setIsLoggedIn(true);
  };

  return (
    <AuthContext.Provider
      value={{
        isLoggedIn: false,
        onLogOut: logOutHandler,
        onLogin: logInHandler,
      }}
    >
      {props.children}
    </AuthContext.Provider>
  );
};
export default AuthContext;
```

auth-context.js | App.js | index.js | Home.js

src > index.js > ...

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

import "./index.css";
import App from "./App";
import { AuthContextProvider } from "./context/auth-context";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <AuthContextProvider>
    <App />
  </AuthContextProvider>
);
```

auth-context.js App.js index.js Home.js

src > App.js > ...

```
import React, { useContext } from "react";

import Login from "./components/Login/Login";
import Home from "./components/Home/Home";
import MainHeader from "./components/MainHeader/MainHeader";
import AuthContext from "./context/auth-context";

function App() {
  const context = useContext(AuthContext);
  return (
    <React.Fragment>
      <MainHeader />
      <main>
        {!context.isLoggedIn && <Login />}
        {context.isLoggedIn && <Home />}
      </main>
    </React.Fragment>
  );
}

export default App;
```

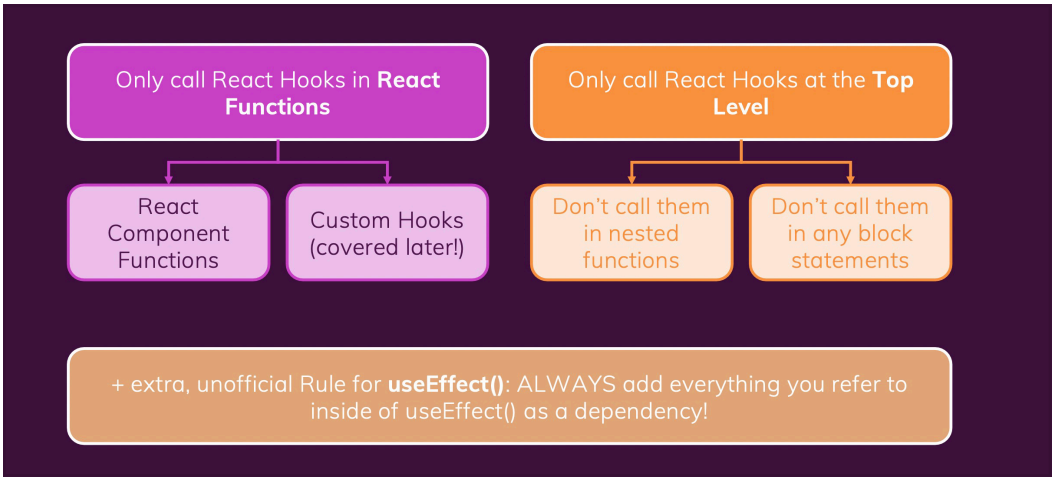
Context Disadvantages:

Only consider using this approach in cases where we have “props chains”.
If we have some value that constantly changes his state then “useContext” is not recommended because “useContext” is not optimized for this scenarios.

Learning the "Rules of Hooks”:

We can call react hooks (such as “useState”, “useReducer”, “useEffect” and more..) only in two places:

1. React Component.
2. Custom Hook.

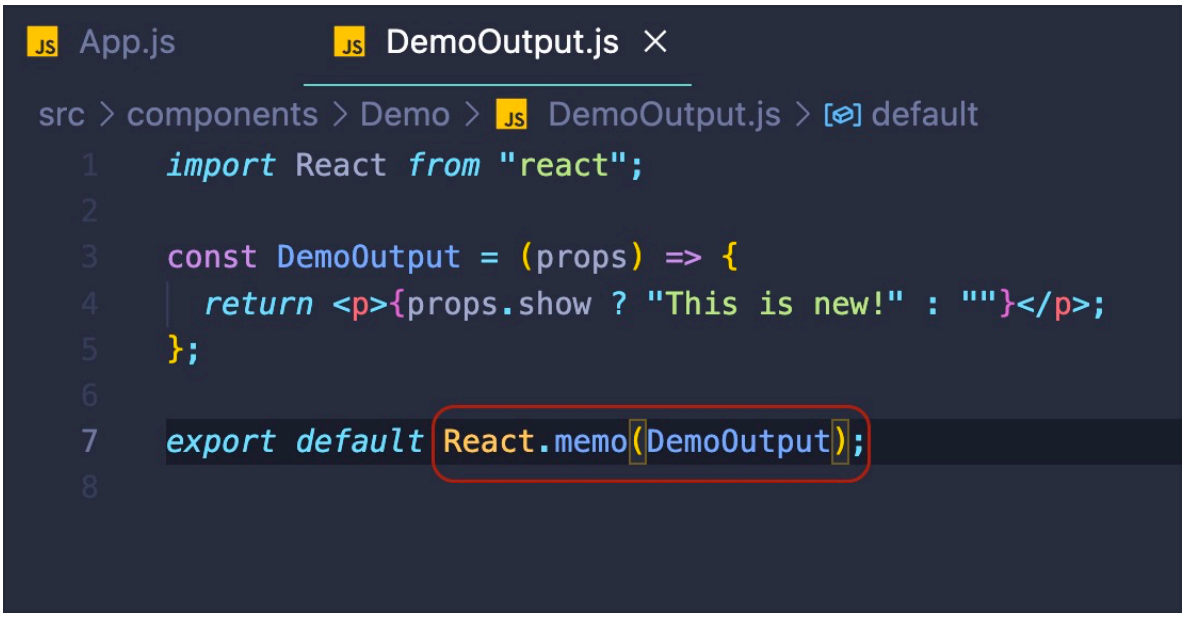


Diving into "Forward Refs”:

Very useful when working with inputs.
In the course we saw an example at section 10 lecture number 129.
The example showed us how we can focus our cursor on input fields in cases where the user pressed “Login” but one of the input fields are

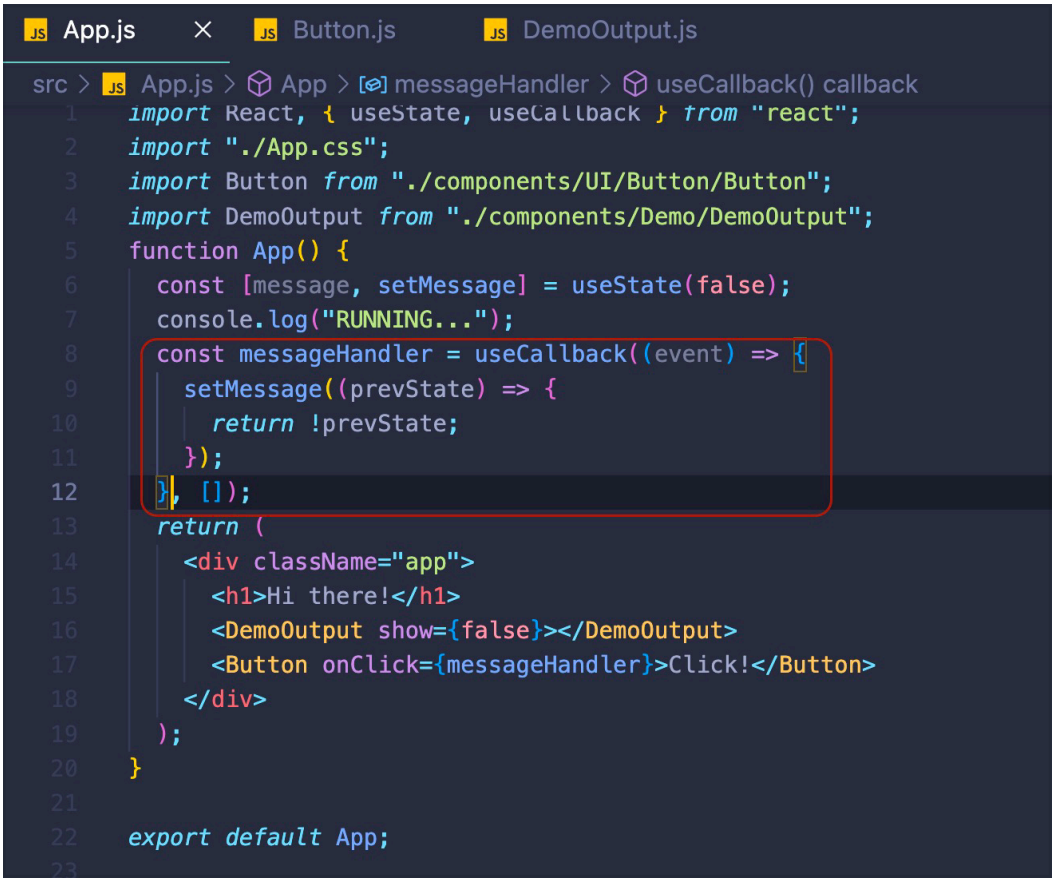
Not valid. Using “useImperativeHandle” and “React.forwardRef”.

Preventing Unnecessary Re-Evaluations with React.memo():
Start by going to the component that we want to tell react ”when” to reevaluate.
The we use “React.memo()” like such:



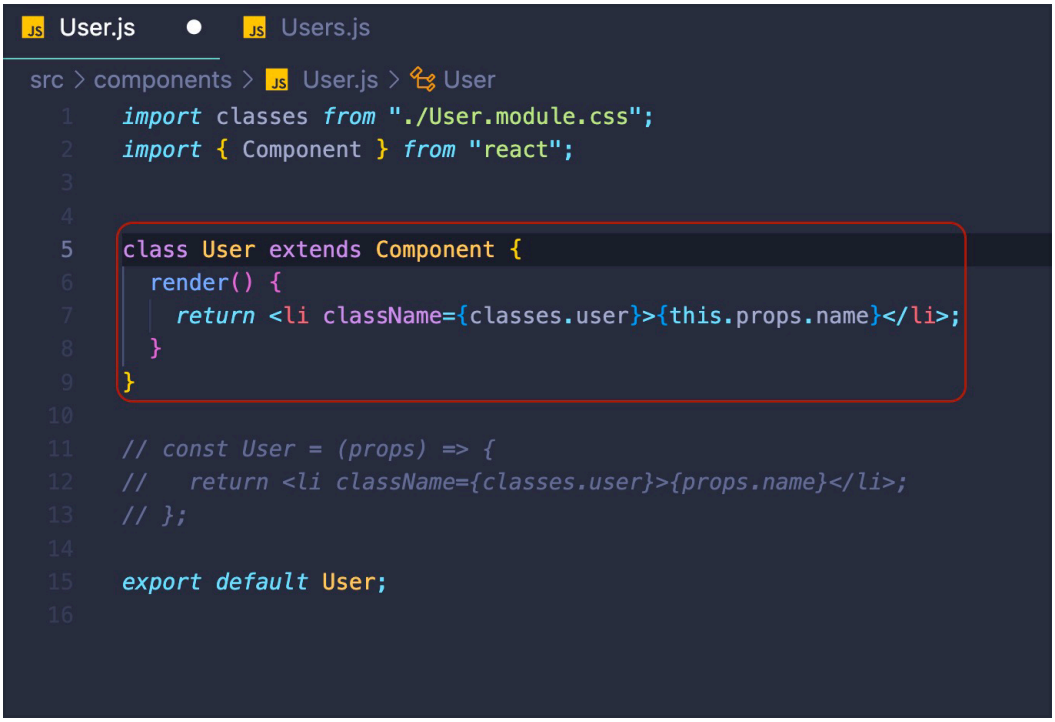
React.memo() - tells react to check (when the component need to reevaluate) the props and if there is a change between the “old” props and the “new” Props then reevaluate this component.
Important to notice is that if the component is not reevaluated then all his “child” components are not evaluated.
Also it checks for value and not reference.

Preventing Function Re-Creation with useCallback():
useCallBck hook will save the object/function in the same place un the memory therefor when it will need to reevaluate
Component he can figure out that this is the same object/function there was no change.
We use it as such:



Class Based Components:

In the class that represent the component we create we need to create a function called “render” and in this function
Return a JSX.
Example:

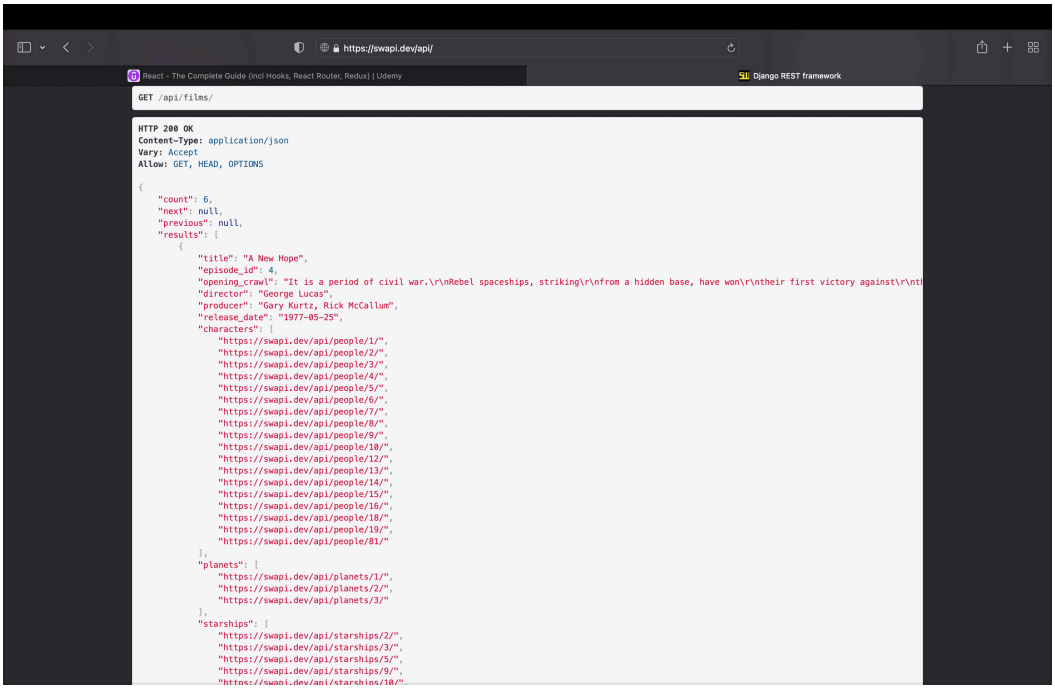


Managing state in class based components:

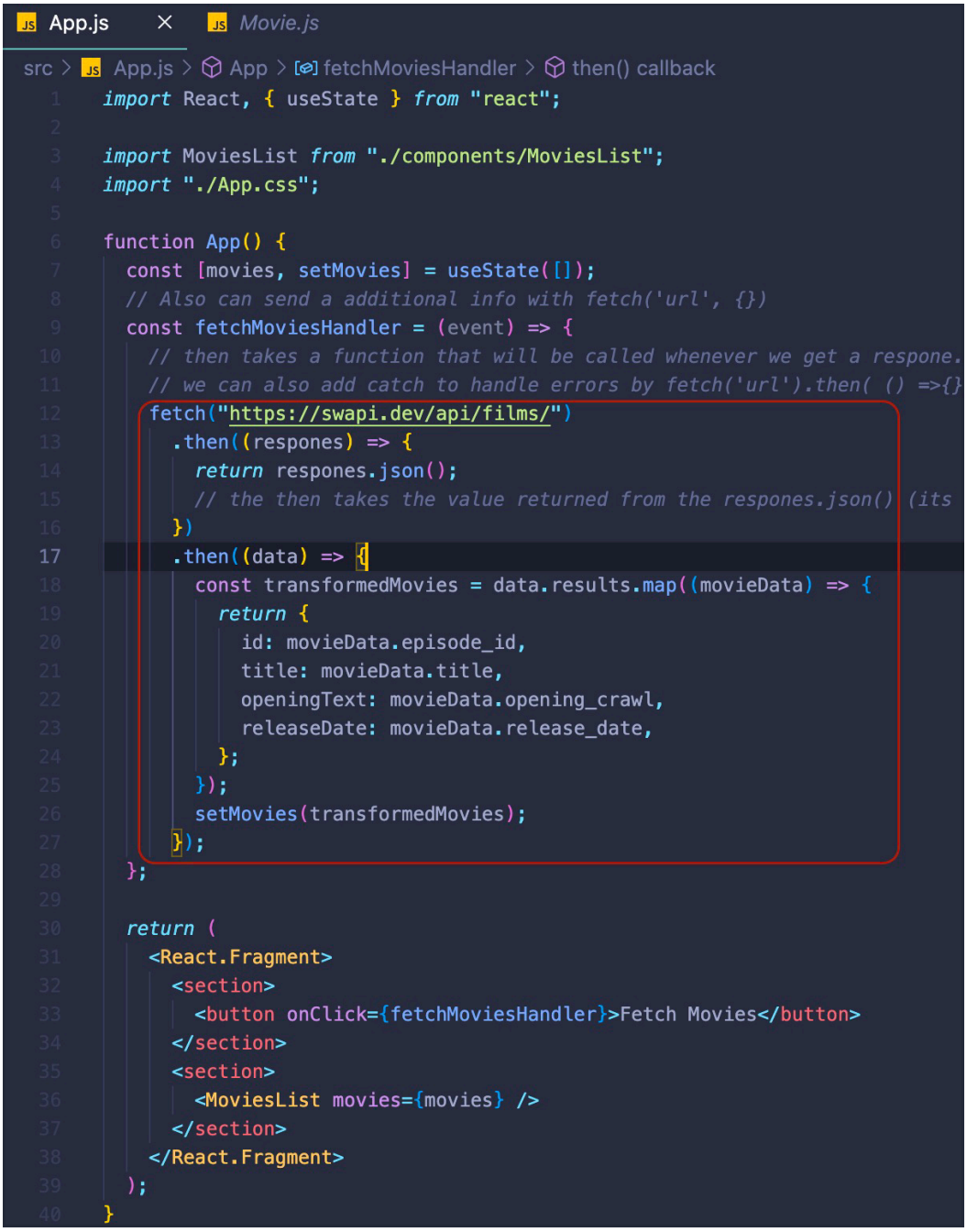
Section 14: Sending Http Requests (e.g. Connecting to a Database):

Sending a GET Request:

The look of the data of the request to know which fields we can access

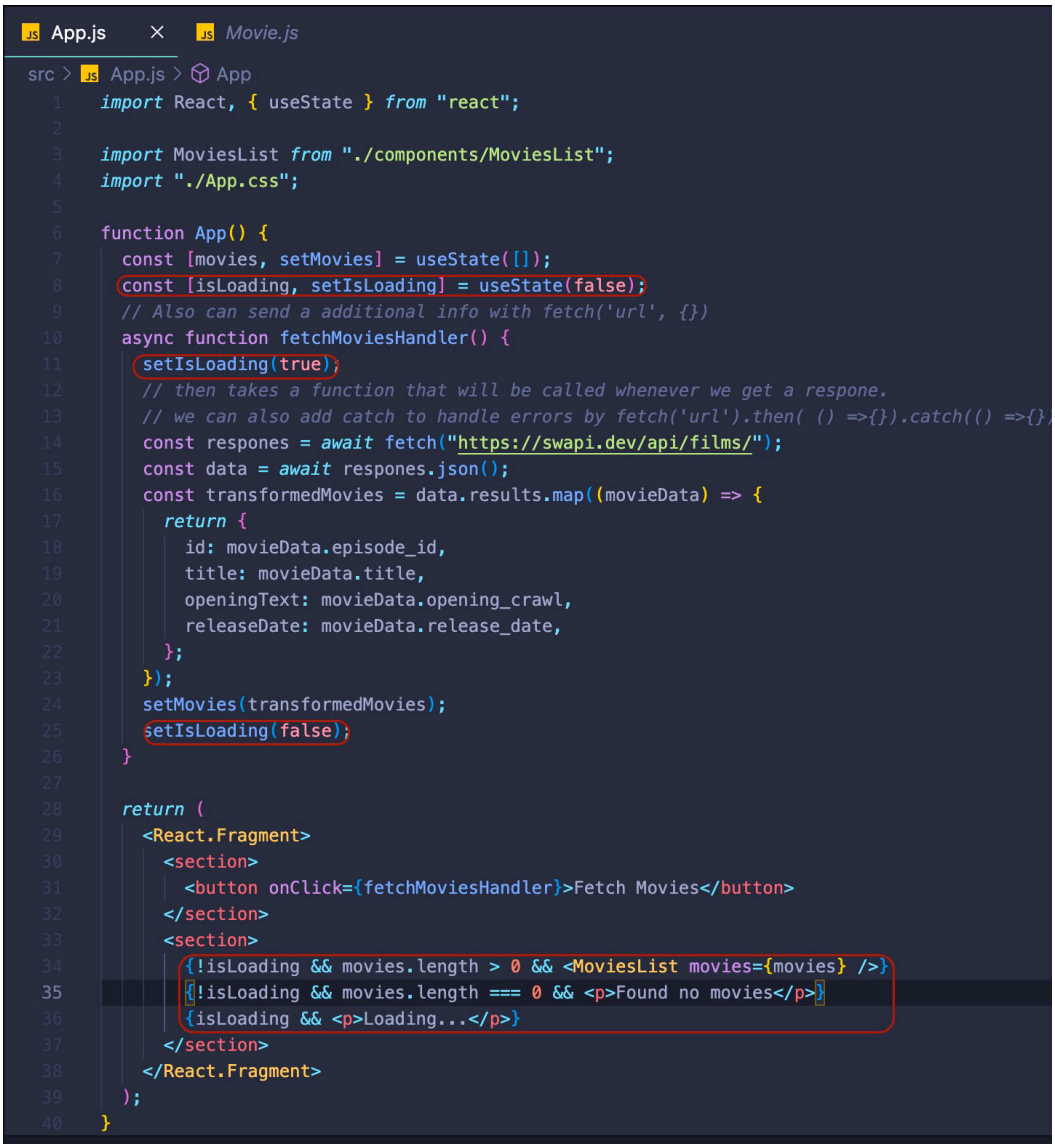


How to send http request from a react app to a backend



Using async / await:

Handling Loading & Data States:



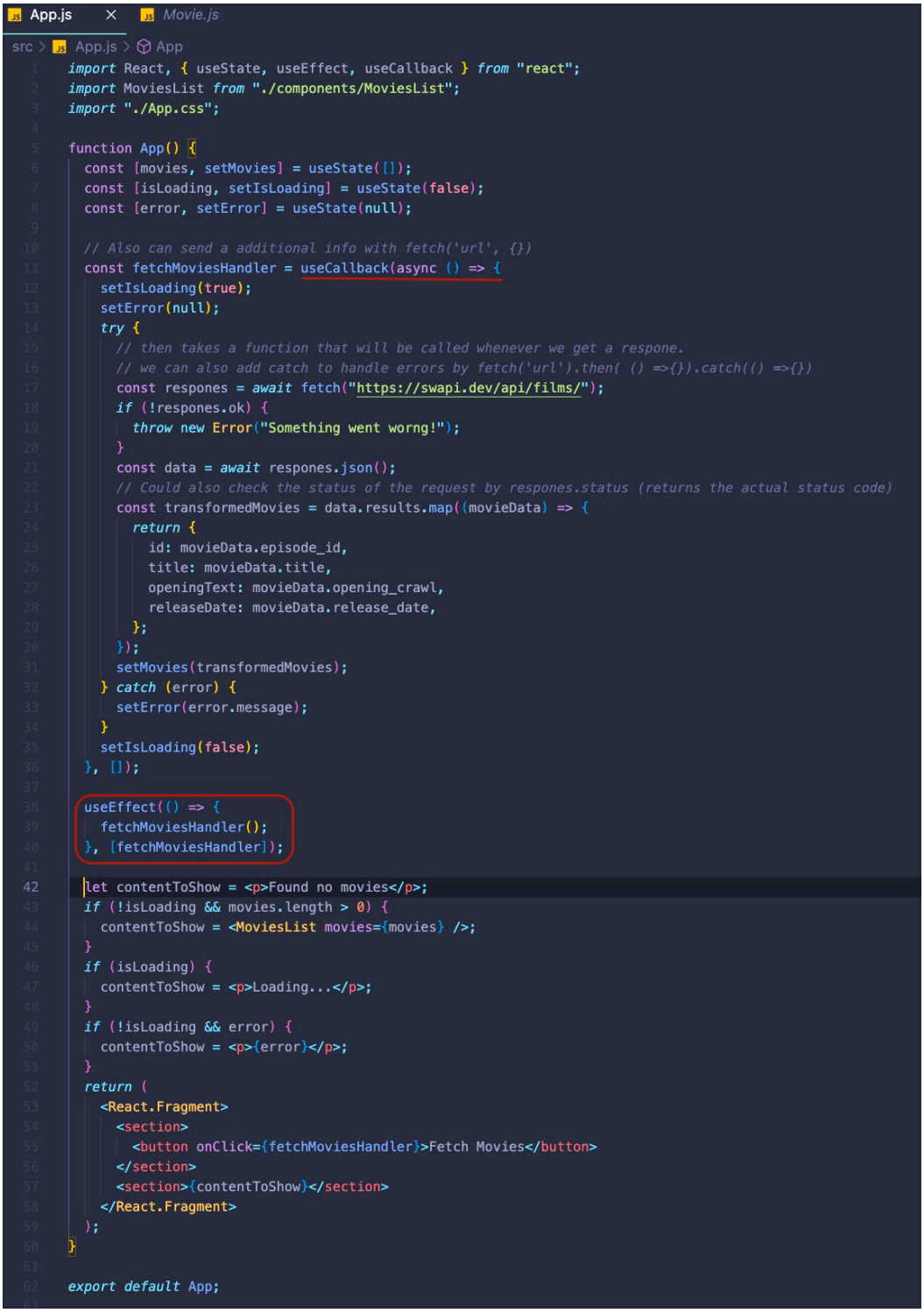
Handling Http Errors:

App.js × Movie.js

src > App.js > ...

```
function App() {
  const [movies, setMovies] = useState([]);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);
  // Also can send a additional info with fetch('url', {})
  async function fetchMoviesHandler() {
    setIsLoading(true);
    setError(null);
    try {
      // then takes a function that will be called whenever we get a respone.
      // we can also add catch to handle errors by fetch('url').then( () =>{}).catch(() =>{})
      const respones = await fetch("https://swapi.dev/api/fi/lms/");
      if (!respones.ok) {
        throw new Error("Something went worng!");
      }
      const data = await respones.json();
      // Could also check the status of the request by respones.status (returns the actual status code)
      const transformedMovies = data.results.map((movieData) => {
        return {
          id: movieData.episode_id,
          title: movieData.title,
          openingText: movieData.opening_crawl,
          releaseDate: movieData.release_date,
        };
      });
      setMovies(transformedMovies);
    } catch (error) {
      setError(error.message);
    }
    setIsLoading(false);
  }

  return (
    <React.Fragment>
      <section>
        <button onClick={fetchMoviesHandler}>Fetch Movies</button>
      </section>
      <section>
        {!isLoading && movies.length > 0 && <MoviesList movies={movies} />}
        {!isLoading && movies.length === 0 && <p>Found no movies</p>}
        {isLoading && <p>Loading...</p>}
        {!isLoading && error && <p>{error}</p>}
      </section>
    </React.Fragment>
  );
}

export default App;
```

Using useEffect() For Requests:



Sending a POST Request:

```
async function addMovieHandler(movie) {
  const response = await fetch(
    "https://react-udemy-course-http-71ec4-default-rtdb.firebaseio.com/movies.json",
    {
      method: "POST",
      body: JSON.stringify(movie), // takes a javaScript object/Array and trsform it to JSON format
      headers: {
        "Content-Type": "application/json",
      },
    }
  );
  setIsNewMovie(true);
  // const data = response.json();
  // fetchMoviesHandler();
}
```

Section 15: Building Custom React Hooks:

Creating a Custom React Hook Function:

We have to start the name of the function with “use”.

App.js | ForwardCounter.js | BackwardCounter.js | use-counter.js

src > hooks > use-counter.js > useCounter

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

const useCounter = (initVal, counterStep) => {
  const [counter, setCounter] = useState(initVal);

  useEffect(() => {
    const interval = setInterval(() => {
      setCounter((prevCounter) => prevCounter + counterStep);
    }, 1000);

    return () => clearInterval(interval);
  }, [counterStep]);
  return counter;
};

export default useCounter;
```

App.js | ForwardCounter.js | use-counter.js

src > components > ForwardCounter.js > ForwardCounter > coun

```
import useCounter from "../hooks/use-counter";

import Card from "./Card";

const ForwardCounter = () => {
  const counter = useCounter(0, 1);

  return <Card>{counter}</Card>;
};

export default ForwardCounter;
```

BackwardCounter.js

src > components > BackwardCounter.js > BackwardCour

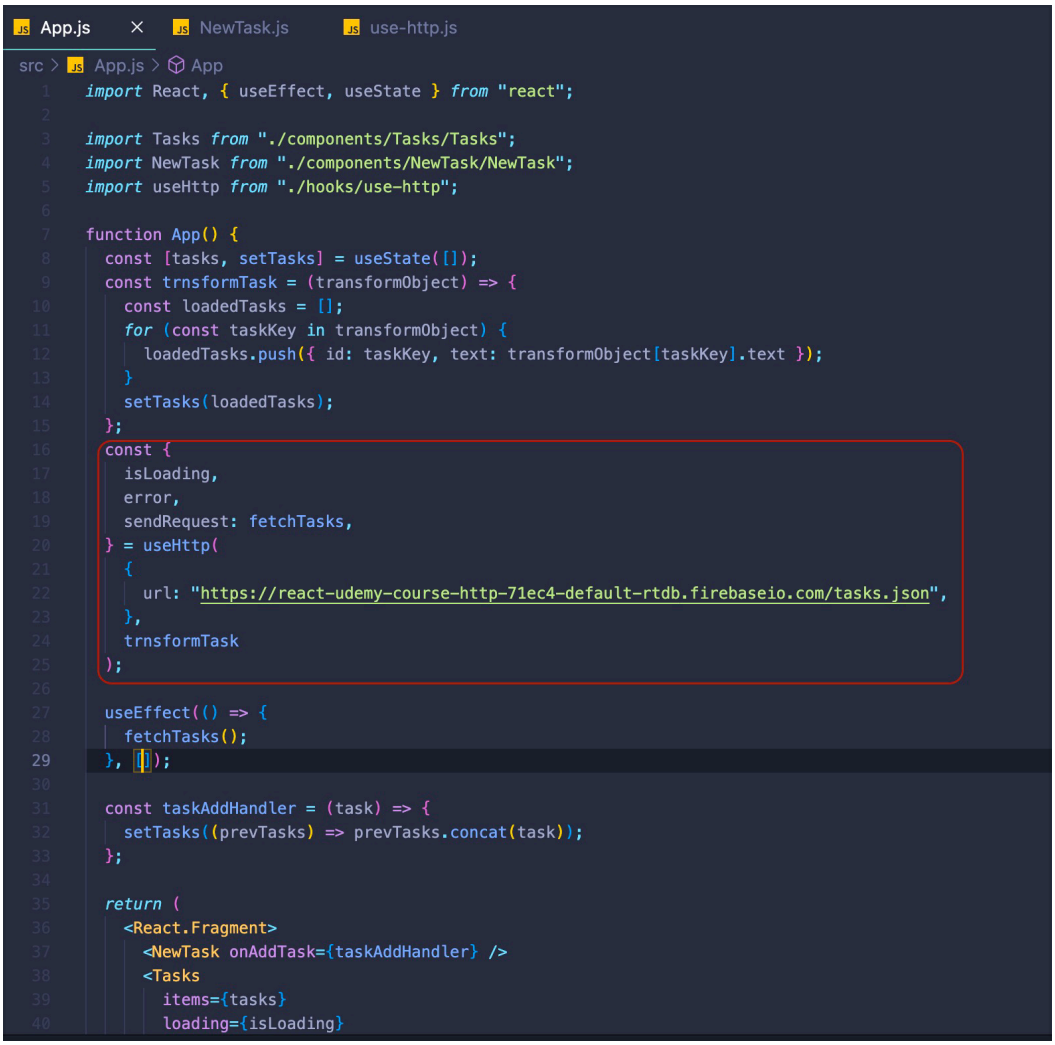
```
import useCounter from "../hooks/use-counter";

import Card from "./Card";

const BackwardCounter = () => {
  const counter = useCounter(0, -1);
  return <Card>{counter}</Card>;
};

export default BackwardCounter;
```

Building & Using the Custom Http Hook:

App.js NewTask.js use-http.js

src > hooks > use-http.js > useHttp

```
import { useState } from "react";
const useHttp = (requestConfig, dataTransformer) => {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);

  const sendRequest = async (taskText) => {
    setIsLoading(true);
    setError(null);
    try {
      const response = await fetch(requestConfig.url, {
        method: requestConfig.method ? requestConfig.method : "GET",
        headers: requestConfig.headers ? requestConfig.headers : {},
        body: requestConfig.body ? JSON.stringify(requestConfig.body) : null,
      });

      if (!response.ok) {
        throw new Error("Request failed!");
      }

      const data = await response.json();

      //   const loadedTasks = [];

      //   for (const taskKey in data) {
      //     loadedTasks.push({ id: taskKey, text: data[taskKey].text });
      //   }
      dataTransformer(data);
    } catch (err) {
      setError(err.message || "Something went wrong!");
    }
    setIsLoading(false);
  };
  return {
    isLoading,
    error,
    sendRequest,
  };
};

export default useHttp;
```



Section 16: Working with Forms & User Input:

Handling the "was touched" State:

SimpleInput.js × index.css

src > components > SimpleInput.js > SimpleInput > formSubmitHandler

```jsx
import { useState } from "react";
const SimpleInput = (props) => {
  const [enteredName, setEnteredName] = useState("");
  const [enteredNameTouch, setEnteredNameTouch] = useState(false);

  const enteredNameIsValid = enteredName.trim() !== "";
  const nameInputIsInvalid = !enteredNameIsValid && enteredNameTouch;

  const nameInputChangeHandler = (event) => {
    setEnteredName(event.target.value);
  };

  const nameInputBlurHandler = (event) => {
    setEnteredNameTouch(true);
  };

  const formSubmitHandler = (event) => {
    event.preventDefault();
    setEnteredNameTouch(true);
    if (!enteredNameIsValid) {
      return;
    }
    setEnteredName("");
    setEnteredNameTouch(false);
  };
  const nameEnteredValid = nameInputIsInvalid
    ? "form-control invalid"
    : "form-control";

  return (
    <form onSubmit={formSubmitHandler}>
      <div className={nameEnteredValid}>
        <label htmlFor="name">Your Name</label>
        <input
          type="text"
          id="name"
          onChange={nameInputChangeHandler}
          onBlur={nameInputBlurHandler}
        />
        {nameInputIsInvalid && (
          <p className="error-text">Can't submit an empty form</p>
        )}
      </div>
      <div className="form-actions">
        <button>Submit</button>
      </div>
    </form>
  );
};

export default SimpleInput;
```

Adding A Custom Input Hook:

SimpleInput.js use-input.js index.css

src > hooks > use-input.js > useInput

```
import { useState } from "react";
// This hook will manage the state of an input

const useInput = (validateValueFunc) => {
  const [enteredValue, setEnteredValue] = useState("");
  const [isTouched, setIsTouched] = useState(false);

  const valueIsValid = validateValueFunc(enteredValue);
  const hasError = !valueIsValid && isTouched;

  const valueChangeHandler = (event) => {
    setEnteredValue(event.target.value);
  };
  const inputBlurHandler = (event) => {
    setIsTouched(true);
  };

  const rest = () => {
    setEnteredValue("");
    setIsTouched(false);
  };

  return {
    value: enteredValue,
    isValid: valueIsValid,
    hasError,
    valueChangeHandler,
    inputBlurHandler,
    rest,
  };
};
export default useInput;
```

Section 18: Diving into Redux (An Alternative To The Context API):

Redux is an alternative for react “Context”.
Overall Redux is proffered for “Cross-Component State” and “App-Wide State”.
In order to start using redux we need to create a store (after we used “npm install redux”).

redux-demo.js

```
const redux = require("redux");

const counterReducer = (currentState = { counter: 0 }, dispatchAction) => {
  return {
    counter: currentState.counter + 1,
  };
};

const store = redux.createStore(counterReducer);

const counterSubscriber = () => {
  const latestStore = store.getState();
  console.log(latestStore);
};

store.subscribe(counterSubscriber);
store.dispatch({ type: "increment" });
```

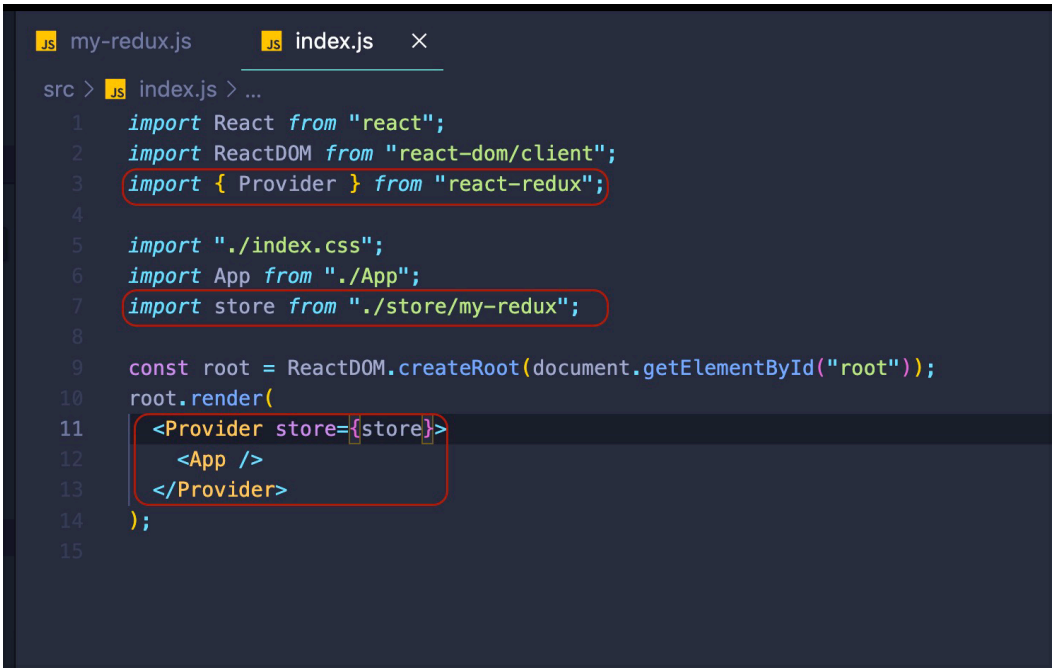
https://redux.js.org/introduction/why-rtk-is-redux-today

Another example using redux in react components:

my-redux.js | index.js

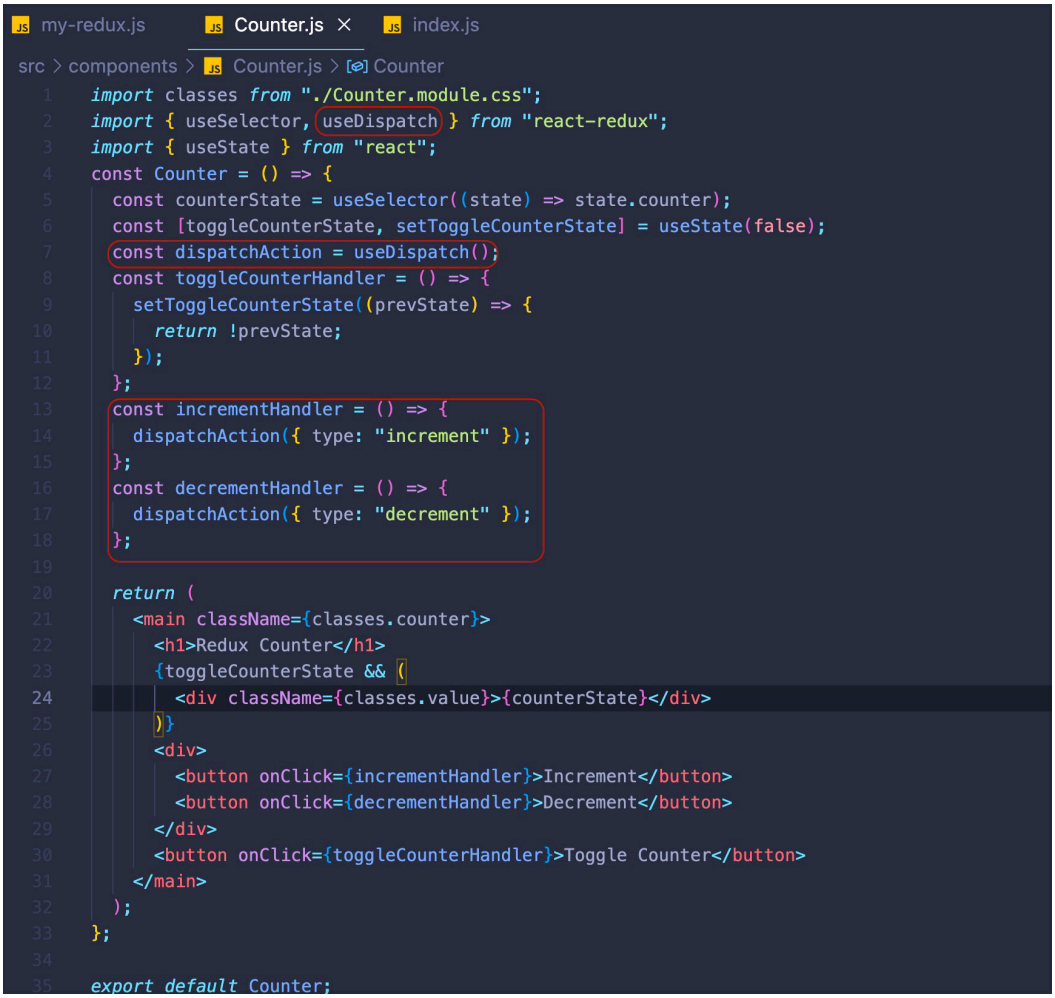
src > store > my-redux.js > default

```
const redux = require("redux");

const counterReducer = (currentState = { counter: 0 }, action) => {
  let value = currentState;
  if (action.type === "increment") {
    value = {
      counter: currentState.counter + 1,
    };
  }
  if (action.type === "decrement") {
    value = {
      counter: currentState.counter - 1,
    };
  }
  return value;
};

const store = redux.createStore(counterReducer);

export default store;
```

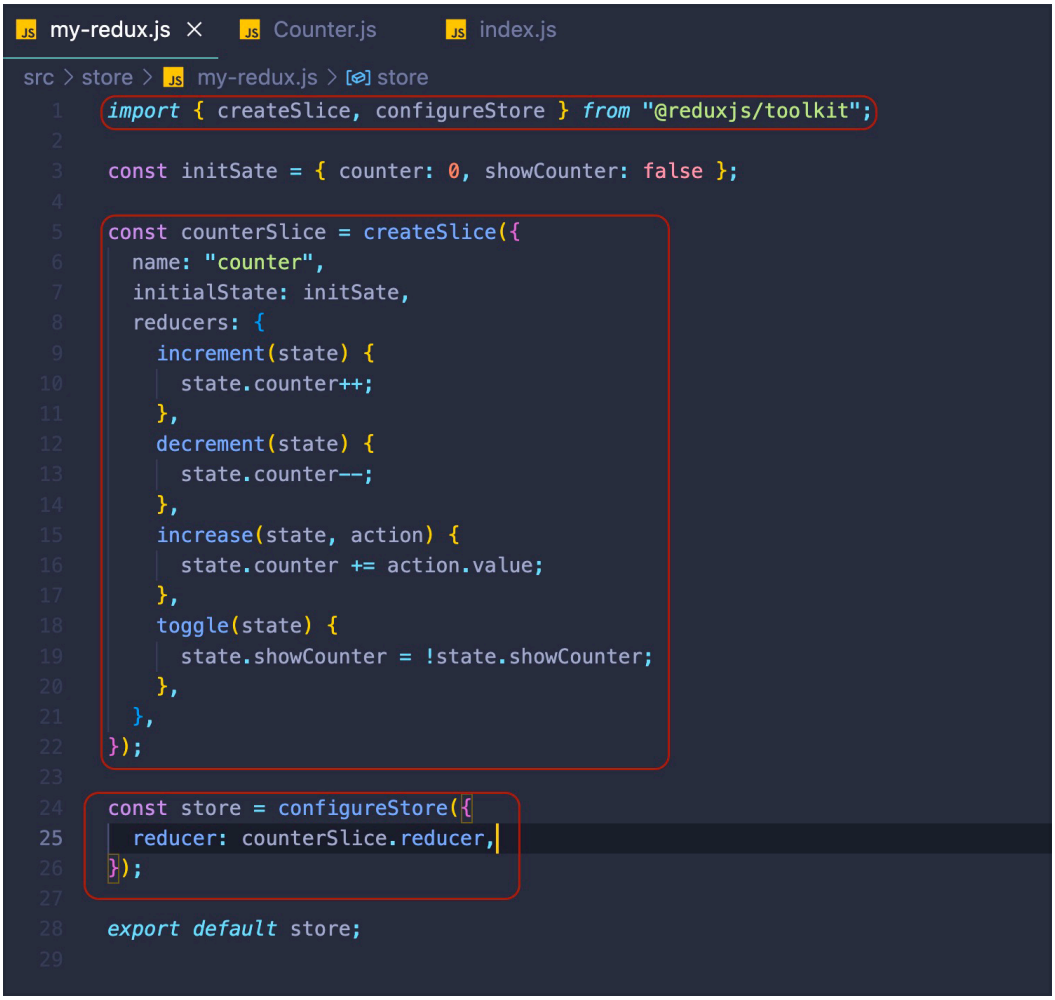


Using Redux Data in React Components:

Dispatching Actions From Inside Components:



Redux toolkit:



Migrating Everything To Redux Toolkit:

my-redux.js | Counter.js | index.js

src > components > Counter.js > Counter > increaseHandler

```
import classes from "./Counter.module.css";
import { useSelector, useDispatch } from "react-redux";
import { counterActions } from "../store/my-redux";
const Counter = () => {
  const counterValue = useSelector((state) => state.counter);
  const counterState = useSelector((state) => state.showCounter);

  const dispatchAction = useDispatch();
  const toggleCounterHandler = () => {
    dispatchAction(counterActions.toggle());
  };
  const incrementHandler = () => {
    dispatchAction(counterActions.increment());
  };
  const increaseHandler = () => {
    dispatchAction(counterActions.increase(5)); // Creates - {type: "Unique", payload: 5}
  };
  const decrementHandler = () => {
    dispatchAction(counterActions.decrement());
  };

  return (
    <main className={classes.counter}>
      <h1>Redux Counter</h1>
      {counterState && <div className={classes.value}>{counterValue}</div>}
      <div>
        <button onClick={increaseHandler}>Increment by 5</button>
        <button onClick={incrementHandler}>Increment</button>
        <button onClick={decrementHandler}>Decrement</button>
      </div>
      <button onClick={toggleCounterHandler}>Toggle Counter</button>
    </main>
  );
};

export default Counter;
```

Working with Multiple Slices:

my-redux.js | counter-slice.js | auth-slice.js

src > store > counter-slice.js > ...

```
import { createSlice } from "@reduxjs/toolkit";
const initialCounterState = { counter: 0, showCounter: false };

const counterSlice = createSlice({
  name: "counter",
  initialState: initialCounterState,
  reducers: {
    increment(state) {
      state.counter++;
    },
    decrement(state) {
      state.counter--;
    },
    increase(state, action) {
      state.counter += action.payload;
    },
    toggle(state) {
      state.showCounter = !state.showCounter;
    },
  },
});
export default counterSlice.reducer;
export const counterActions = counterSlice.actions;
```

my-redux.js | counter-slice.js | auth-slice.js

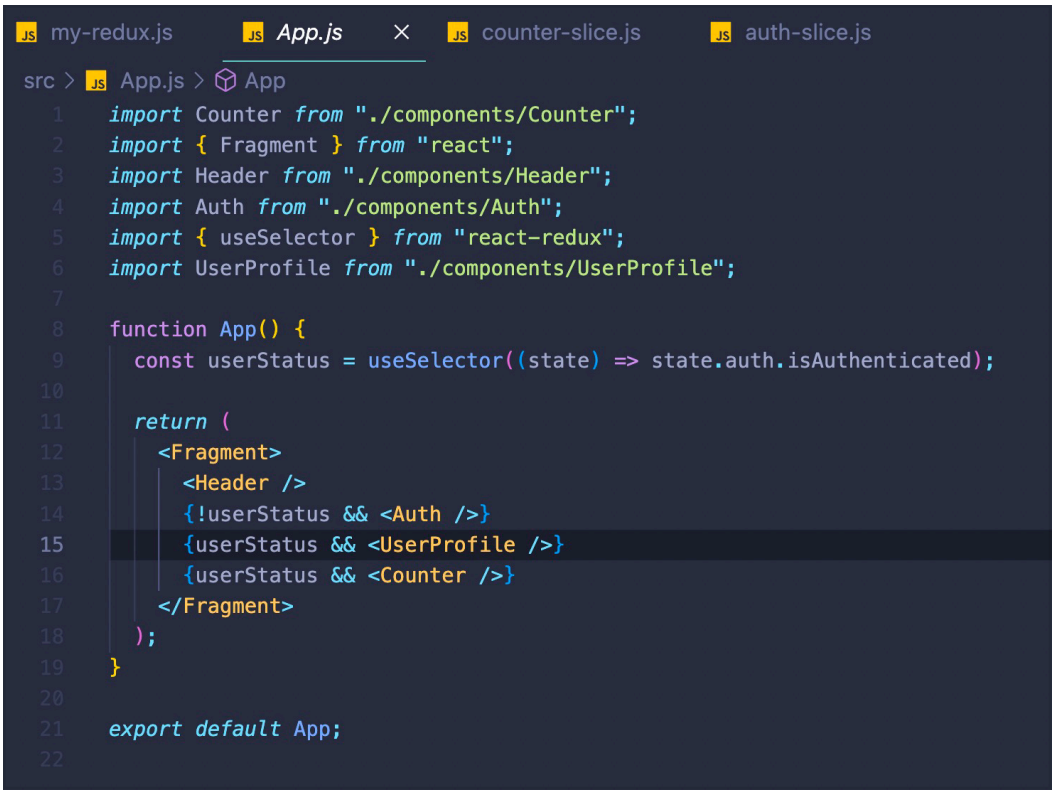
src > store > auth-slice.js > ...

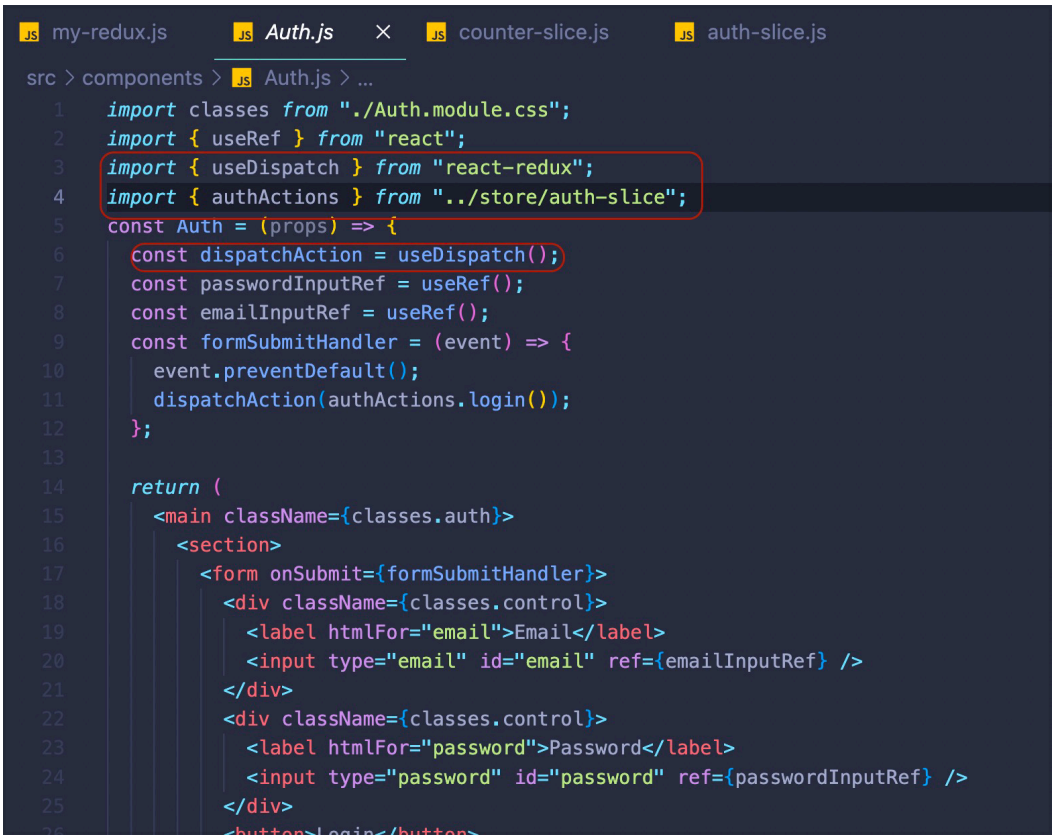
```
import { createSlice } from "@reduxjs/toolkit";

const initialAuthState = {
  isAuthenticated: false,
};

const authSlice = createSlice({
  name: "authentication",
  initialState: initialAuthState,
  reducers: {
    login(state) {
      state.isAuthenticated = true;
    },
    logout(state) {
      state.isAuthenticated = false;
    },
  },
});
export default authSlice.reducer;
export const authActions = authSlice.actions;
```

my-redux.js | counter-slice.js | auth-slice.js

src > store > my-redux.js > store > reducer > auth

```
import { configureStore } from "@reduxjs/toolkit";
import counterReducer from "./counter-slice";
import authReducer from "./auth-slice";
const store = configureStore({
  reducer: {
    counter: counterReducer,
    auth: authReducer,
  },
});

export default store;
```





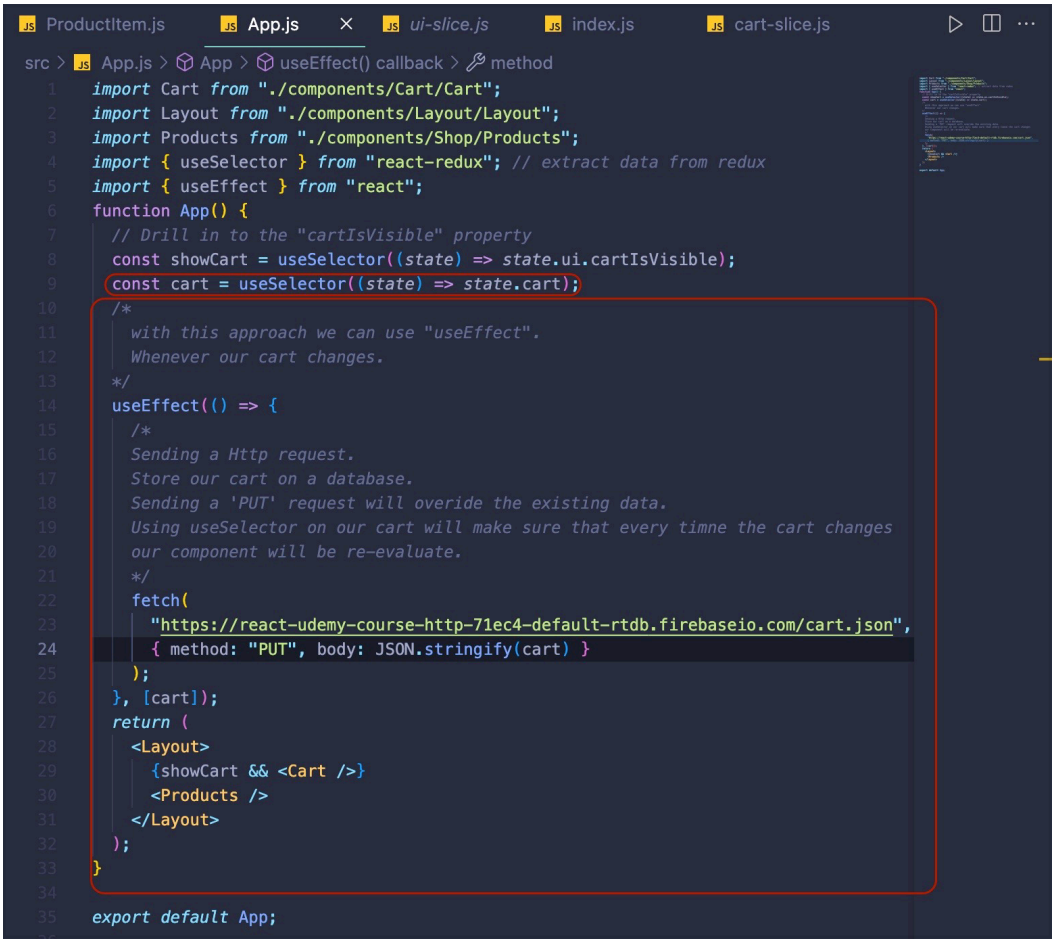
Clone from a branch - **git clone --single-branch --branch <branchname> <remote-repo>**

Advance Redux:

Frontend Code vs Backend Code:

In this section we learn how to send data to a backend server while using redux

Using useEffect with Redux:

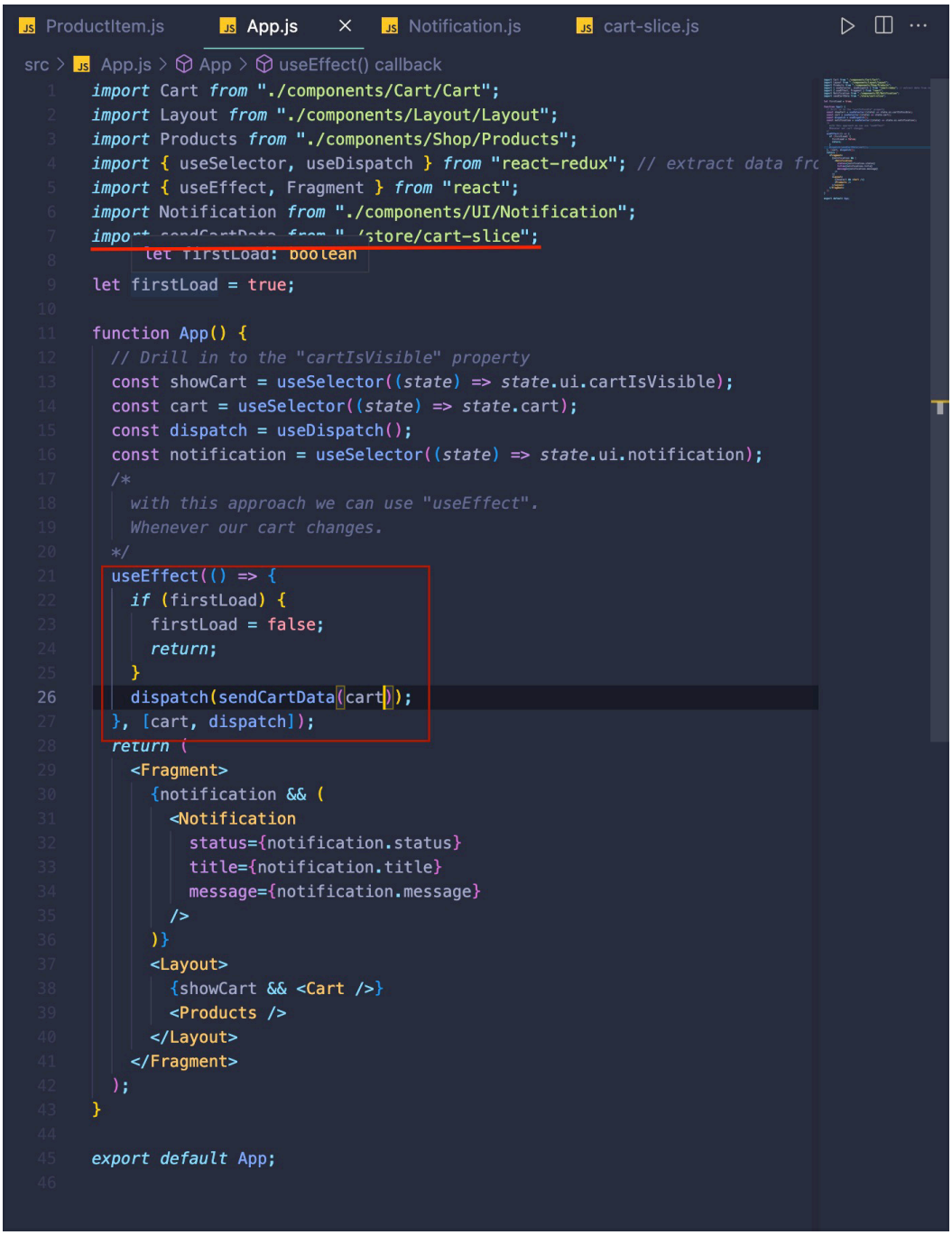


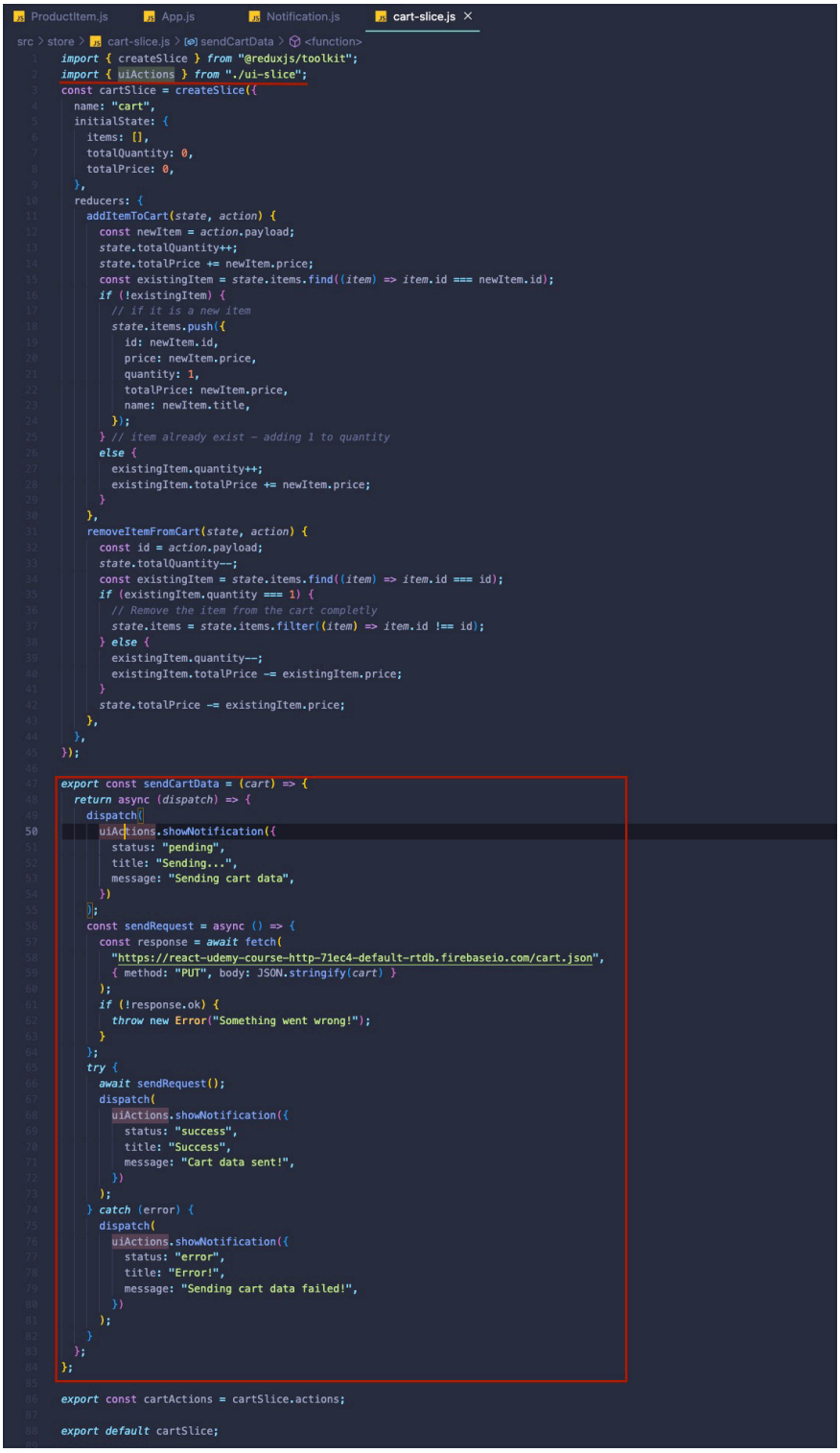
We face one problem when using useEffect the way we currently do it: It will execute when our app starts.
Why is this an issue?
It's a problem because this will send the initial (i.e. empty) cart to our backend and overwrite any data stored there.
We'll fix this over the next lectures, I just wanted to point it out here

Handling Http States & Feedback with Redux:

Another way of accomplishing this is by using “action creators”:

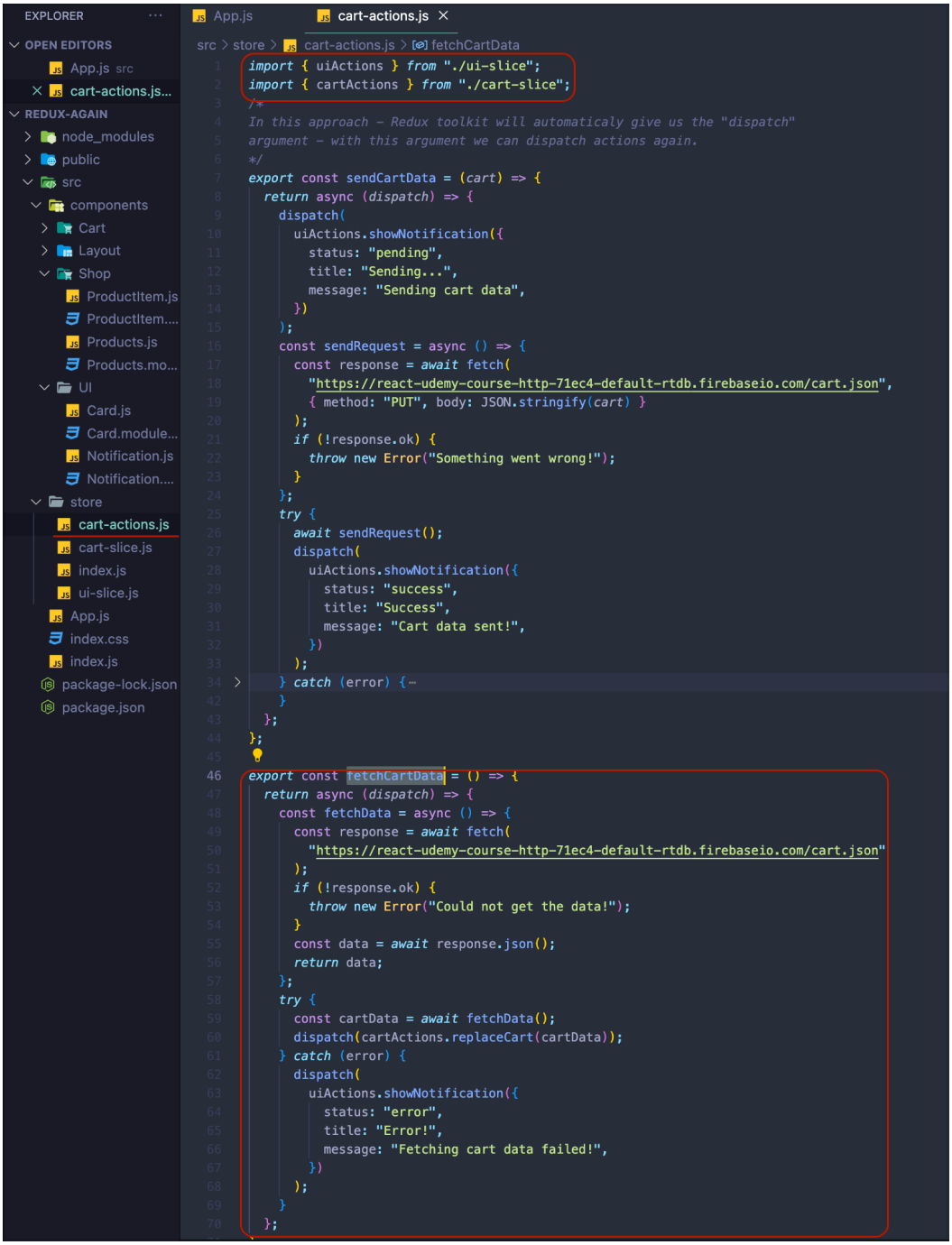
App.js a lot slimmer


ProductItem.js
App.js
Notification.js
cart-slice.js
src > App.js > App > useEffect() callback
import Cart from "./components/Cart/Cart";
import Layout from "./components/Layout/Layout";
import Products from "./components/Shop/Products";
import { useSelector, useDispatch } from "react-redux"; // extract data fro
import { useEffect, Fragment } from "react";
import Notification from "./components/UI/Notification";
import sendCartData from "../store/cart-slice";
let firstLoad: boolean
let firstLoad = true;
function App() {
// Drill in to the "cartIsVisible" property
const showCart = useSelector((state) => state.ui.cartIsVisible);
const cart = useSelector((state) => state.cart);
const dispatch = useDispatch();
const notification = useSelector((state) => state.ui.notification);
/*
with this approach we can use "useEffect".
Whenever our cart changes.
*/
useEffect(() => {
if (firstLoad) {
firstLoad = false;
return;
}
dispatch(sendCartData(cart));
}, [cart, dispatch]);
return (
<Fragment>
{notification && (
<Notification
status={notification.status}
title={notification.title}
message={notification.message}
/>
)}
<Layout>
{showCart && <Cart />}
<Products />
</Layout>
</Fragment>
);
}
export default App;



Getting Started with Fetching Data:

Moved the logic of storing the data in the database to a separate file. Also, added a new function for fetching the cart from the database.



The new App.js file.
There is a problem - when we fetch the cart for the first time we also set the cart parameter too, this parameter is also in the second useEffect as a dependency.

With this fix its complete:


App.js
cart-slice.js
cart-actions.js
src > store > cart-slice.js > cartSlice > reducers > removeItemFromCart
import { createSlice } from "@reduxjs/toolkit";
const cartSlice = createSlice({
name: "cart",
initialState: {
items: [],
totalQuantity: 0,
totalPrice: 0,
changed: false,
},
reducers: {
replaceCart(state, action) {
state.items = action.payload.items;
state.totalQuantity = action.payload.totalQuantity;
state.totalPrice = action.payload.totalPrice;
},
addItemToCart(state, action) {
const newItem = action.payload;
state.totalQuantity++;
state.totalPrice += newItem.price;
state.changed = true;
const existingItem = state.items.find((item) => item.id === newItem.i
if (!existingItem) {
// if it is a new item
state.items.push({
id: newItem.id,
price: newItem.price,
quantity: 1,
totalPrice: newItem.price,
name: newItem.title,
});
} // item already exist - adding 1 to quantity
else {
existingItem.quantity++;
existingItem.totalPrice += newItem.price;
}
},
removeItemFromCart(state, action) {
const id = action.payload;
state.totalQuantity--;
state.changed = true;
const existingItem = state.items.find((item) => item.id === id);
if (existingItem.quantity === 1) {
// Remove the item from the cart completly
state.items = state.items.filter((item) => item.id !== id);
} else {
existingItem.quantity--;
existingItem.totalPrice -= existingItem.price;
}
state.totalPrice -= existingItem.price;
},

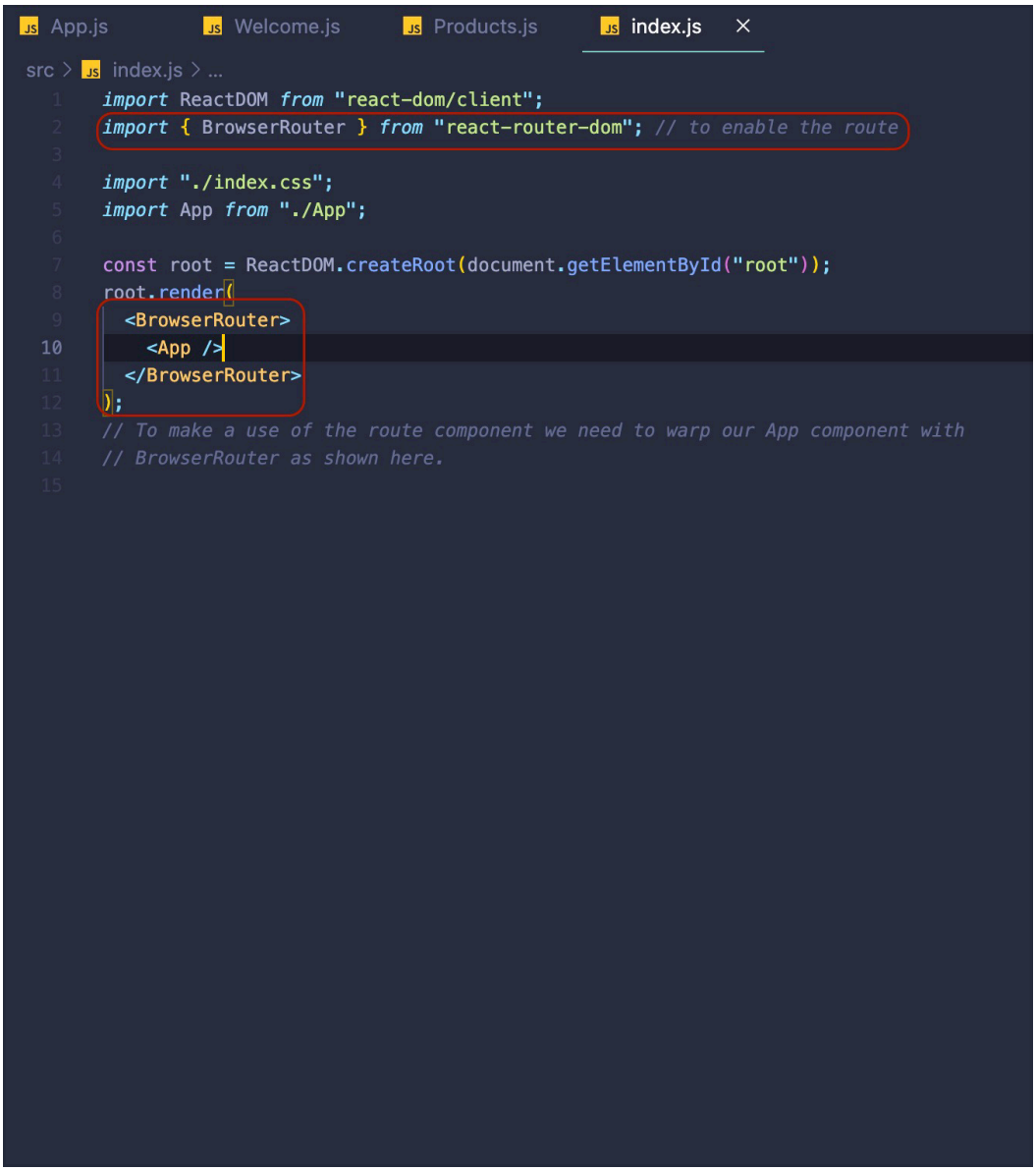
When working with redux download redux DevTools - an extension to debug react applications using redux. (I need to check it out)
https://redux-toolkit.js.org
Udemy course lecture - https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/25600378#content

Section 20: Building a Multi-Page SPA with React Router:

to install react router package (Version 5):
In terminal: npm install react-router-dom@5

Defining & Using Routes:

What we want to achieve is:
When we have a request for our-domain.com/ => Render Component A
When we have a request for our-domain.com/WhatEver => Render Component B



App.js Welcome.js Products.js index.js

src > App.js > App

```
import { Route } from "react-router-dom"; // A component that let us define a certine path and the component that should be render
import Products from "./Components/Products";
import Welcome from "./Components/Welcome";
function App() {
  return (
    <div>
      <Route path="/welcome">
        <Welcome />
      </Route>
      <Route path="/products">
        <Products />
      </Route>
    </div>
  );
}

export default App;
/*
<Route path="/welcome"/> - with that we saying - this route should be active when this:
our-domain.com/welcome will be active

*/
```

Working with Links:

App.js MainHeader.js Welcome.js Products.js index.js

src > Components > MainHeader.js > MainHeader

```
import { Link } from "react-router-dom"; // Unlike the regular <a> tag this component prevents the browser default of loading another html page.
const MainHeader = () => {
  return (
    <header>
      <nav>
        <ul>
          <li>
            <Link to="/welcome">welcome</Link>
          </li>
          <li>
            <Link to="/products">products </Link>
          </li>
        </ul>
      </nav>
    </header>
  );
};
export default MainHeader;
```

Using NavLinks:

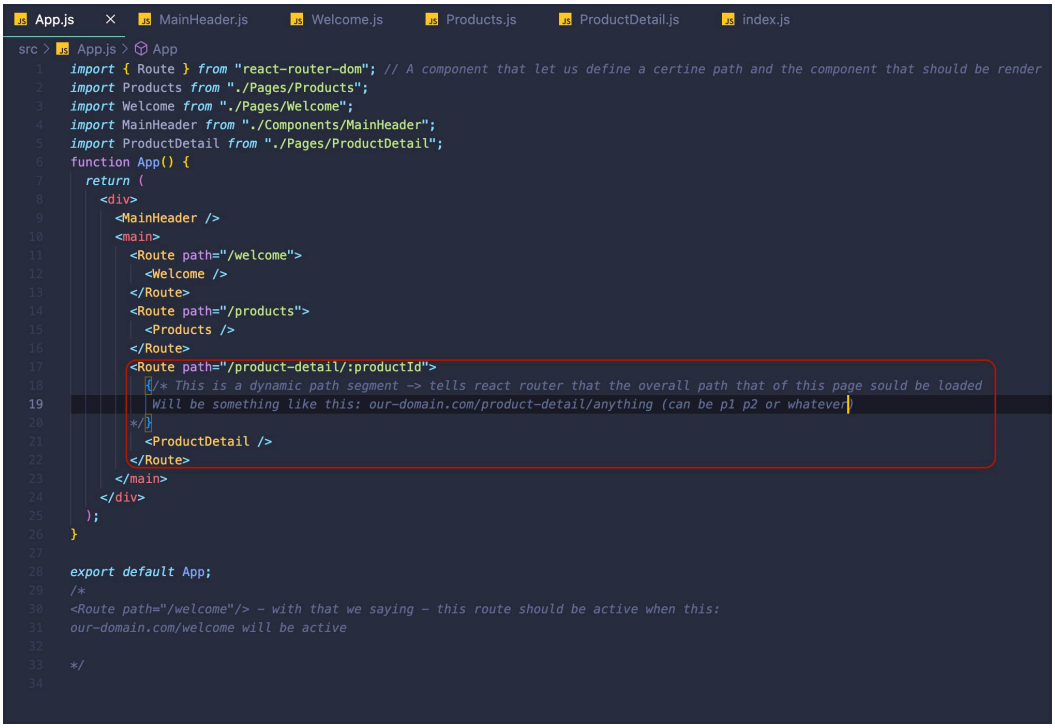
If we want to highlight the current link that we pressed we can use another feature from react-router:
We can use the

App.js MainHeader.js Welcome.js Products.js index.js

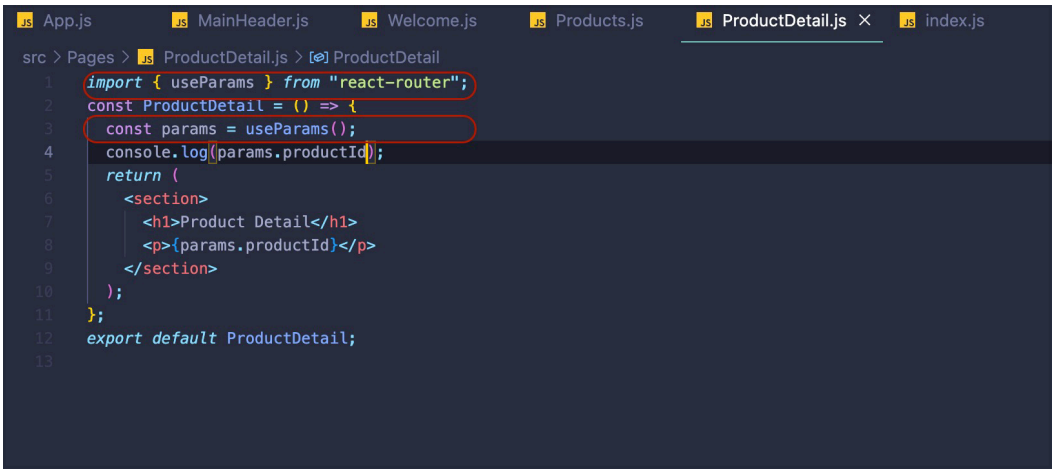
src > Components > MainHeader.js > MainHeader

```
import { NavLink } from "react-router-dom"; // Unlike the regular <a> tag this component prevents the browser default of loading another html page.
import classes from "./MainHeader.module.css";
const MainHeader = () => {
  return (
    <header className={classes.header}>
      <nav>
        <ul>
          <li>
            <NavLink activeClassName={classes.active} to="/welcome">
              Welcome
            </NavLink>
          </li>
          <li>
            <NavLink activeClassName={classes.active} to="/products">
              Products{" "}
            </NavLink>
          </li>
        </ul>
      </nav>
    </header>
  );
};
export default MainHeader;
```

Adding Dynamic Routes with Params:



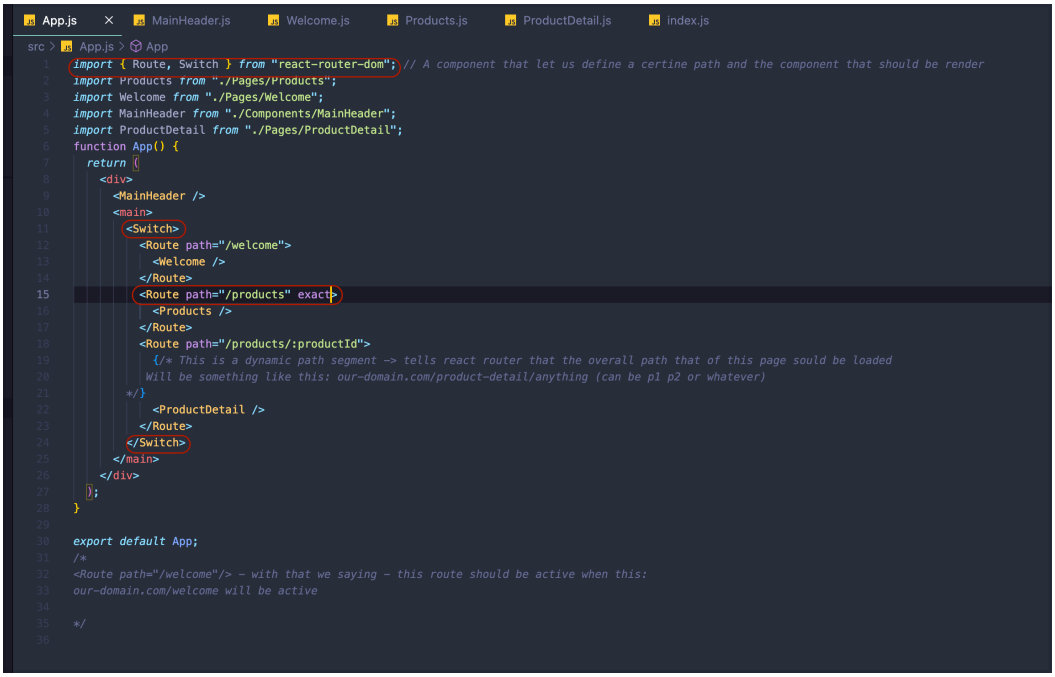
Extracting Route Params:



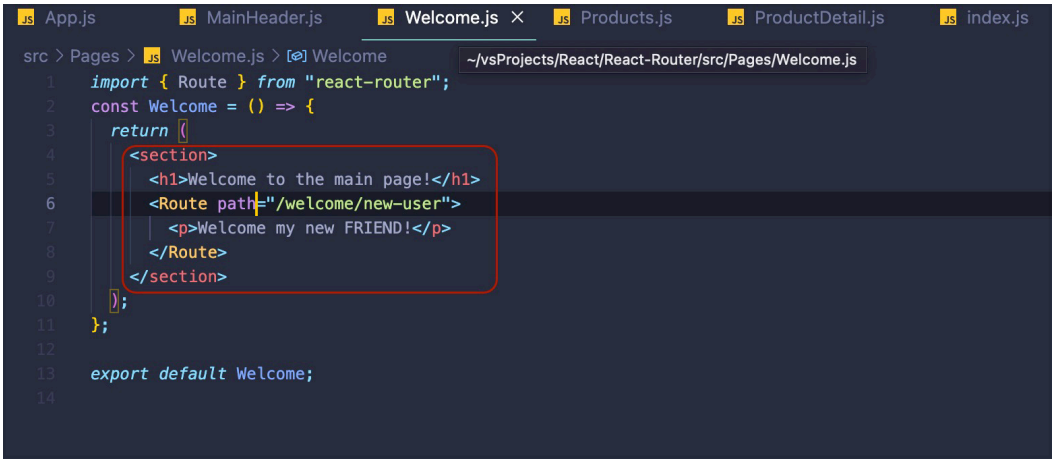
Using "Switch" and "exact" For Configuring Routes:

The Switch import make sure that only one Route will be rendered. It goes top to bottom to find the first Route that its path is a part of our path (that we are on the website).

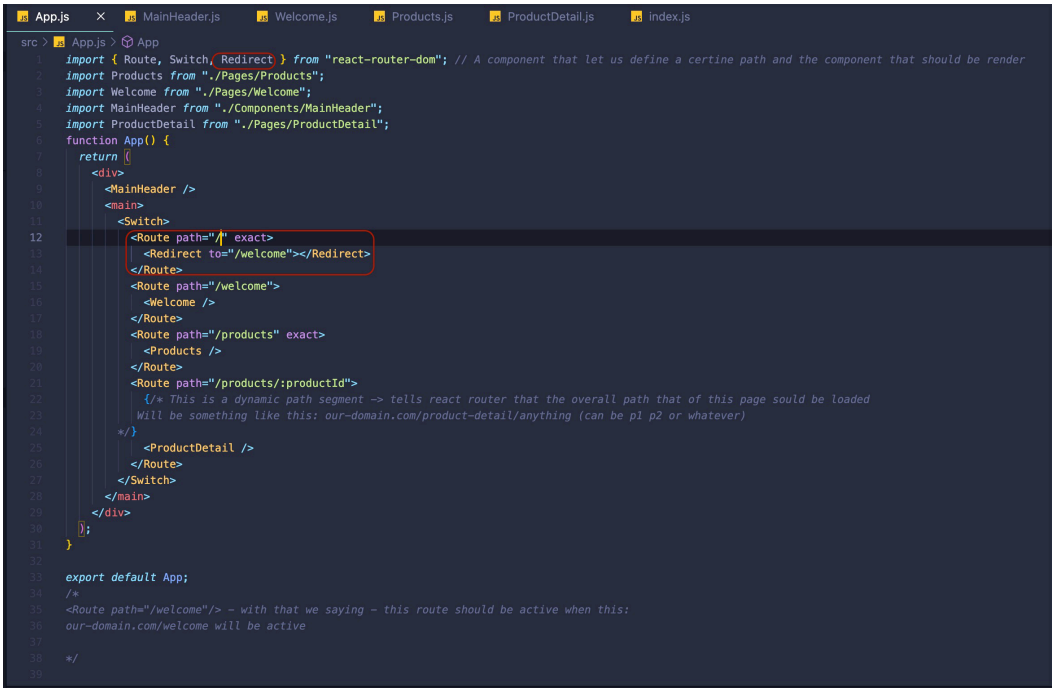
The “exact” param make sure that the path is exactly as our path we on (in the website) and not part of it.



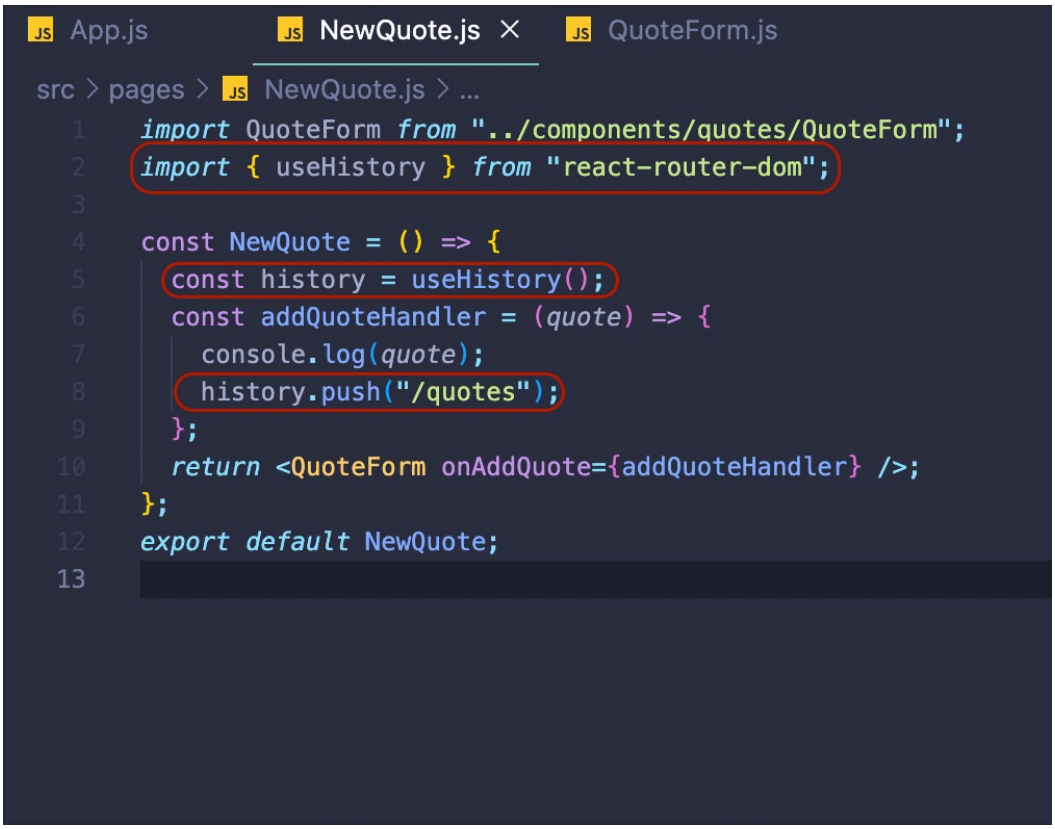
Working with Nested Routes:



Redirecting The User:



Implementing Programmatic (Imperative) Navigation:



Preventing Possibly Unwanted Route Transitions with the "Prompt" Component:

Working with Query Parameters:

Getting Creative With Nested Routes:

Here is an example on hoe to take advantage of the route.
If we want something to appear on the screen only in cases of where the url is specific.
Creative way.

Writing More Flexible Routing Code:

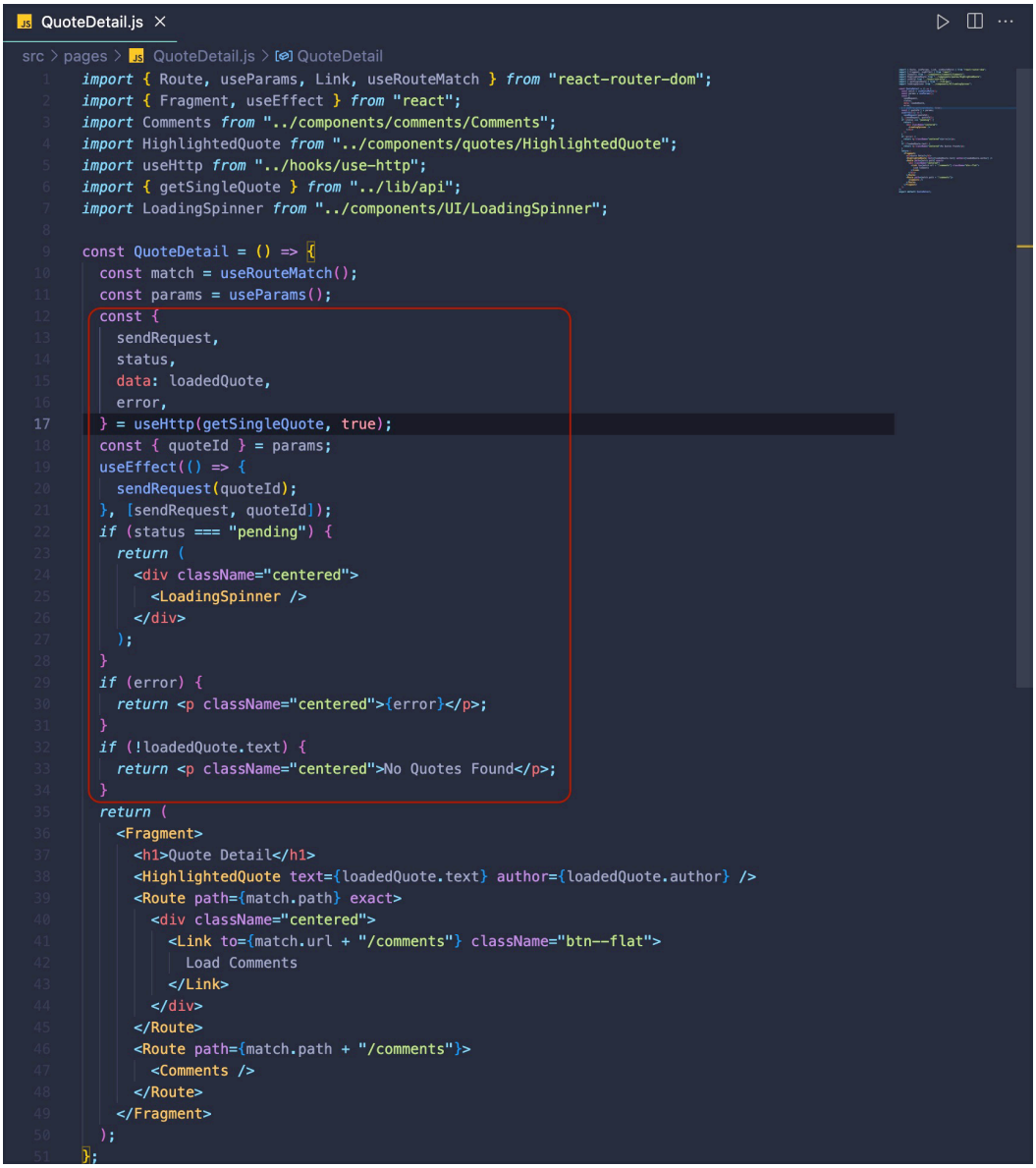
App.js QuoteForm.js QuoteList.js QuoteDetail.js

src > pages > QuoteDetail.js > QuoteDetail

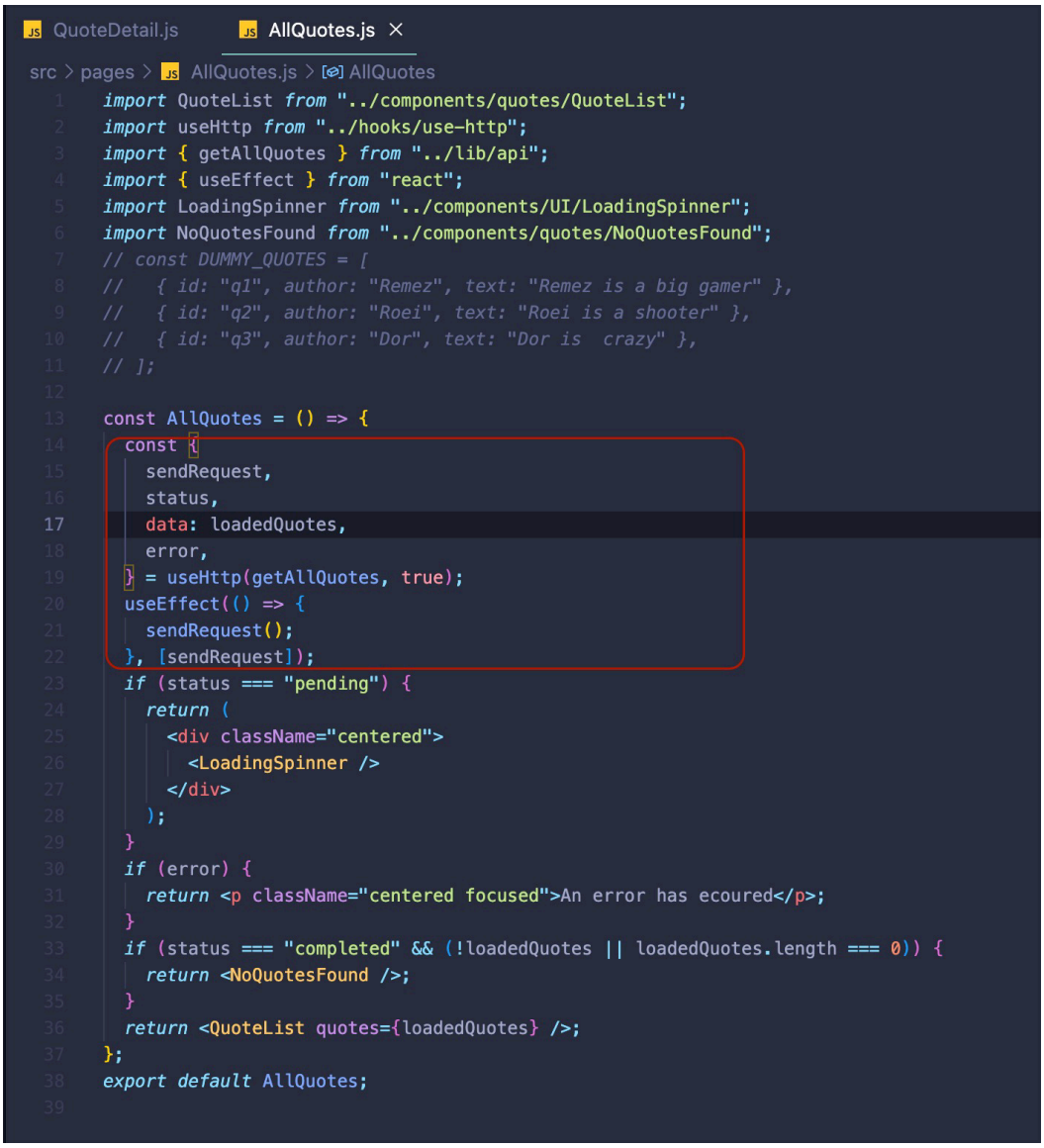
```
import { Route, useParams, Link, useRouteMatch } from "react-router-dom";
import { Fragment } from "react";
import Comments from "../components/comments/Comments";
import HighlightedQuote from "../components/quotes/HighlightedQuote";
const DUMMY_QUOTES = [
  { id: "q1", author: "Remez", text: "Remez is a big gamer" },
  { id: "q2", author: "Roei", text: "Roei is a shooter" },
  { id: "q3", author: "Dor", text: "Dor is  crazy" },
];
const QuoteDetail = () => {
  const match = useRouteMatch();
  const params = useParams();
  const quote = DUMMY_QUOTES.find((quote) => quote.id === params.quoteId);
  if (!quote) {
    return <p>No quote found!</p>;
  }
  return (
    <Fragment>
      <h1>Quote Detail</h1>
      <HighlightedQuote text={quote.text} author={quote.author} />
      <Route path={match.path} exact>
        <div className="centered">
          <Link to={match.url + "/comments"} className="btn--flat">
            Load Comments
          </Link>
        </div>
      </Route>
      <Route path={match.path + "/comments"}>
        <Comments />
      </Route>
    </Fragment>
  );
};
export default QuoteDetail;
```

Sending & Getting Quote Data via Http:

Using a custom http hook and a custom methods to fetch a single quote from firebase.



Using a custom http hook to send request to get all the quotes data from firebase. Also, using a custom methods
In (lib/api.js)



Adding the "Comments" Features:

Saved the finale project on /Users/remez19/vsProjects/React/React-Route-Practice

Adding the comments functionality at the end. For more info Udemy - https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/29303718#overview

Section 21: Deploying React Apps:

Adding Lazy Loading:

App.js

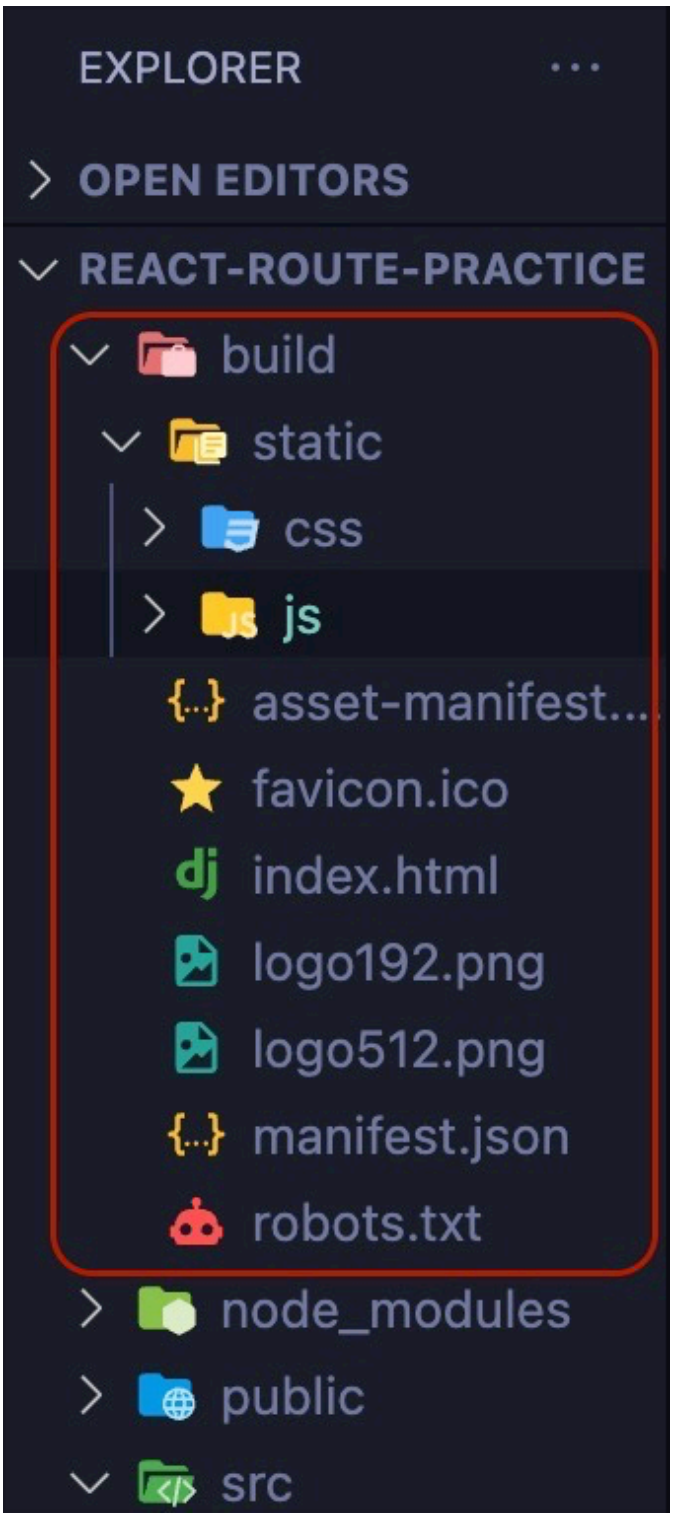
src > App.js > ...

```jsx
import React, { Suspense } from "react";
import { Route, Switch, Redirect } from "react-router-dom";

import AllQuotes from "./pages/AllQuotes";
import Layout from "./components/layout/Layout";
import LoadingSpinner from "./components/UI/LoadingSpinner";

/*
A method that helps us splliting our code.
The React.lazy will be called only when the page is needed.
To make this work we need to define a 'fall-back ui'
the code that we are downloading with:
 React.lazy(() => {
  import("./pages/NewQuote");
});
can take some time therefore we need to use special react component
called "Suspense" and warp all the react code that use React.lazy.
*/
const NewQuote = React.lazy(() => import("./pages/NewQuote"));
const QuoteDetail = React.lazy(() => import("./pages/QuoteDetail"));
const NotFound = React.lazy(() => import("./pages/NotFound"));

function App() {
  return (
    <Layout>
      <Suspense
        fallback={
          <div className="centered">
            <LoadingSpinner />
          </div>
        }
      >
        <Switch>
          <Route path="/" exact>
            <Redirect to="/quotes"></Redirect>
          </Route>
          <Route path="/quotes" exact>
            <AllQuotes />
          </Route>
          <Route path="/quotes/:quoteId">
            <QuoteDetail />
          </Route>
          <Route path="/new-quote">
            <NewQuote />
          </Route>
          <Route path="*">
            <NotFound />
          </Route>
        </Switch>
      </Suspense>
    </Layout>
  );
}
//<Route path='*'> -> tell's react to accept all the paths
```

Building The Code For Production:

Typing in the terminal - “**npm run build**”.

This script will **optimize** our code and, creates a “**build**” folder.



Getting Started With Deployment (Uploading Files):
https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/25600988#overview

Exploring Routing Issues & Finishing Deployment:
https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/25600992#overview

Section 22: Adding Authentication To React Apps:

Adding User Signup:

App.js AuthForm.js

src > components > Auth > AuthForm.js > AuthForm > submitHandler

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

import classes from "./AuthForm.module.css";

const API_KEY = "AIzaSyDVxBZnx2-yQwfW4crmWoyjgnxrS4gEGn4";

const AuthForm = () => {
  const [isLogin, setIsLogin] = useState(true);
  const emailInputRef = useRef();
  const passwordInputRef = useRef();
  const switchAuthModeHandler = () => {
    setIsLogin((prevState) => !prevState);
  };
  const submitHandler = (event) => {
    event.preventDefault();

    const userEmail = emailInputRef.current.value;
    const userPassword = passwordInputRef.current.value;
    // Can check for valid input

    if (isLogin) {
    } else {
      // Sending a sign-up request
      fetch(
        `https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=${API_KEY}`,
        {
          method: "POST",
          body: JSON.stringify({
            email: userEmail,
            password: userPassword,
            returnSecureToken: true,
          }),
          headers: { "Content-Type": "application/json" },
        }
      ).then((response) => {
        if (response.ok) {
          // Success case
        } else {
          response.json().then((data) => {
            // show error modal
            console.log(data);
          });
        }
      });
    }
  };
  return (
    <section className={classes.auth}>
      <h1>{isLogin ? "Login" : "Sign Up"}</h1>
      <form onSubmit={submitHandler}>
        <div className={classes.control}>
          <label htmlFor="email">Your Email</label>
          <input type="email" id="email" required ref={emailInputRef} />
        </div>
        <div className={classes.control}>
```

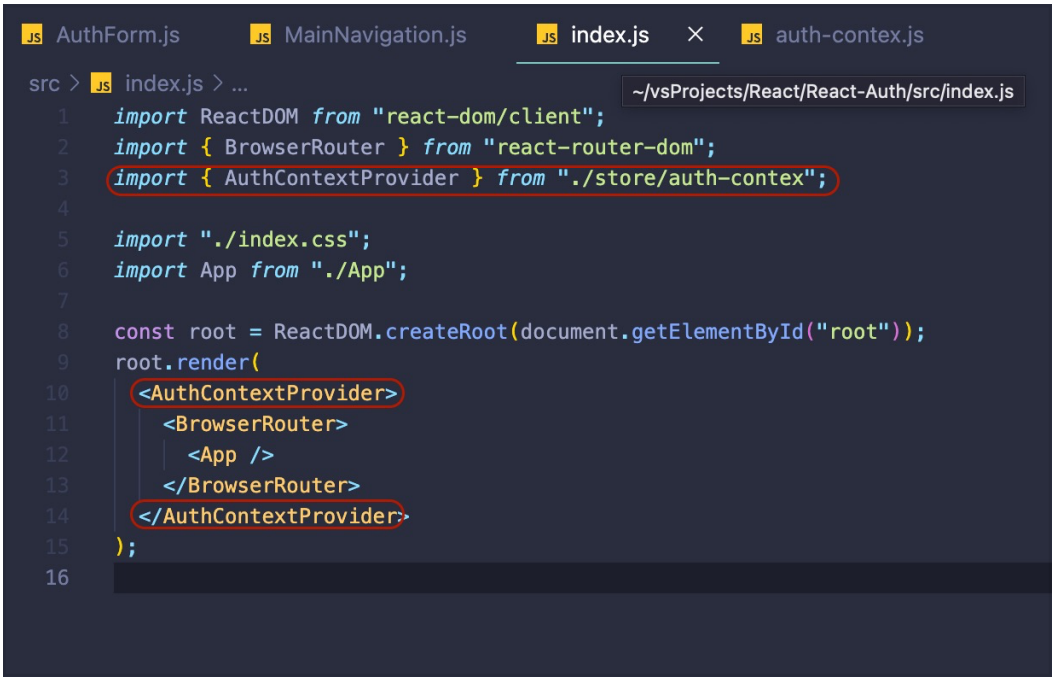
Adding User Login:

Managing The Auth State With Context:

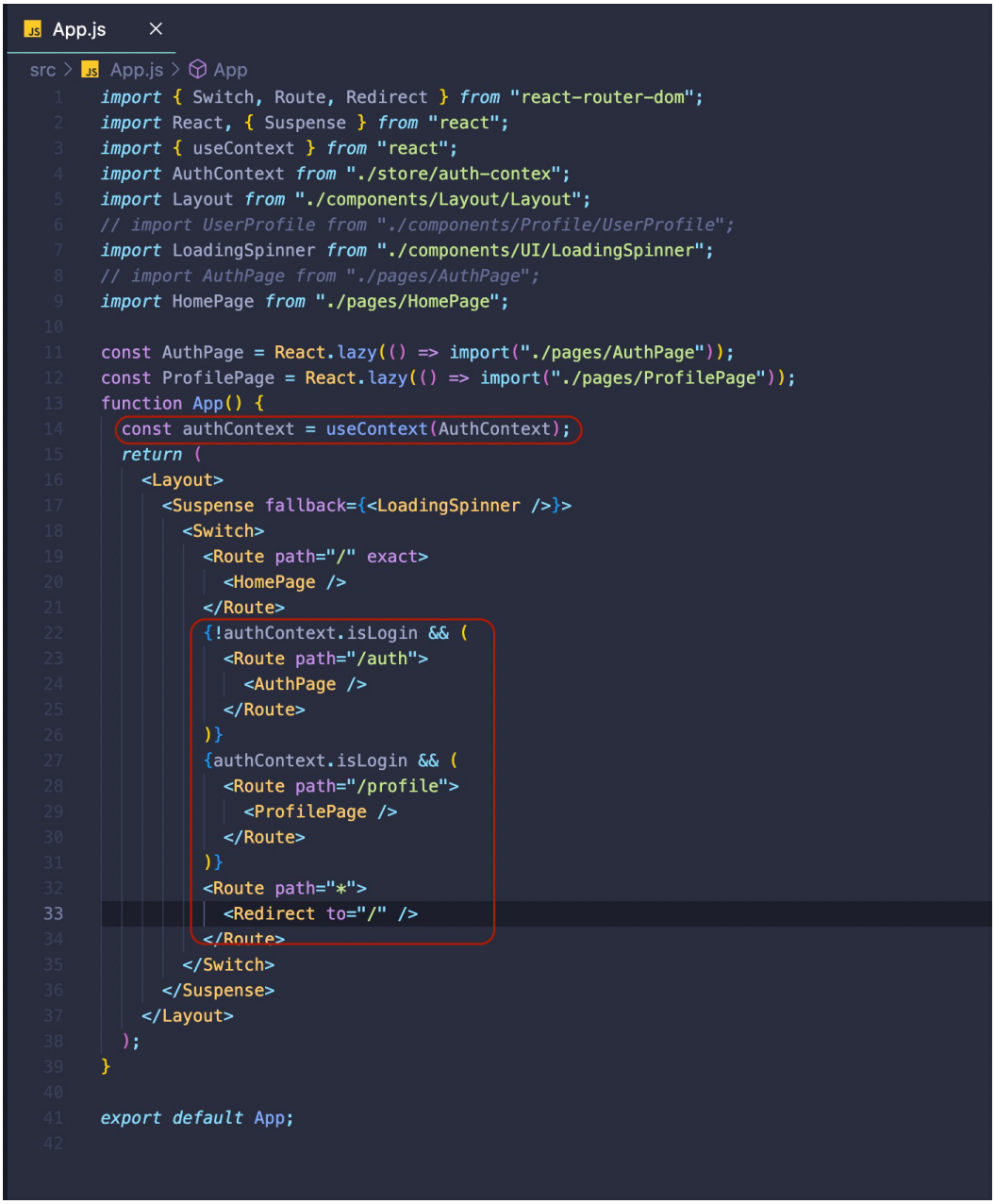
AuthForm.js MainNavigation.js auth-contex.js

src > store > auth-contex.js > AuthContextProvider > loginHandler

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

const AuthContext = React.createContext({
  token: "",
  isLogin: false,
  login: (token) => {},
  logout: () => {},
});

export const AuthContextProvider = (props) => {
  const [token, setToken] = useState(null);

  const userIsLoggedIn = !!token;

  const loginHandler = (token) => {
    setToken(token);
  };
  const logeOutHandler = () => {
    setToken(null);
  };
  const contextValue = {
    token: token,
    isLoggedIn: userIsLoggedIn,
    login: loginHandler,
    logout: logeOutHandler,
  };
  return (
    <AuthContext.Provider value={contextValue}>
      {props.children}
    </AuthContext.Provider>
  );
};

export default AuthContext;

```


AuthForm.js
MainNavigation.js
index.js
auth-contex.js
src > index.js > ...
~/vsProjects/React/React-Auth/src/index.js
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import { AuthContextProvider } from "./store/auth-contex";
import "./index.css";
import App from "./App";
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
<AuthContextProvider>
<BrowserRouter>
<App />
</BrowserRouter>
</AuthContextProvider>
);

Protecting Frontend Pages:



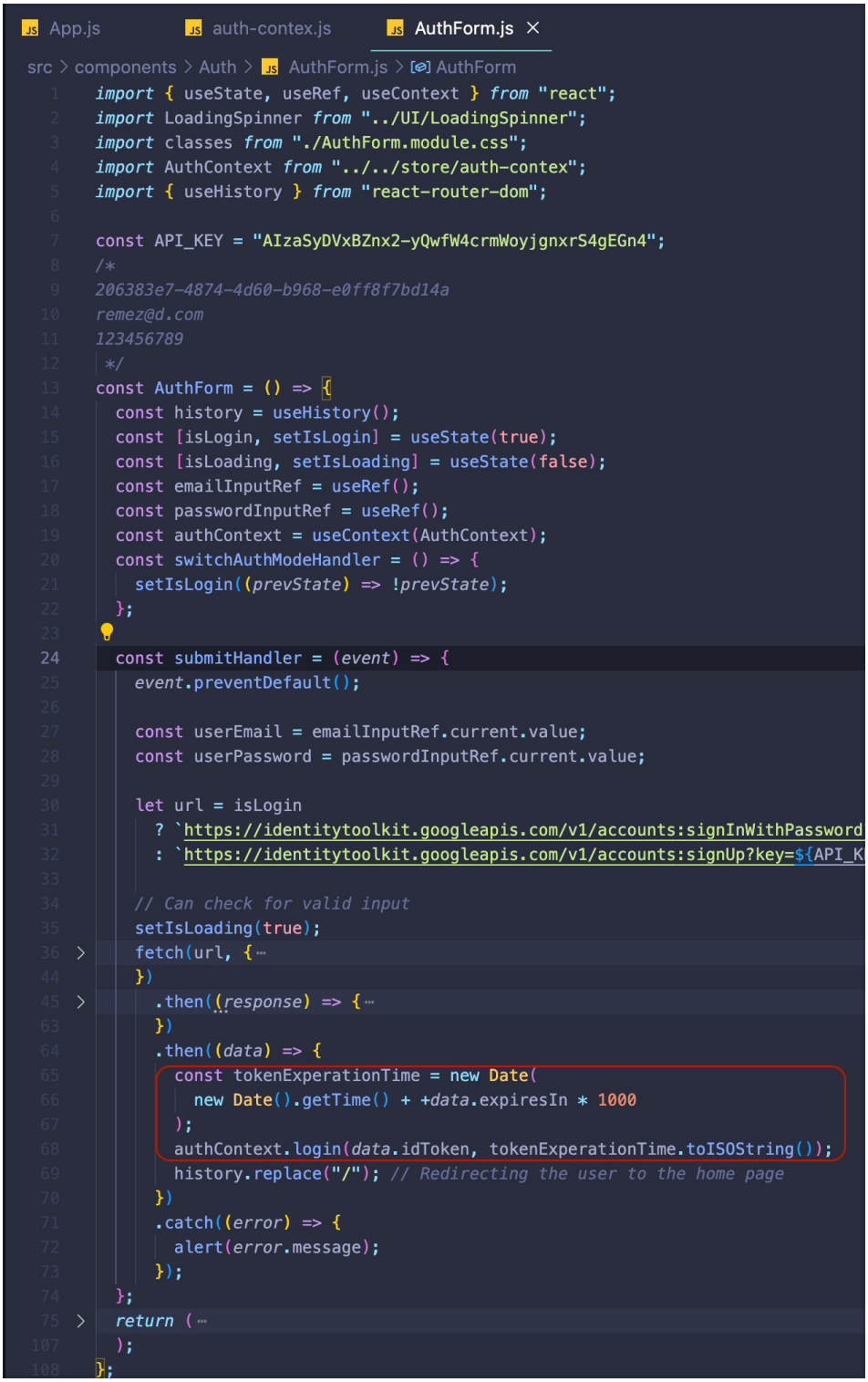
Persisting The User Authentication Status:

App.js | auth-contex.js | AuthForm.js

src > store > auth-contex.js > AuthContextProvider > loginHandler

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

const AuthContext = React.createContext({
  token: "",
  isLogin: false,
  login: (token) => {},
  logout: () => {},
});

export const AuthContextProvider = (props) => {
  const initToken = localStorage.getItem("token");
  const [token, setToken] = useState(initToken);

  const userIsLoggedIn = !!token;

  const loginHandler = (token) => {
    // Saving the token in the browser local storage.
    // that way once a user logged in and refresh the page he will be
    // still logged in
    localStorage.setItem("token", token);
    setToken(token);
  };
  const logeOutHandler = () => {
    // localStorage.clear() - clears all the local storage
    localStorage.removeItem("token");
    setToken(null);
  };
  const contextValue = {
    token: token,
    isLoggedIn: userIsLoggedIn,
    login: loginHandler,
    logout: logeOutHandler,
  };
  return (
    <AuthContext.Provider value={contextValue}>
      {props.children}
    </AuthContext.Provider>
  );
};

export default AuthContext;
```

Adding Auto-Logout:


App.js
auth-contex.js
AuthForm.js
src > components > Auth > AuthForm.js > AuthForm
import { useState, useRef, useContext } from "react";
import LoadingSpinner from "../UI/LoadingSpinner";
import classes from "./AuthForm.module.css";
import AuthContext from "../../store/auth-contex";
import { useHistory } from "react-router-dom";
const API_KEY = "AIzaSyDVxBZnx2-yQwfW4crmWoyjgnxrS4gEGn4";
/*
206383e7-4874-4d60-b968-e0ff8f7bd14a
remez@d.com
123456789
*/
const AuthForm = () => {
const history = useHistory();
const [isLogin, setIsLogin] = useState(true);
const [isLoading, setIsLoading] = useState(false);
const emailInputRef = useRef();
const passwordInputRef = useRef();
const authContext = useContext(AuthContext);
const switchAuthModeHandler = () => {
setIsLogin((prevState) => !prevState);
};
const submitHandler = (event) => {
event.preventDefault();
const userEmail = emailInputRef.current.value;
const userPassword = passwordInputRef.current.value;
let url = isLogin
? `https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword
: `https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=${API_K
// Can check for valid input
setIsLoading(true);
fetch(url, {…
})
.then((response) => {…
})
.then((data) => {
const tokenExperationTime = new Date(
new Date().getTime() + +data.expiresIn * 1000
);
authContext.login(data.idToken, tokenExperationTime.toISOString());
history.replace("/"); // Redirecting the user to the home page
})
.catch((error) => {
alert(error.message);
});
};
return (…
);
};

Section 23: A (Pretty Deep Dive) Introduction to Next.js:

What is NextJS?:

React framework for production.

Key Feature 1: Built-in Server-side Rendering (Improved SEO!):

Allow us to pre-render react app on the server side.

How to create a Nexthjs project:

In terminal run - “npx create-next-app” , give a name for the project with “—name”
Navigate to the project and type (in terminal) - “npm run dev”

Adding First Pages:
Creating regular React component files in the pages folder.
The file name will be the path which the server get request
For example (“our-domain” = “localhost3000:”):

1. index.js - our-domain/
2. news.js - our-domain/news

Adding Nested Paths & Pages (Nested Routes):
Alternatively we can create folders (in the pages folder).
For example:

- Pages/news/index.js will be corresponds to - our-domain/news/
- Pages/news/something.js will be corresponds to - our-domain/news/something
- Pages/news/cities/madrid.js will be corresponds to - our-domain/news/cities/madrid

Creating Dynamic Pages (with Parameters):
With renaming the files with “[]” for example [something].js

Extracting Dynamic Parameter Values:
With a special hook - “import { useRouter } from ’next/router’” than:
Const router = useRouter();
router.query.our-name. (our-name is the name we chose to put in [])

Using CSS modules:

MeetupDetail.module.css MeetupDetail.js

components > meetups > MeetupDetail.js > ...

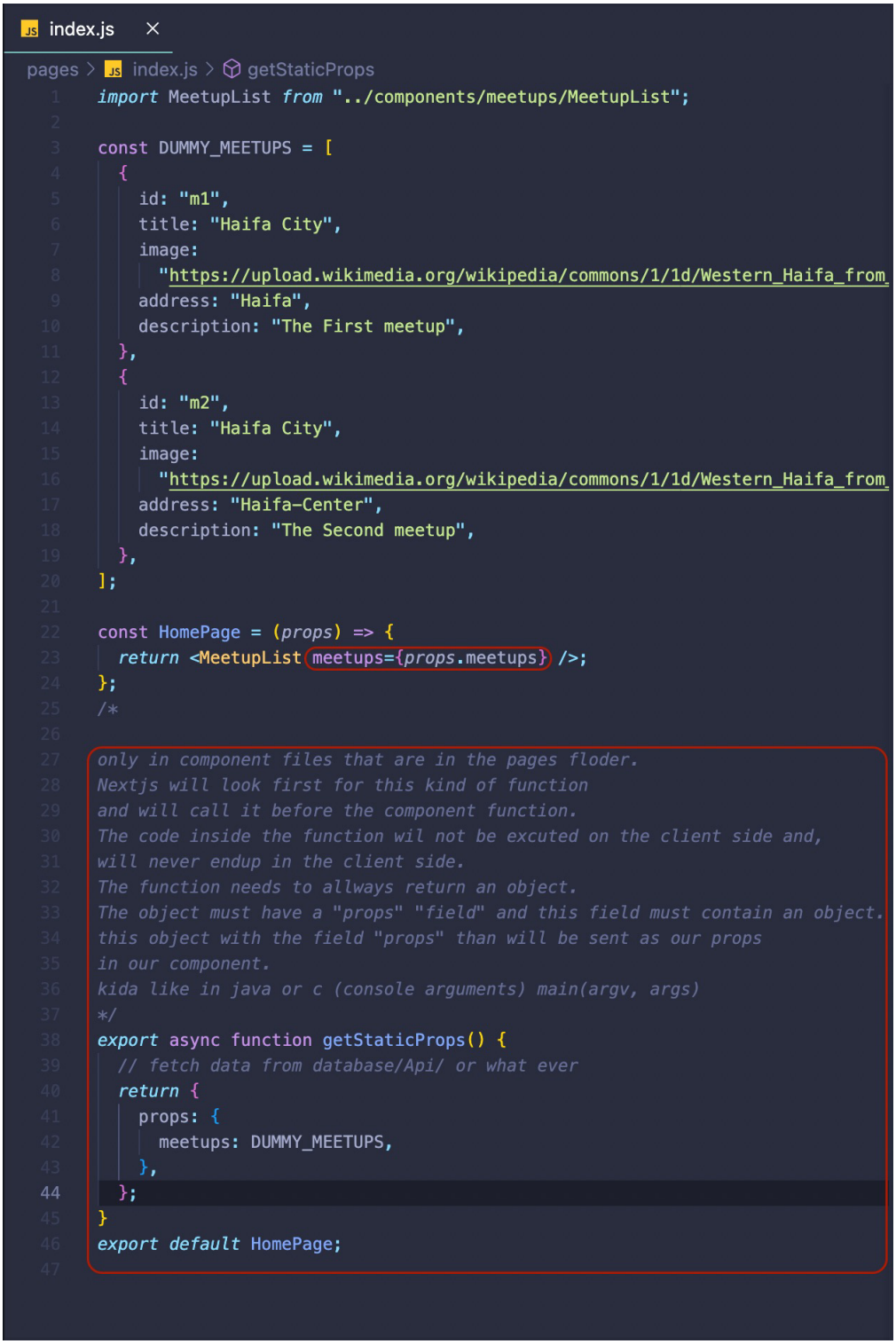
```
import classes from "./MeetupDetail.module.css";

const MeetupDetail = (props) => {
  return (
    <section className={classes.detail}>
      <img src={props.img} alt={props.title} />
      <h1>{props.title}</h1>
      <address>{props.address}</address>
      <p>{props.description}</p>
    </section>
  );
};
export default MeetupDetail;
```

MeetupDetail.module.css MeetupDetail.js

components > meetups > MeetupDetail.module.css >

```
.detail {
  text-align: center;
}
.detail img {
  width: 100%;
}
```

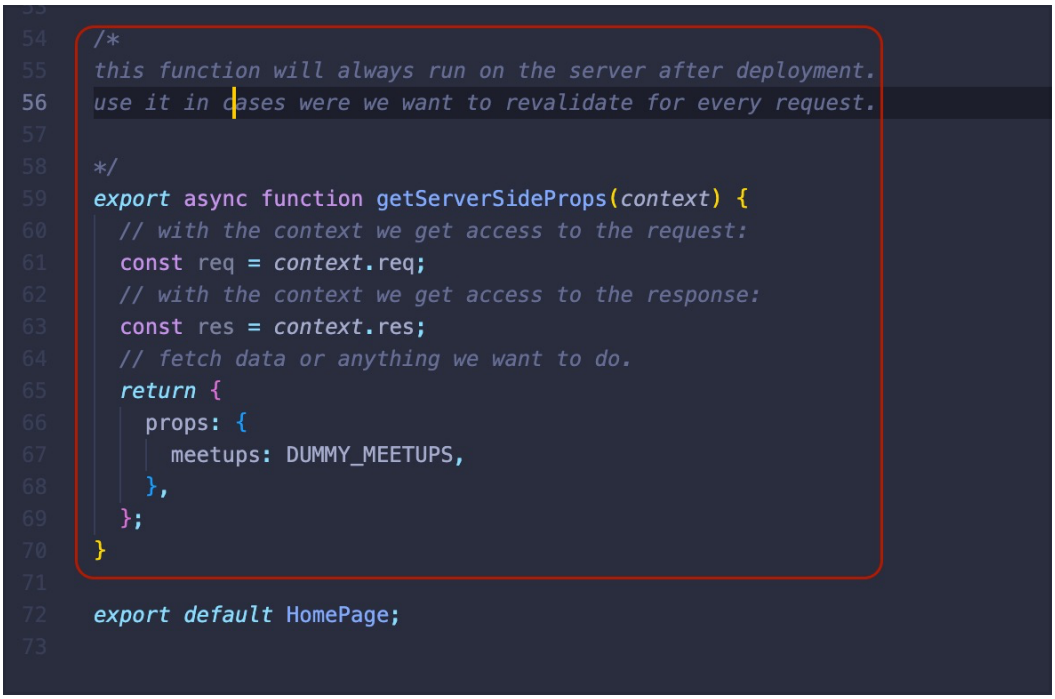
Data Fetching for Static Pages:



More on Static Site Generation (SSG):

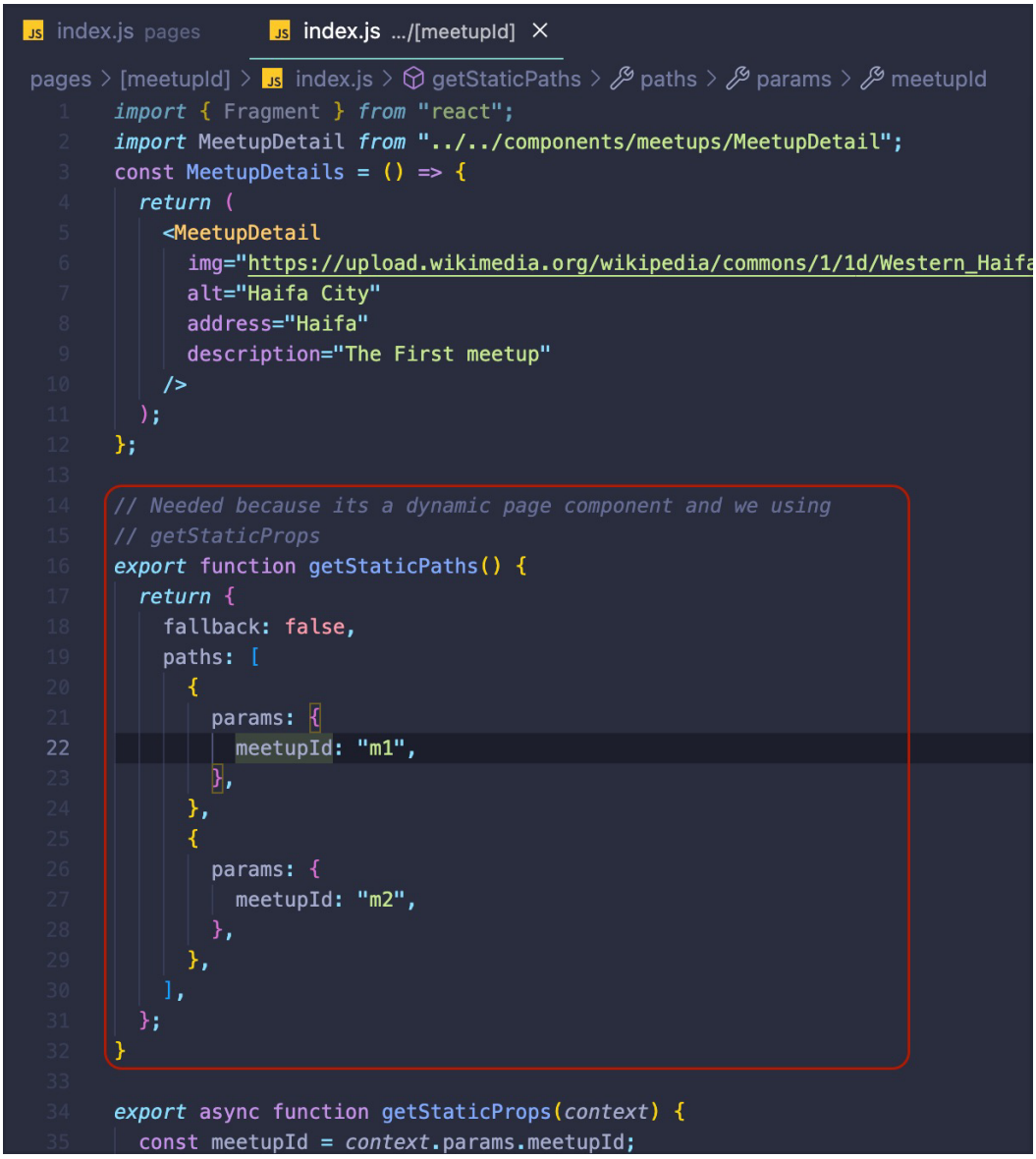
```
const HomePage = (props) => {
  return <MeetupList meetups={props.meetups} />;
};
/*

only in component files that are in the pages floder.
Nextjs will look first for this kind of function
and will call it before the component function.
The code inside the function wil not be excuted on the client side and,
will never endup in the client side.
The function needs to allways return an object.
The object must have a "props" "field" and this field must contain an object.
this object with the field "props" than will be sent as our props
in our component.
kida like in java or c (console arguments) main(argv, args).
The revalidate field handy in cases where we want to revalidate (get the data again).
sometimes the data we are working with can be out-dated quickly therefor
if we use revalidate (and as a value we need to give it a number)
we tell nextjs that we want to call getStaticProps every number of seconds
(the number we set as an argument. here we set it to 10 - every 10 seconds)
(only if requests come's to this path)
*/
export async function getStaticProps() {
  // fetch data from database/Api/ or what ever
  return {
    props: {
      meetups: DUMMY_MEETUPS,
    },
    revalidate: 10,
  };
}
export default HomePage;
```

Exploring Server-side Rendering (SSR) with "getServerSideProps"

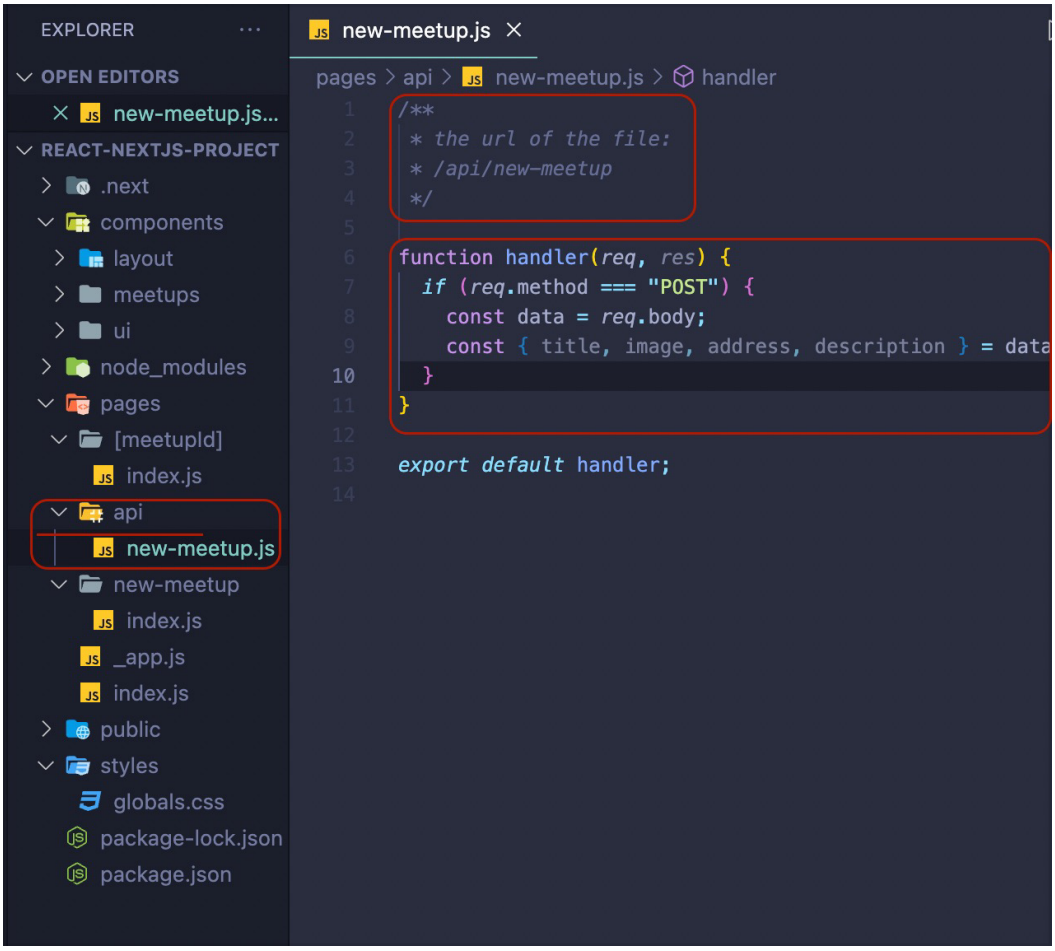


```
/*
this function will always run on the server after deployment.
use it in cases were we want to revalidate for every request.

*/
export async function getServerSideProps(context) {
  // with the context we get access to the request:
  const req = context.req;
  // with the context we get access to the response:
  const res = context.res;
  // fetch data or anything we want to do.
  return {
    props: {
      meetups: DUMMY_MEETUPS,
    },
  };
}

export default HomePage;
```

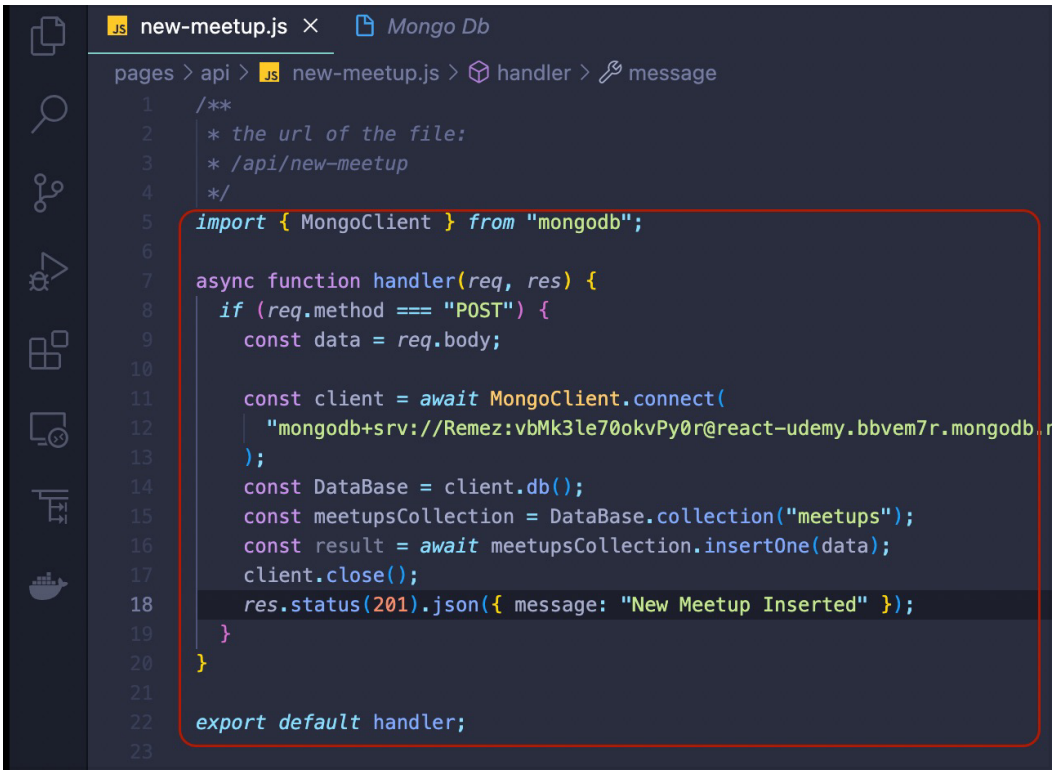
Preparing Paths with "getStaticPaths" & Working With Fallback Pages:



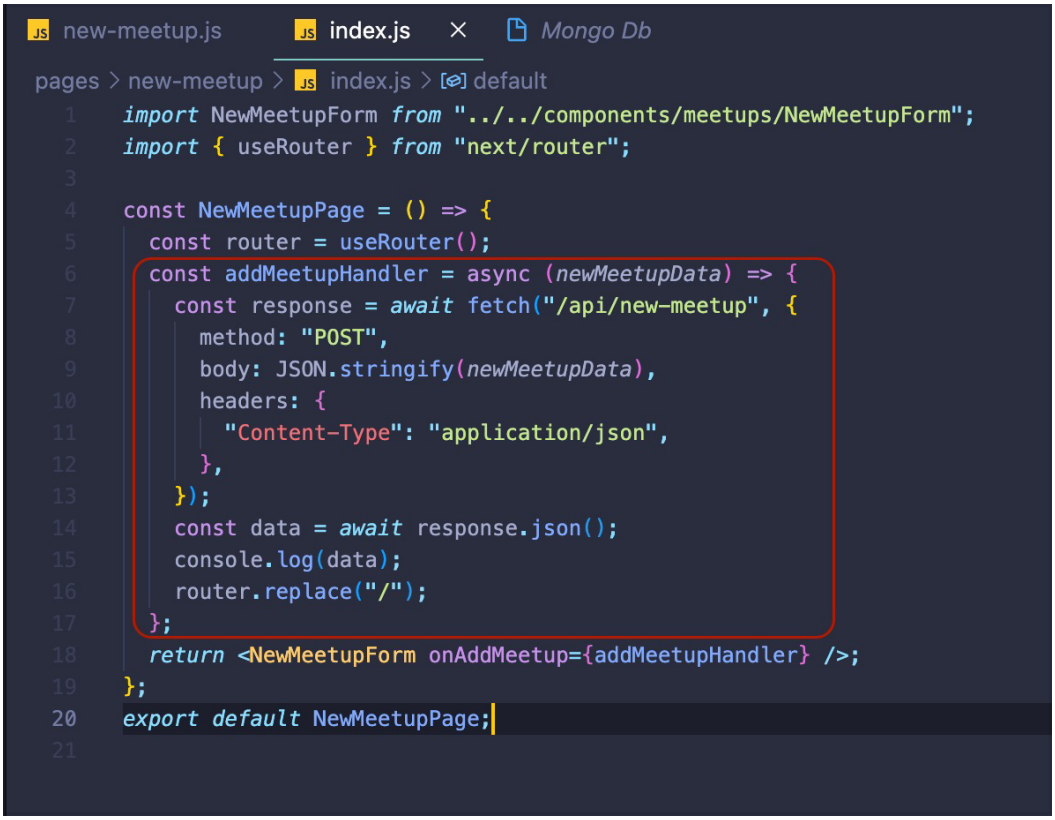
Introducing API Routes:



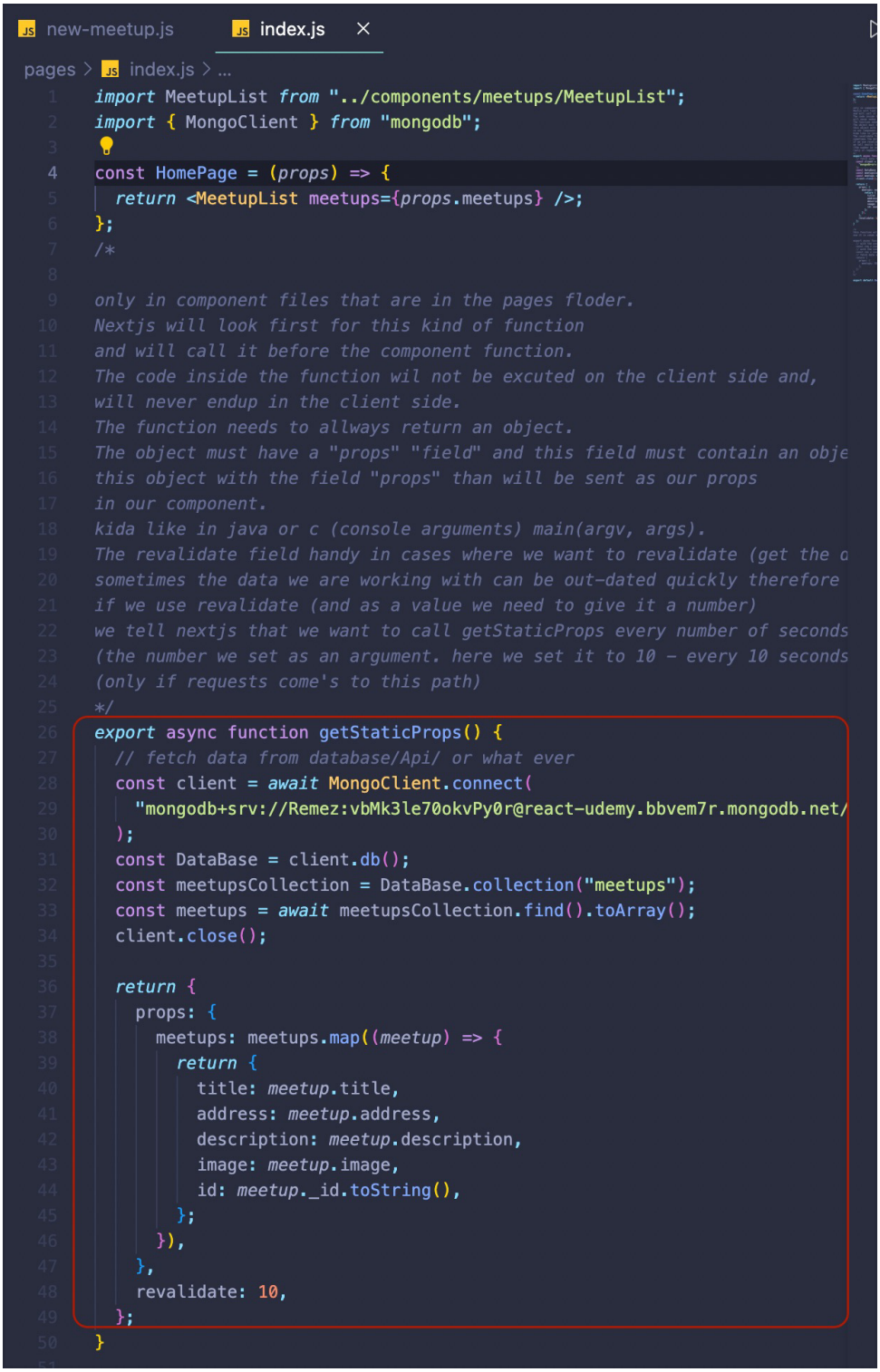
Working with MongoDB:



Sending Http Requests To Our API Routes:

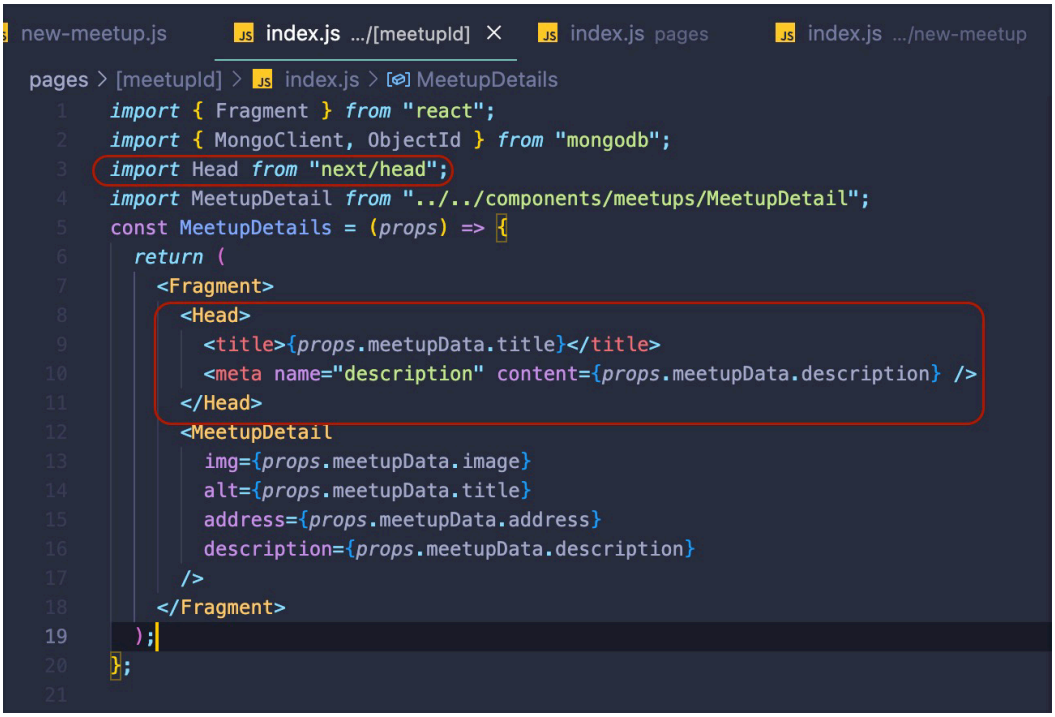


Getting Data From The Database:



Getting Meetup Details Data & Preparing Pages:

Adding "head" Metadata:



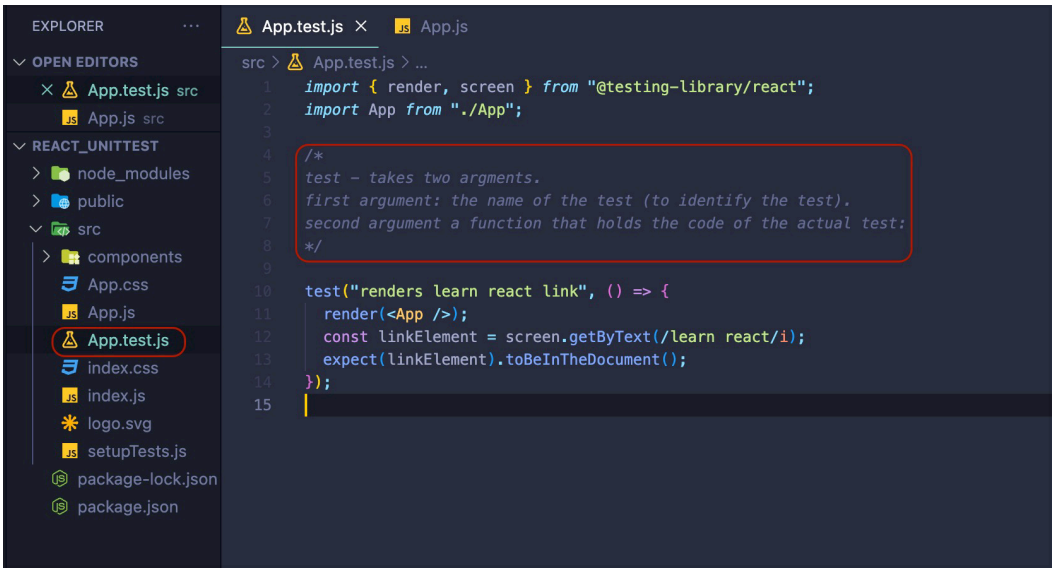
Section 25: Replacing Redux with React Hooks:

Finishing the Store Hook:

Section 26: Testing React Apps (Unit Tests):

Running a First Test:

Use - “npm test” to run our test.
The script will look for a file named “App.test.js” in this file we can write our tests.



Testing User Interaction & State:

Testing Connected Components:

Regular - can test with render nested components !

Output.js Greeting.js

src > components > Greeting.js > Greeting

```
import { useState } from "react";
import Output from "./Output";
const Greeting = () => {
  const [changedText, setChangedText] = useState(false);
  const changeTextHandler = () => {
    setChangedText(true);
  };
  return (
    <div>
      <h2>Hello World!</h2>
      {!changedText && <Output>Its good to see you!</Output>}
      {changedText && <Output>Change!</Output>}
      <button onClick={changeTextHandler}>Change Text</button>
    </div>
  );
};

export default Greeting;
```

Testing Asynchronous Code:

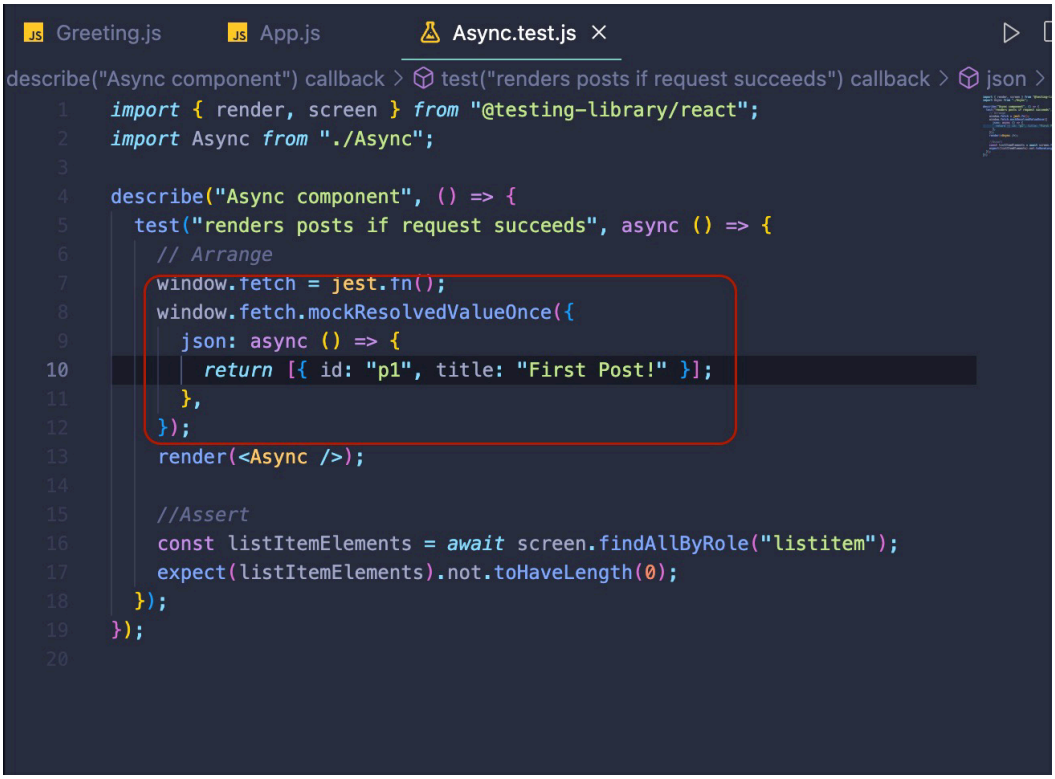
How to test when sending a Http request to an api (Asynchronous Code)

Greeting.js Async.js App.js Async.test.js

scribe("Async component") callback > test("renders posts if request succeeds") callback > listIte

```
import { render, screen } from "@testing-library/react";
import Async from "./Async";

describe("Async component", () => {
  test("renders posts if request succeeds", async () => {
    // Arrange
    render(<Async />);

    //Assert
    const listItemElements = await screen.findAllByRole("listitem");
    expect(listItemElements).not.toHaveLength(0);
  });
});
```

Working With Mocks:



Section 27: React + TypeScript:

To install typeScript for a specific project run in terminal: “nom install typescript --save-dev”.

To then compile the typeScript code run in terminal: “npx tsc”. (Without a configuration file)

To compile an individual file run in the terminal: “npx tsc [path to file]”.

Adding TypeScript

> Note: this feature is available with `react-scripts@2.1.0` and higher.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Installation

To start a new Create React App project with TypeScript, you can run:

```
npx create-react-app my-app --template typescript
```

Copy

Working with Props & TypeScript:

App.tsx Todos.tsx

src > components > Todos.tsx > ...

```tsx
const Todos: React.FC<{ items: string[] }> = (props) => {
  return (
    <ul>
      {props.items.map((item) => {
        return <li>{`${item}.`}</li>;
      })}
    </ul>
  );
};
export default Todos;
```

Adding a Data Model:

App.tsx Todos.tsx todo.ts

src > models > todo.ts > Todo > constructor

```ts
class Todo {
  id: string;
  text: string;
  constructor(todoText: string) {
    this.text = todoText;
    this.id = new Date().toISOString();
  }
}
export default Todo;
```

App.tsx × Todos.tsx todo.ts

src > App.tsx > ...

```tsx
import Todos from "./components/Todos";
import Todo from "./models/todo";
function App() {
  const todos = [new Todo("Remez"), new Todo("Ron"), new Todo("Avihai")];
  // const dummy_todos = ["Remez", "Dor", "Avihai", "Ron"];
  return (
    <div>
      <Todos items={todos} />
    </div>
  );
}

export default App;
```

App.tsx Todos.tsx × todo.ts

src > components > Todos.tsx > Todos > props.items.map() callback

```tsx
import Todo from "../models/todo";

const Todos: React.FC<{ items: Todo[] }> = (props) => {
  return (
    <ul>
      {props.items.map((item) => {
        return <li key={item.id}>{`name: ${item.text}.`}</li>;
      })}
    </ul>
  );
};
export default Todos;

```