

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.

```
JS ExpenseItem.js U X JS App.js M JS index.js M
src > components > JS ExpenseItem.js > ...
1  function ExpenseItem() {
2    return <h2>This is my first component!</h2>;
3  }
4
5  export default ExpenseItem;
6
```

Using the custom component inside another component.

```
import ExpenseItem from "./components/ExpenseItem";
function App() {
  return (
    <div>
      <ExpenseItem></ExpenseItem>
      <h3>This Will Be Amazing</h3>
      <p>By Remez David</p>
    </div>
  );
}

export default App;
```

Adding Basic CSS Styling:

```

JS ExpenseItem.css U      # ExpenseItem.css U X
src > components > # ExpenseItem.css > ExpenseItem.css
1 .expense-item {
2   display: flex;
3   justify-content: space-between;
4   align-items: center;
5   box-shadow: 0 2px 8px rgba(0, 0, 0,
6   padding: 0.5rem;
7   margin: 1rem 0;
8   border-radius: 12px;
9   background-color: #4b4b4b;
10 }
11
12 .expense-item__description {
13   display: flex;
14   flex-direction: column;
15   gap: 1rem;
16   align-items: flex-end;
17   flex-flow: column-reverse;
18   justify-content: flex-start;

```

```

JS ExpenseItem.js U X      # ExpenseItem.css U      JS App.js M      JS index.css M
src > components > JS ExpenseItem.js > ExpenseItem
1 import "./ExpenseItem.css";
2 function ExpenseItem() {
3   return (
4     <div className="expense-item">
5       <div>Mar 28th 2021</div>
6       <div className="expense-item__description">
7         <h2>Car Insurance</h2>
8         <div className="expense-item__price">$264.94</div>
9       </div>
10    </div>
11  );
12 }
13
14 export default ExpenseItem;
15

```

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

```
JS ExpenseItem.js ✘ # ExpenseItem.css ✘ JS App.js M
src > components > JS ExpenseItem.js > ⚡ ExpenseItem
1 import "./ExpenseItem.css";
2 function ExpenseItem() {
3     const expenseDate = new Date(2021, 2, 28);
4     const expenseTitle = "Car Insurance";
5     const expenseAmount = 265.73;
6
7     return (
8         <div className="expense-item">
9             <div>{expenseDate.toISOString()}</div>
10            <div className="expense-item__description">
11                <h2>{expenseTitle}</h2>
12                <div className="expense-item__price">${expenseAmount}</div>
13            </div>
14        </div>
15    );
16 }
17
18 export default ExpenseItem;
19
```

Passing Data via "props":

```
JS ExpenseItem.js ✘ # ExpenseItem.css ✘ JS App.js M
src > components > JS ExpenseItem.js > ⚡ ExpenseItem
1 import "./ExpenseItem.css";
2 function ExpenseItem(props) {
3     return (
4         <div className="expense-item">
5             <div>{props.date.toISOString()}</div>
6             <div className="expense-item__description">
7                 <h2>{props.title}</h2>
8                 <div className="expense-item__price">${props.amount}</div>
9             </div>
10        </div>
11    );
12 }
13
14 export default ExpenseItem;
15
```

```
JS ExpenseItem.js U      # ExpenseItem.css U      JS App.js M X
src > JS App.js > ⚙ App
1 import ExpenseItem from "./components/ExpenseItem";
2 function App() {
3   const expenses = [
4     {
5       id: "e1",
6       title: "Toilet Paper",
7       amount: 94.12,
8       date: new Date(2020, 7, 14),
9     },
10    { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },
11    {
12      id: "e3",
13      title: "Car Insurance",
14      amount: 294.67,
15      date: new Date(2021, 2, 28),
16    },
17    {
18      id: "e4",
19      title: "New Desk (Wooden)",
20      amount: 450,
21      date: new Date(2021, 5, 12),
22    },
23  ];
24  return (
25    <div>
26      <h3>This Will Be Amazing</h3>
27      <ExpenseItem
28        title={expenses[0].title}
29        amount={expenses[0].amount}
30        date={expenses[0].date}
31      ></ExpenseItem>
32    )
33 }
```

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 X
src > components > JS Card.js > ⚙ Card
1 import "./Card.css";
2 function Card(props) {
3   const classes = "card " + props.className;
4   return <div className={classes}>{props.children}</div>;
5 }
6
7 export default Card;
8
```

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

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

```
1 .card {  
2   border-radius: 12px;  
3   box-shadow: 0 1px 8px rgba(0, 0, 0, 0.25);  
4 }  
5
```

JS Expenses.js U X **JS Card.js** U

src > components > **JS Expenses.js** > Expenses

```
1 import ExpenseItem from './ExpenseItem';  
2 import './Expenses.css';  
3 import Card from './Card';  
4 function Expenses(props) {  
5   return (  
6     <Card className="expenses">  
7       <ExpenseItem  
8         title={props.items[0].title}  
9         amount={props.items[0].amount}  
10        date={props.items[0].date}  
11      />  
12      <ExpenseItem  
13        title={props.items[1].title}  
14        amount={props.items[1].amount}  
15        date={props.items[1].date}  
16      />  
17      <ExpenseItem  
18        title={props.items[2].title}  
19        amount={props.items[2].amount}  
20        date={props.items[2].date}  
21      />  
22      <ExpenseItem  
23        title={props.items[3].title}  
24        amount={props.items[3].amount}  
25        date={props.items[3].date}  
26      />  
27    </Card>  
28  );  
29}  
30  
31 export default Expenses;  
32
```

Working with "State":

JS ExpenseItem.js M X

src > components > Expenses > JS ExpenseItem.js > ⚡ ExpenseItem

```
1 import "./ExpenseItem.css";
2 import ExpenseDate from "./ExpenseDate";
3 import Card from "../UI/Card";
4 import React, { useState } from "react";
5
6 function ExpenseItem(props) {
7   // Array destructuring -> First argument is the var itself, second argument is function
8   // that called when this value is changed
9   const [title, setTitle] = useState(props.title);
10  const clickHandler = () => {
11    setTitle("Updated!");
12  };
13
14  return (
15    <Card className="expense-item">
16      <ExpenseDate date={props.date} />
17      <div className="expense-item__description">
18        <h2>{title}</h2>
19        <div className="expense-item__price">$ {props.amount}</div>
20      </div>
21      <button onClick={clickHandler}>Change Title</button>
22    </Card>
23  );
24}
25
26 export default ExpenseItem;
27
```

Listening to User Input:

JS ExpenseItem.js M JS NewExpense.js U JS App.js M JS ExpenseForm.js U X

src > components > NewExpense > JS ExpenseForm.js > ⚡ ExpenseForm > [Θ] titleChangeHandler

```
1 import "./ExpenseForm.css";
2
3 function ExpenseForm() {
4   const titleChangeHandler = (event) => [
5     // Getting the value the user entered !
6     console.log(event.target.value);
7   ];
8   return (
9     <form>
10       <div className="new-expense__controls">
11         <div className="new-expense__control">
12           <label>Title</label>
13           <input type="text" onChange={titleChangeHandler} />
14         </div>
15         <div className="new-expense__control">
16           <label>Amount</label>
17           <input type="number" min="0.01" step="0.01" />
18         </div>
19         <div className="new-expense__control">
20           <label>Date</label>
21           <input type="date" min="2019-01-01" max="2022-12-31" />
22         </div>
23       </div>
24       <div className="new-expense__actions">
25         <button type="submit">Add Expense</button>
26       </div>
27     </form>
28   );
29}
```

Handling Form Submission:

```
JS ExpenseForm.js U X
src > components > NewExpense > JS ExpenseForm.js > ExpenseForm > [o] submit
  ...
17  const dateChangeHandler = (event) => {
18    setEnteredDate(event.target.value);
19  };
20  const submitHandler = (event) => {
21    event.preventDefault();
22    const expenseData = {
23      title: enteredTitle,
24      amount: enteredAmount,
25      date: new Date(enteredDate),
26    };
27    console.log(expenseData);
28  };
29
30  return (
31    <form onSubmit={submitHandler}>
32      <div className="new-expense__controls">
33        <div className="new-expense__control">
34          <label>Title</label>
35          <input type="text" onChange={titleChangeHandler} />
36        </div>
37        <div className="new-expense__control">
38          <label>Amount</label>
39          <input
40            type="number"
41            min="0.01"
42            step="0.01"
43          />
44      </div>
45    </form>
46  );
47
```

Adding Two-Way Binding:

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

```
JS ExpenseForm.js U X
src > components > NewExpense > JS ExpenseForm.js > ExpenseForm
  19 |   };
  20 |   const submitHandler = (event) => {
  21 |     event.preventDefault();
  22 |     const expenseData = {
  23 |       title: enteredTitle,
  24 |       amount: enteredAmount,
  25 |       date: new Date(enteredDate),
  26 |     };
  27 |     console.log(expenseData);
  28 |     setEnteredAmount("");
  29 |     setEnteredDate("");
  30 |     setEnteredTitle("");
  31 |   };
  32 |
  33 |   return (
  34 |     <form onSubmit={submitHandler}>
  35 |       <div className="new-expense__controls">
  36 |         <div className="new-expense__control">
  37 |           <label>Title</label>
  38 |           <input
  39 |             type="text"
  40 |             onChange={titleChangeHandler}
  41 |             value={enteredTitle}
  42 |           />
  43 |         </div>
  44 |         <div className="new-expense__control">
  45 |           <label>Amount</label>
  46 |           <input
  47 |             type="number"
  48 |             min="0.01"
  49 |             step="0.01"
  50 |             onChange={amountChangeHandler}
  51 |             value={enteredAmount}
  52 |           />
  53 |         </div>
  54 |         <div className="new-expense__control">
  55 |           <label>Date</label>
  56 |           <input
  57 |             type="date"
  58 |             min="2019-01-01"
  59 |             max="2022-12-31"
  60 |             onChange={dateChangeHandler}
  61 |             value={enteredDate}
  62 |           />
  63 |         </div>
  64 |       </div>
  65 |       <div className="new-expense__actions">
  66 |         <button type="submit">Add Expense</button>
```

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

JS ExpenseForm.js U JS NewExpense.js U X

src > components > NewExpense > **JS** NewExpense.js > NewExpense > [?] saveExpenseData

```
1 import './NewExpense.css';
2 import ExpenseForm from './ExpenseForm';
3 function NewExpense() {
4   const saveExpenseDataHandler = (enteredExpenseData) => {
5     const expenseData = {
6       ...enteredExpenseData,
7       id: Math.random().toString(),
8     };
9     console.log[expenseData];
10   };
11   return (
12     <div className="new-expense">
13       <ExpenseForm onSaveExpenseData={saveExpenseDataHandler}>
14     </div>
15   );
16 }
17
18 export default NewExpense;
19
```

JS ExpenseForm.js × JS NewExpense.js

```
src > components > NewExpense > JS ExpenseForm.js > ExpenseForm > [submit] ExpenseForm.js
```

```
1 import './ExpenseForm.css';
2 import React, { useState } from "react";
3
4 function ExpenseForm(props) {
5   const [enteredTitle, setEnteredTitle] = useState("");
6   const [enteredAmount, setEnteredAmount] = useState("");
7   const [enteredDate, setEnteredDate] = useState("");
8
9   const titleChangeHandler = (event) => {
10     // Getting the value the user entered -> event.target.value!
11     setEnteredTitle(event.target.value);
12   };
13
14   const amountChangeHandler = (event) => {
15     setEnteredAmount(event.target.value);
16   };
17   const dateChangeHandler = (event) => {
18     setEnteredDate(event.target.value);
19   };
20   const submitHandler = (event) => {
21     event.preventDefault();
22     const expenseData = {
23       title: enteredTitle,
24       amount: enteredAmount,
25       date: new Date(enteredDate),
26     };
27     props.onSaveExpenseData(expenseData);
28     setEnteredAmount("");
29     setEnteredDate("");
30     setEnteredTitle("");
31   };
32 }
```

Rendering Lists of Data:

```
src > components > Expenses > Expenses.js > Expenses
1 import ExpenseItem from "./ExpenseItem";
2 import ExpensesFilter from "./ExpensesFilter";
3 import "./Expenses.css";
4 import Card from "../UI/Card";
5 import React, { useState } from "react";
6 function Expenses(props) {
7   const [filteredYear, setFilteredYear] = useState("2020");
8   const filterChangeHandler = (selectedYear) => {
9     setFilteredYear(selectedYear);
10  };
11  return [
12    <div>
13      <Card className="expenses">
14          <ExpensesFilter
15              selected={filteredYear}
16              onYearChange={filterChangeHandler}
17          />
18          {props.items.map((expense) => (
19              <ExpenseItem
20                  title={expense.title}
21                  amount={expense.amount}
22                  date={expense.date}
23              />
24          ))}
25      </Card>
26    </div>
27  ];
28}
```

The map function documentation.

Array.prototype.map()

The `map()` method **creates a new array** populated with the results of calling a provided function on every element in the calling array.

Try it

JavaScript Demo: Array.map()

```
1 const array1 = [1, 4, 9, 16];
2 // pass a function to map
3 const map1 = array1.map(x => x * 2);
4
5 console.log(map1);
6 // expected output: Array [2, 8, 18, 32]
```

Using Stateful Lists:

```

JS Expenses.js M      JS App.js M X    JS NewExpense.js

src > JS App.js > ⚡ App > 🐛 addExpenseHandler > ⚡ setExpenses() callback
1 import Expenses from "./components/Expenses/Expenses.js";
2 import NewExpense from "./components/NewExpense/NewExpense.js";
3 import React, { useState } from "react";
4
5 const DUMMY_EXPENSES = [
6   {
7     id: "e1",
8     title: "Toilet Paper",
9     amount: 94.12,
10    date: new Date(2020, 7, 14),
11  },
12  { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },
13  {
14    id: "e3",
15    title: "Car Insurance",
16    amount: 294.67,
17    date: new Date(2021, 2, 28),
18  },
19  {
20    id: "e4",
21    title: "New Desk (Wooden)",
22    amount: 450,
23    date: new Date(2021, 5, 12),
24  },
25];
26
27 function App() {
28   const [expenses, setExpenses] = useState(DUMMY_EXPENSES);
29   const addExpenseHandler = (newExpenseData) => {
30     // Because are new state depends on the last state we should use
31     // The function approach likewise:
32     setExpenses((prevExpenses) => {
33       return [newExpenseData, ...prevExpenses];
34     });
35   };
36   return (
37     <div>
38       <NewExpense onAddExpense={addExpenseHandler} />
39       <Expenses items={expenses} />
40     </div>
41   );
42 }
43
44 export default App;
45

```

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).

JS Expenses.js M X JS App.js M

```
src > components > Expenses > JS Expenses.js > Expenses.js
1 import ExpenseItem from "./ExpenseItem";
2 import ExpensesFilter from "./ExpensesFilter";
3 import "./Expenses.css";
4 import Card from "../UI/Card";
5 import React, { useState } from "react";
6 function Expenses(props) {
7     const [filteredYear, setFilteredYear] = useState("2021");
8     const filterChangeHandler = (selectedYear) =>
9         setFilteredYear(selectedYear);
10    };
11    return (
12        <div>
13            <Card className="expenses">
14                <ExpensesFilter
15                    selected={filteredYear}
16                    onYearChange={filterChangeHandler}>
17                />
18                {props.items.map((expense) => (
19                    <ExpenseItem
20                        key={expense.id}
21                        title={expense.title}
22                        amount={expense.amount}
23                        date={expense.date}>
24                    />
25                )));
26            </Card>
27        </div>
28    );
}
```

Outputting Conditional Content:

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

JS Expenses.js M X JS ExpensesFilter.js M JS App.js M

src > components > Expenses > **JS Expenses.js** > Expenses

```
10  };
11  const filteredExpenses = props.items.filter((expense) =>
12  |   return expense.date.getFullYear().toString() === filteredYear
13  );
14  return (
15  |   <div>
16  |       <Card className="expenses">
17  |           <ExpensesFilter
18  |               selected={filteredYear}
19  |               onYearChange={filterChangeHandler}
20  |           />
21  |           {filteredExpenses.length === 0 ? (
22  |               <p>No Expenses Found.</p>
23  |           ) : (
24  |               filteredExpenses.map((expense) => (
25  |                   <ExpenseItem
26  |                       key={expense.id}
27  |                       title={expense.title}
28  |                       amount={expense.amount}
29  |                       date={expense.date}
30  |                   />
31  |               ))
32  |           )
33  |       </Card>
34  |   </div>
35  );
36
37 }
```

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

JS Expenses.js M X JS ExpensesFilter.js M JS App.js M

src > components > Expenses > JS Expenses.js > Expenses

```
12 |     return expense.date.getFullYear().toString() === filteredYear;
13 |   );
14 |
15 |   let expensesContent = <p>No Expenses Found.</p>;
16 |
17 |   if (filteredExpenses.length > 0) {
18 |     expensesContent = filteredExpenses.map((expense) => (
19 |       <ExpenseItem
20 |         key={expense.id}
21 |         title={expense.title}
22 |         amount={expense.amount}
23 |         date={expense.date}
24 |       />
25 |     )));
26 |   }
27 |
28 |   return (
29 |     <div>
30 |       <Card className="expenses">
31 |         <ExpensesFilter
32 |           selected={filteredYear}
33 |           onYearChange={filterChangeHandler}
34 |         />
35 |         {expensesContent}
36 |       </Card>
37 |     </div>
38 |   );
39 }
```

Setting CSS Classes Dynamically:

JS CourseInput.js M CSS CourseInput.css M X

src > components > CourseGoals > CourseInput > CSS CourseInput.css

```
24 |
25 |
26 |   .form-control.invalid input {
27 |     border-color: red;
28 |     background: #ffd7d7;
29 |   }
30 |
31 |   .form-control.invalid label {
32 |     color: red;
33 |   }
34 |
```

```
src > components > CourseGoals > CourseInput > CourseInput.js > CourseInput
13  ;
14
15  const formSubmitHandler = (event) => {
16    event.preventDefault();
17    if (enteredValue.trim() === "") {
18      setIsValid(false);
19      return;
20    }
21    props.onAddGoal(enteredValue);
22  };
23
24  return (
25    <form onSubmit={formSubmitHandler}>
26      <div className={`form-control ${!isValid ? "invalid" : "valid"}`}>
27        <label>Course Goal</label>
28        <input type="text" onChange={goalInputChangeHandler} />
29      </div>
30      <Button type="submit">Add Goal</Button>
31    </form>
32  );
33
34
35  export default CourseInput;
```

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.

```
src > components > UI > Button > Button.js > ...
1 // import "./Button.css";
2 import styled from "styled-components";
3
4 // "styled" is an object. "button" is function in that object.
5 // Instead of calling the method like "()" we use ``.
6
7 const Button = styled.button`
8   font: inherit;
9   padding: 0.5rem 1.5rem;
10  border: 1px solid #8b005d;
11  color: white;
12  background: #8b005d;
13  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
14  cursor: pointer;
15
16  &:focus {
17    outline: none;
18  }
19
20  &:hover,
21  &:active {
22    background: #ac0e77;
23    border-color: #ac0e77;
24    box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
25  }
26`;
27
28 // const Button = props => {
29 //   return (
30 //     <button type={props.type} className="button" onClick={props.onClick}>
31 //       {props.children}
32 //     </button>
33 //   );
34 // };
35
36 export default Button;
```



```

src > components > CourseGoals > CourseInput > CourseInput.js > CourseInput
1 import React, { useState } from "react";
2 import styled from "styled-components";
3 import Button from "../../UI/Button/Button";
4 import "./CourseInput.css";
5
6 const FormControl = styled.div`
7   margin: 0.5rem 0;
8
9   & label {
10     font-weight: bold;
11     color: ${(props) => (props.invalid ? "red" : "black")};
12     display: block;
13     margin-bottom: 0.5rem;
14   }
15
16   & input {
17     display: block;
18     width: 100%;
19     border: 1px solid ${(props) => (props.invalid ? "red" : "#ccc")};
20     background: ${(props) => (props.invalid ? "#ffd7d7" : "transparent")};
21     font: inherit;
22     line-height: 1.5rem;
23     padding: 0 0.25rem;
24   }
25
26   & input:focus {
27     outline: none;
28     background: #fad0ec;
29     border-color: #8b005d;
30   }
31 `;
32
33 const CourseInput = (props) => {
34   const [enteredValue, setEnteredValue] = useState("");
35   const [isValid, setIsValid] = useState(true);
36
37   const goalInputChangeHandler = (event) => {
38     if (event.target.value.trim() !== "") setIsValid(true);
39     setEnteredValue(event.target.value);
40   };
41
42   const formSubmitHandler = (event) => {
43     event.preventDefault();
44     if (enteredValue.trim() === "") {
45       setIsValid(false);
46       return;
47     }
48     props.onAddGoal(enteredValue);
49   };
50
51   return (
52     <form onSubmit={formSubmitHandler}>
53       <FormControl invalid={isValid}>
54         <label>Course Goal</label>
55         <input type="text" onChange={goalInputChangeHandler} />
56       </FormControl>
57       <Button type="submit">Add Goal</Button>
58     </form>
59   );
60 };

```

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 !

src > components > UI > Button > `Button.js`

```
1 import styles from "./Button.module.css";  
2  
3 // "styled" is an object. "button" is function in that object.  
4 // Instead of calling the method like "() we use ``.  
5  
6 // const Button = styled.button`  
7 //   width: 100%;  
8 //   font: inherit;  
9 //   padding: 0.5rem 1.5rem;  
10 //   border: 1px solid #8b005d;  
11 //   color: white;  
12 //   background: #8b005d;  
13 //   box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);  
14 //   cursor: pointer;  
15  
16 //   @media (min-width: 768px) {  
17 //     width: auto;  
18 //   }  
19  
20 //   &:focus {  
21 //     outline: none;  
22 //   }  
23  
24 //   &:hover,  
25 //   &:active {  
26 //     background: #ac0e77;  
27 //     border-color: #ac0e77;  
28 //     box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);  
29 //   }  
30 // `;  
31  
32 const Button = (props) => {  
33   return (  
34     <button type={props.type} className={styles.button} onClick={props.onClick}>  
35       {props.children}  
36     </button>  
37   ).
```

```
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).

```
JS AddUser.js      JS ErrorModal.js      JS Wrapper.js X
src > components > Helper > JS Wrapper.js > [?] Wrapper
1  const Wrapper = (props) => [
2    return props.children;
3  };
4
5  export default Wrapper;
6
```

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".

```
public > index.html > html > body > div#overlay-root
  19   Notice the use of %PUBLIC_URL% in the tags above.
  20   It will be replaced with the URL of the `public` folder during the build.
  21   Only files inside the `public` folder can be referenced from the HTML.
  22
  23   Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
  24   work correctly both with client-side routing and a non-root public URL.
  25   Learn how to configure a non-root public URL by running `npm run build`
  26   -->
  27   <title>React App</title>
  28 </head>
  29 <body>
  30   <noscript>You need to enable JavaScript to run this app.</noscript>
  31   <div id="backdrop-root"></div>
  32   <div id="overlay-root"></div>
  33   <div id="root"></div>
  34   <!--
  35   This HTML file is a template.
  36   If you open it directly in the browser, you will see an empty page.
```

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

The screenshot shows a code editor with several tabs at the top: AddUser.js, ErrorModal.js (which is the active tab), index.html, and App.js. The code in ErrorModal.js is as follows:

```
src > components > UI > ErrorModal.js > [?] ErrorModal
1 import Card from "./Card";
2 import Button from "./Button";
3 import styles from "./ErrorModal.module.css";
4 import ReactDOM from "react-dom";
5
6 const Backdrop = (props) => {
7   return <div className={styles.backdrop} onClick={props.onConfirm}>
8 };
9 const ModalOverlay = (props) => {
10   return (
11     <Card className={styles.modal}>
12       <header className={styles.header}>
13         <h2>{props.title}</h2>
14       </header>
15       <div className={styles.content}>
16         <p>{props.message}</p>
17       </div>
18       <footer className={styles.actions}>
19         <Button onClick={props.onConfirm}>OK</Button>
20       </footer>
21     </Card>
22   );
23 };
24 const ErrorModal = (props) => {
25   return (
26     <>
27       <ReactDOM.createPortal(
28         <Backdrop onClick={props.onConfirm} />,
29         document.getElementById("backdrop-root")
30       )>
31     </>
32   );
33 };
34
35 export default ErrorModal;
36
```

Working with "ref"s:

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

```

import React, { useState, useRef } from "react";
import Card from "./UI/Card";
import Button from "./UI/Button";
import styles from "./AddUser.module.css";
import ErrorModal from "./ErrorModal";
// https://www.w3schools.com/html/html_form_input_types.asp#text
const AddUser = () => {
  const nameInputRef = useRef();
  const ageInputRef = useRef();
  const [error, setError] = useState();
  const addressHandler = (event) => {
    event.preventDefault();
    const enteredUserName = nameInputRef.current.value;
    const enteredUserAge = ageInputRef.current.value;
    if (enteredUserName === "" || enteredUserAge === "") {
      setError({
        title: "Invalid Input",
        message: "Please enter a valid name and age!",
      });
      return;
    }
    if (+enteredUserAge < 1) {
      setError({
        title: "Invalid Age",
        message: "Please enter a valid age (> 0)",
      });
      return;
    }
    // console.log(enteredUserName, enteredUserAge);
    props.onAddUser(enteredUserName, enteredUserAge);
    randomInputRef.current.value = "";
    ageInputRef.current.value = "";
  };
  const handleBlur = (event) => {
    setError(null);
  };
  return (
    <div>
      <ErrorModal error={error} onConfirm={addressHandler}>
        <Card>
          <form>
            <label htmlFor="username">User Name:</label>
            <input ref={nameInputRef} type="text" id="username"/>
            <label htmlFor="age">Age (Years):</label>
            <input ref={ageInputRef} type="number" id="age" ref={ageInputRef}/>
            <button type="submit">Add User</button>
          </form>
        </Card>
      </ErrorModal>
    </div>
  );
}

```

refs

What are "Side Effects" & Introducing useEffect:

Store data in browser storage, sending http requests to backend server, set and manage 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_execute, 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).

```
js App.js  X
src > js App.js > ⚡ App
1 import React, { useState, useEffect } from "react";
2
3 import Login from "./components/Login/Login";
4 import Home from "./components/Home/Home";
5 import MainHeader from "./components/MainHeader/MainHeader";
6
7 function App() {
8   const [isLoggedIn, setIsLoggedIn] = useState(false);
9
10  useEffect(() => {
11    const userLogInStatus = localStorage.getItem("isLoggedIn");
12    if (userLogInStatus === "True") {
13      setIsLoggedIn(true);
14    }
15  }, []);
16
17  const loginHandler = (email, password) => {
18    // We should of course check email and password
19    // But it's just a dummy/ demo anyways
20    localStorage.setItem("isLoggedIn", "True");
21    setIsLoggedIn(true);
22  };
23
24  const logoutHandler = () => {
25    setIsLoggedIn(false);
26  };
27
28  return (
29    <React.Fragment>
30      <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />
31      <main>
```

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!

```
App.js      Login.js X
src > components > Login > Login.js > Login > passwordChangeHandler
1 import React, { useState, useEffect } from "react";
2
3 import Card from "../UI/Card/Card";
4 import classes from "./Login.module.css";
5 import Button from "../UI/Button/Button";
6
7 const Login = (props) => {
8   const [enteredEmail, setEnteredEmail] = useState("");
9   const [emailIsValid, setEmailIsValid] = useState();
10  const [enteredPassword, setEnteredPassword] = useState("");
11  const [passwordIsValid, setPasswordIsValid] = useState();
12  const [formIsValid, setFormIsValid] = useState(false);
13
14  useEffect(() => {
15    setEmailIsValid(
16      enteredEmail.includes("@") && enteredPassword.trim().length > 6
17    );
18  }, [enteredEmail, enteredPassword]);
```

Using the useEffect Cleanup Function:

```
useEffect(() => {
  const identifier = setTimeout(() => {
    console.log("Checking for validity!");
    setFormIsValid(
      enteredEmail.includes "@" && enteredPassword.trim().length > 6
    );
  }, 500);
  return () => {
    console.log("CLEANUP");
    clearTimeout(identifier);
  };
}, [enteredEmail, enteredPassword]);
```

introducing useReducer & Reducers In General:

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

Using the useReducer() Hook:

```
JS Login.js X
src > components > Login > JS Login.js > Login
1 import React, { useState, useEffect, useReducer } from "react";
2
3 import Card from "../UI/Card/Card";
4 import classes from "./Login.module.css";
5 import Button from "../UI/Button/Button";
6
7 const emailReducer = (state, action) => {
8   if (action.type === "USER_INPUT") {
9     return { value: action.val, isValid: action.val.includes("@") };
10  }
11  if (action.type === "INPUT_BLUR") {
12    return { value: state.value, isValid: state.value.includes("@") };
13  }
14  return { value: "", isValid: false };
15};
16
17 const Login = (props) => {
18   // const [enteredEmail, setEnteredEmail] = useState("");
19   // const [emailIsValid, setEmailIsValid] = useState();
20   const [enteredPassword, setPasswordEnteredPassword] = useState("");
21   const [passwordIsValid, setPasswordisValid] = useState();
22   const [formIsValid, setFormIsValid] = useState(false);
23   const [emailState, dispatchEmail] = useReducer(emailReducer, {
24     value: "",
25     isValid: null,
26   });
27
28   // useEffect(() => {
29
30     const emailChangeHandler = (event) => {
31       dispatchEmail({ type: "USER_INPUT", val: event.target.value });
32
33       setPasswordEnteredPassword(enteredPassword.trim().length > 6);
34     };
35
36     const passwordChangeHandler = (event) => {
37       setPasswordEnteredPassword(event.target.value);
38
39       setFormIsValid(emailState.isValid && event.target.value.trim().length > 6);
40     };
41
42     const validateEmailHandler = () => {
43       dispatchEmail({ type: "INPUT_BLUR" });
44     };
45
46     const validatePasswordHandler = () => {
47       setPasswordisValid(enteredPassword.trim().length > 6);
48     };
49
50     const submitHandler = (event) => {
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
```

Using the React Context API:

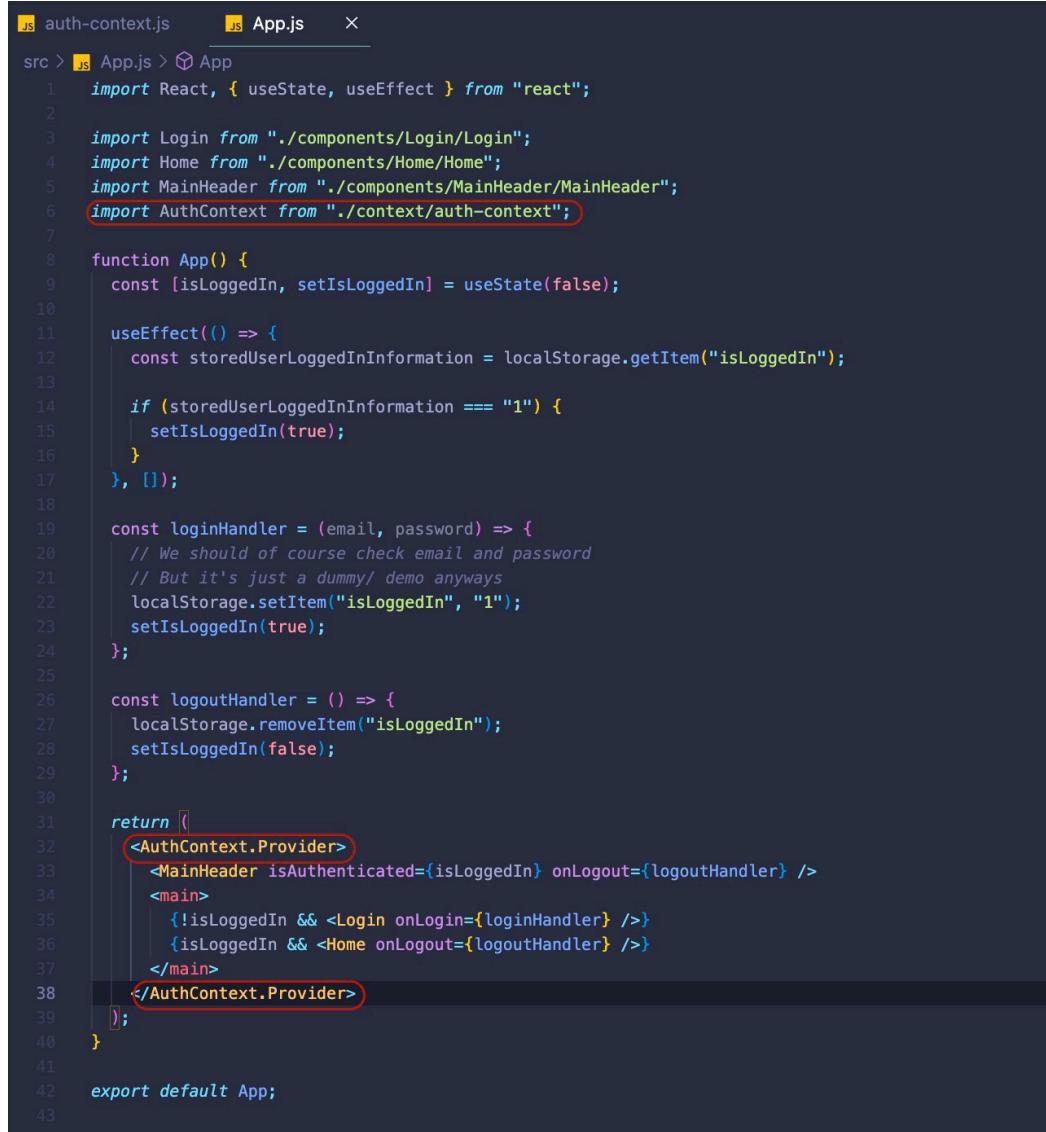
Creating a context object

```
JS auth-context.js X
src > context > JS auth-context.js > ...
1 const AuthContext = React.createContext({
2   isLoggedIn: false,
3 });
4 export default AuthContext;
5
```

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.



```
src > App.js > App
  1 import React, { useState, useEffect } from "react";
  2
  3 import Login from "./components/Login/Login";
  4 import Home from "./components/Home/Home";
  5 import MainHeader from "./components/MainHeader/MainHeader";
  6 import AuthContext from "./context/auth-context";
  7
  8 function App() {
  9   const [isLoggedIn, setIsLoggedIn] = useState(false);
 10
 11   useEffect(() => {
 12     const storedUserLoggedInInformation = localStorage.getItem("isLoggedIn");
 13
 14     if (storedUserLoggedInInformation === "1") {
 15       setIsLoggedIn(true);
 16     }
 17   }, []);
 18
 19   const loginHandler = (email, password) => {
 20     // We should of course check email and password
 21     // But it's just a dummy/ demo anyways
 22     localStorage.setItem("isLoggedIn", "1");
 23     setIsLoggedIn(true);
 24   };
 25
 26   const logoutHandler = () => {
 27     localStorage.removeItem("isLoggedIn");
 28     setIsLoggedIn(false);
 29   };
 30
 31   return (
 32     <AuthContext.Provider>
 33       <MainHeader isAuthenticated={isLoggedIn} onLogout={logoutHandler} />
 34       <main>
 35         {!isLoggedIn && <Login onLogin={loginHandler} />}
 36         {isLoggedIn && <Home onLogout={logoutHandler} />}
 37       </main>
 38     </AuthContext.Provider>
 39   );
 40 }
 41
 42 export default App;
 43
```

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

```
auth-context.js      App.js      Navigation.js X
src > components > MainHeader > Navigation.js > Navigation

3
4 import classes from "./Navigation.module.css";
5
6 const Navigation = [props] => {
7   return (
8     <AuthContext.Consumer>
9       {(context) => {
10         return (
11           <nav className={classes.nav}>
12             <ul>
13               {context.isLoggedIn && (
14                 <li>
15                   <a href="/">Users</a>
16                 </li>
17               )}
18               {context.isLoggedIn && (
19                 <li>
20                   <a href="/">Admin</a>
21                 </li>
22               )}
23               {context.isLoggedIn && (
24                 <li>
25                   <button onClick={context.onLogout}>Logout</button>
26                 </li>
27               )}
28             </ul>
29           </nav>
30         );
31       }
32     </AuthContext.Consumer>
33   );
34 };
35
36 export default Navigation;
```

The more elegant solution is to use the "useContext" Hook as follows:

JS auth-context.js JS App.js X JS Home.js

src > JS App.js > App > onLogOut

```
24  };
25
26  const logoutHandler = () => {
27    localStorage.removeItem("isLoggedIn");
28    setIsLoggedIn(false);
29  };
30
31  return (
32    <AuthContext.Provider
33      value={{
34        isLoggedIn: isLoggedIn,
35        onLogout: logoutHandler,
36        onLogin: loginHandler,
37      }}>
38      <MainHeader />
39      <main>
40        {!isLoggedIn && <Login />}
41        {isLoggedIn && <Home />}
42      </main>
43    </AuthContext.Provider>
44  );
45}
46
47
48 export default App;
49
```

The screenshot shows a code editor with several tabs at the top: auth-context.js, App.js, Navigation.js (which is the active tab), and Home.js. Below the tabs, the file structure is shown: src > components > MainHeader > Navigation.js. The code itself is a functional component named Navigation. It imports React, useContext from react, and AuthContext from a context file. It also imports classes from a CSS module. The component uses the useContext hook to get the AuthContext, which contains a isLoggedIn boolean. It then renders a navigation bar with three items: 'Users' (for logged-in users), 'Admin' (for logged-in users), and a 'Logout' button (also for logged-in users). The code is numbered from 1 to 26.

```
auth-context.js      App.js      Navigation.js X      Home.js
src > components > MainHeader > Navigation.js > Navigation
1 import React, { useContext } from "react";
2 import AuthContext from "../../context/auth-context";
3
4 import classes from "./Navigation.module.css";
5
6 const Navigation = (props) => {
7   const context = useContext(AuthContext);
8   return (
9     <nav className={classes.nav}>
10       <ul>
11         {context.isLoggedIn && (
12           <li>
13             <a href="/">Users</a>
14           </li>
15         )}
16         {context.isLoggedIn && (
17           <li>
18             <a href="/">Admin</a>
19           </li>
20         )}
21         {context.isLoggedIn && (
22           <li>
23             <button onClick={context.onLogOut}>Logout</button>
24           </li>
25         )}
26       </ul>

```

Building & Using a Custom Context Provider Component:

```
src > context > auth-context.js > ...
1 import React, { useState, useEffect } from "react";
2 const AuthContext = React.createContext({
3   isLoggedIn: false,
4   onLogOut: null,
5   onLogin: (email, password) => {},
6 });
7
8 export const AuthContextProvider = (props) => {
9   const [isLoggedIn, setIsLoggedIn] = useState(false);
10  useEffect(() => {
11    const storedUserLoggedInInformation = localStorage.getItem("isLoggedIn");
12
13    if (storedUserLoggedInInformation === "1") {
14      setIsLoggedIn(true);
15    }
16  }, []);
17
18  const logOutHandler = () => {
19    localStorage.removeItem("isLoggedIn");
20    setIsLoggedIn(false);
21  };
22  const logInHandler = () => {
23    localStorage.setItem("isLoggedIn", "1");
24    setIsLoggedIn(true);
25  };
26
27  return (
28    <AuthContext.Provider
29      value={{
30        isLoggedIn: false,
31        onLogOut: logOutHandler,
32        onLogin: logInHandler,
33      }}
34    >
35      {props.children}
36    </AuthContext.Provider>
37  );
38};
39 export default AuthContext;
```

```
src > index.js > ...
1 import React from "react";
2 import ReactDOM from "react-dom/client";
3
4 import "./index.css";
5 import App from "./App";
6 import { AuthContextProvider } from "./context/auth-context";
7
8 const root = ReactDOM.createRoot(document.getElementById("root"));
9 root.render([
10   <AuthContextProvider>
11     <App />
12   </AuthContextProvider>
13 ]);
```

```

1 import React, { useContext } from "react";
2
3 import Login from "./components/Login/Login";
4 import Home from "./components/Home/Home";
5 import MainHeader from "./components/MainHeader/MainHeader";
6 import AuthContext from "./context/auth-context";
7
8 function App() {
9   const context = useContext(AuthContext);
10  return (
11    <React.Fragment>
12      <MainHeader />
13      <main>
14        {!context.isLoggedIn && <Login />}
15        {context.isLoggedIn && <Home />}
16      </main>
17    </React.Fragment>
18  );
19}
20
21 export default App;
22

```

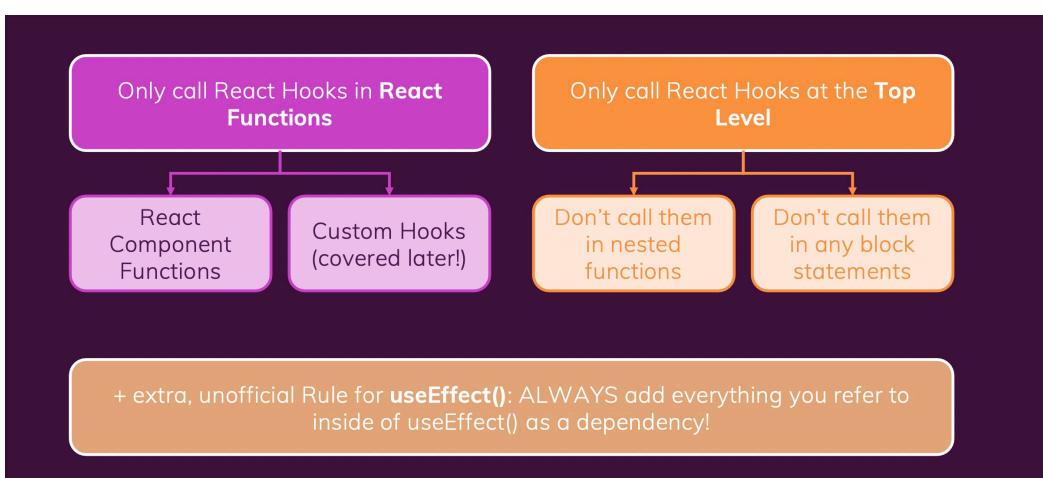
Context Disadvantages:

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:

```
src > components > Demo > DemoOutput.js > [?] default
1 import React from "react";
2
3 const DemoOutput = (props) => {
4   return <p>{props.show ? "This is new!" : ""}</p>;
5 }
6
7 export default React.memo(DemoOutput);
```

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:

```
src > [JS] App.js > [JS] App > [JS] messageHandler > [JS] useState() callback
1 import React, { useState, useCallback } from "react";
2 import "./App.css";
3 import Button from "./components/UI/Button/Button";
4 import DemoOutput from "./components/Demo/DemoOutput";
5 function App() {
6   const [message, setMessage] = useState(false);
7   console.log("RUNNING...");
8   const messageHandler = useCallback((event) => [
9     setMessage((prevState) => {
10       return !prevState;
11     });
12   ], []);
13   return (
14     <div className="app">
15       <h1>Hi there!</h1>
16       <DemoOutput show={false}></DemoOutput>
17       <Button onClick={messageHandler}>Click!</Button>
18     </div>
19   );
20 }
21
22 export default App;
23
```

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:

```
src > components > [JS] User.js > [JS] User
1 import classes from "./User.module.css";
2 import { Component } from "react";
3
4
5 class User extends Component {
6   render() {
7     return <li className={classes.user}>{this.props.name}</li>;
8   }
9 }
10
11 // const User = (props) => {
12 //   return <li className={classes.user}>{props.name}</li>;
13 // };
14
15 export default User;
16
```

Managing state in class based components:

```

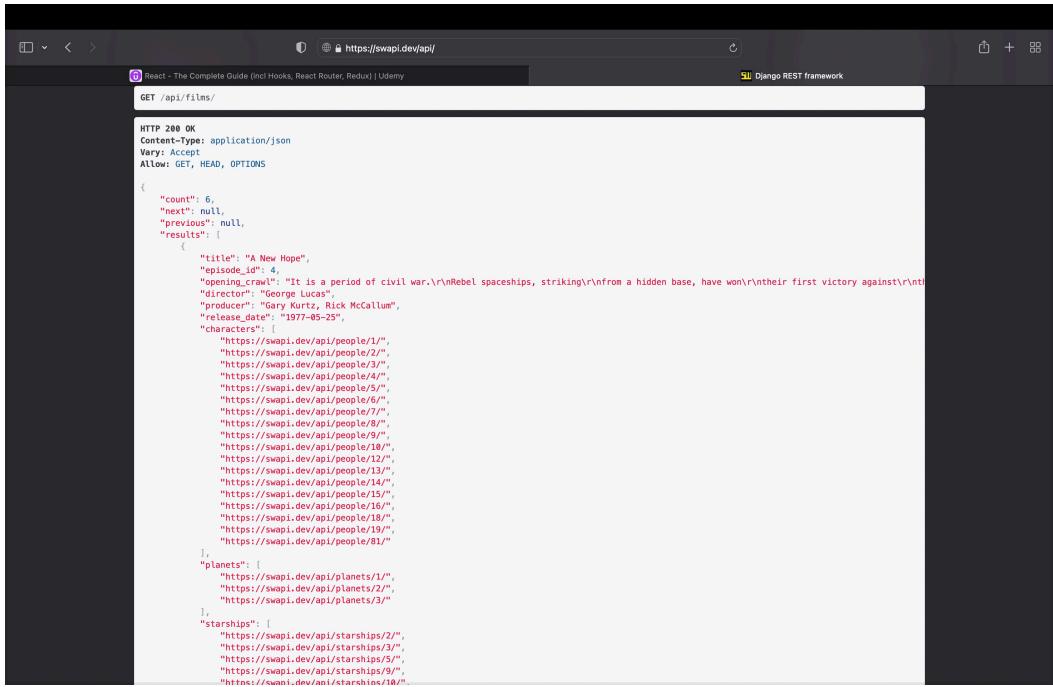
  User.js          Users.js  X
src > components > Users.js > Users
1  import { useState, Component } from "react";
2  import User from "./User";
3
4  import classes from "./Users.module.css";
5
6  const DUMMY_USERS = [
7    { id: "u1", name: "Remez" },
8    { id: "u2", name: "Gigo" },
9    { id: "u3", name: "Fredi" },
10   ];
11
12 class Users extends Component {
13   constructor() {
14     super();
15     this.state = {
16       showUsers: true,
17     }; // Had to be setuped like this.
18   }
19
20   toggleUsersHandler() {
21     this.setState((currState) => {
22       return { showUsers: !currState.showUsers };
23     });
24   }
25   render() {
26     const usersList = (
27       <ul>
28         {DUMMY_USERS.map((user) => (
29           <User key={user.id} name={user.name} />
30         ))}
31       </ul>
32     );
33     return (
34       <div className={classes.users}>
35         <button onClick={this.toggleUsersHandler.bind(this)}>
36           {this.state.showUsers ? "Hide" : "Show"} Users
37         </button>
38         {this.state.showUsers && usersList}
39       </div>
40     );
41   }
42 }
43 export default Users;

```

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



A screenshot of a web browser window. The address bar shows `https://swapi.dev/api/`. The title bar says "React - The Complete Guide [incl Hooks, React Router, Redux] | Udemy". The main content area displays a JSON response from a Django REST framework endpoint. The response is for a GET request to `/api/films/`. It includes headers like `HTTP 200 OK`, `Content-Type: application/json`, `Vary: Accept`, and `Allow: GET, HEAD, OPTIONS`. The JSON payload contains fields for count, next, previous, and results, which list various Star Wars films with their titles, episode IDs, opening crawl descriptions, directors, producers, release dates, and URLs for characters, planets, and starships.

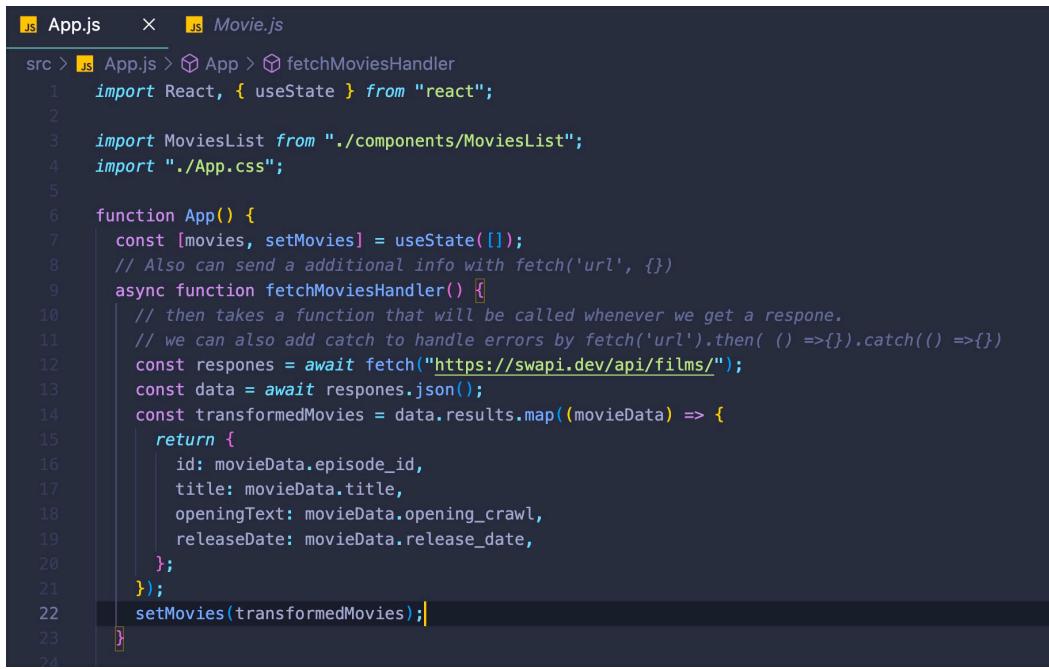
```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS

{
    "count": 6,
    "next": null,
    "previous": null,
    "results": [
        {
            "title": "A New Hope",
            "episode_id": 4,
            "opening_crawl": "It is a period of civil war.\r\nRebel spaceships, striking\r\nfrom a hidden base, have won\r\ntheir first victory against\r\nthe Galactic Empire.\r\nThe Empire strike back,\r\nreaching out to grab their own\n            "directors": "George Lucas",
            "producers": "Gary Kurtz, Rick McCallum",
            "release_date": "1977-05-25",
            "characters": [
                "https://swapi.dev/api/people/1/",
                "https://swapi.dev/api/people/2/",
                "https://swapi.dev/api/people/3/",
                "https://swapi.dev/api/people/4/",
                "https://swapi.dev/api/people/5/",
                "https://swapi.dev/api/people/6/",
                "https://swapi.dev/api/people/7/",
                "https://swapi.dev/api/people/8/",
                "https://swapi.dev/api/people/9/",
                "https://swapi.dev/api/people/10/",
                "https://swapi.dev/api/people/11/",
                "https://swapi.dev/api/people/12/",
                "https://swapi.dev/api/people/13/",
                "https://swapi.dev/api/people/14/",
                "https://swapi.dev/api/people/15/",
                "https://swapi.dev/api/people/16/",
                "https://swapi.dev/api/people/18/",
                "https://swapi.dev/api/people/19/",
                "https://swapi.dev/api/people/81/"
            ],
            "planets": [
                "https://swapi.dev/api/planets/1/",
                "https://swapi.dev/api/planets/2/",
                "https://swapi.dev/api/planets/3/"
            ],
            "starships": [
                "https://swapi.dev/api/starships/2/",
                "https://swapi.dev/api/starships/3/",
                "https://swapi.dev/api/starships/5/",
                "https://swapi.dev/api/starships/9/",
                "https://swapi.dev/api/starships/10/"
            ]
        }
    ]
}
```

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

```
src > App.js > App > [o] fetchMoviesHandler > [o] then() callback
1 import React, { useState } from "react";
2
3 import MoviesList from "./components/MoviesList";
4 import "./App.css";
5
6 function App() {
7   const [movies, setMovies] = useState([]);
8   // Also can send a additional info with fetch('url', {})
9   const fetchMoviesHandler = (event) => {
10     // then takes a function that will be called whenever we get a response.
11     // we can also add catch to handle errors by fetch('url').then( () =>{})
12     fetch("https://swapi.dev/api/films/")
13       .then((responses) => {
14         return responses.json();
15         // the then takes the value returned from the responses.json() (its
16       })
17       .then((data) => {
18         const transformedMovies = data.results.map((movieData) => {
19           return {
20             id: movieData.episode_id,
21             title: movieData.title,
22             openingText: movieData.opening_crawl,
23             releaseDate: movieData.release_date,
24           };
25         });
26         setMovies(transformedMovies);
27       });
28     };
29
30   return (
31     <React.Fragment>
32       <section>
33         <button onClick={fetchMoviesHandler}>Fetch Movies</button>
34       </section>
35       <section>
36         <MoviesList movies={movies} />
37       </section>
38     </React.Fragment>
39   );
40 }
```

Using `async` / `await`:



```
src > App.js > App > fetchMoviesHandler
1 import React, { useState } from "react";
2
3 import MoviesList from "./components/MoviesList";
4 import "./App.css";
5
6 function App() {
7   const [movies, setMovies] = useState([]);
8   // Also can send a additional info with fetch('url', {})
9   async function fetchMoviesHandler() {
10     // then takes a function that will be called whenever we get a response.
11     // we can also add catch to handle errors by fetch('url').then( () =>{}).catch(() =>{})
12     const responses = await fetch("https://swapi.dev/api/films/");
13     const data = await responses.json();
14     const transformedMovies = data.results.map((movieData) => {
15       return {
16         id: movieData.episode_id,
17         title: movieData.title,
18         openingText: movieData.opening_crawl,
19         releaseDate: movieData.release_date,
20       };
21     });
22     setMovies(transformedMovies);
23   }
24 }
```

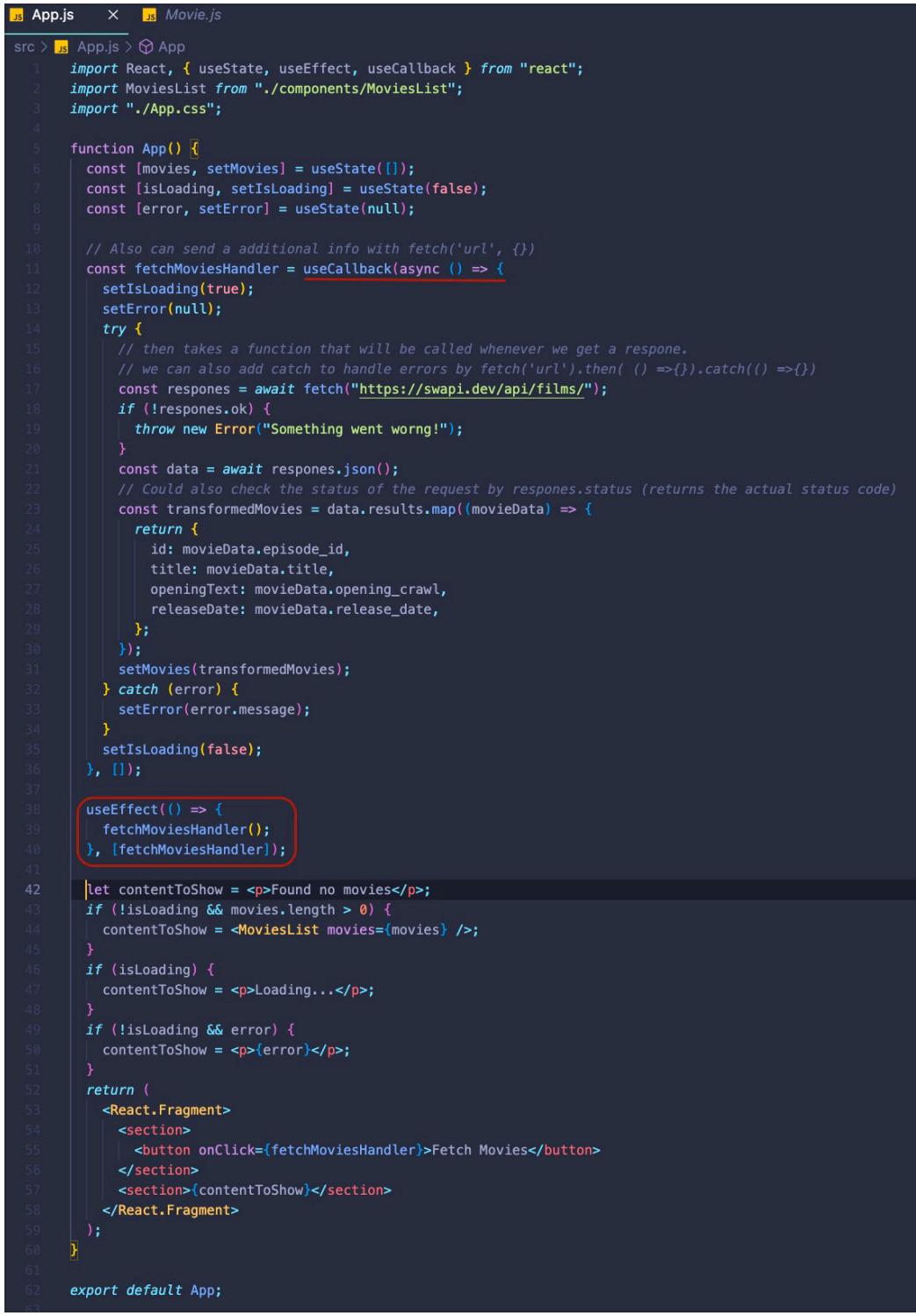
Handling Loading & Data States:

```
App.js
src > App.js > App
1 import React, { useState } from "react";
2
3 import MoviesList from "./components/MoviesList";
4 import "./App.css";
5
6 function App() {
7   const [movies, setMovies] = useState([]);
8   const [isLoading, setIsLoading] = useState(false);
9   // Also can send a additional info with fetch('url', {})
10  async function fetchMoviesHandler() {
11    setIsLoading(true);
12    // then takes a function that will be called whenever we get a response.
13    // we can also add catch to handle errors by fetch('url').then( () =>{}).catch(() =>{})
14    const responses = await fetch("https://swapi.dev/api/films/");
15    const data = await responses.json();
16    const transformedMovies = data.results.map((movieData) => {
17      return {
18        id: movieData.episode_id,
19        title: movieData.title,
20        openingText: movieData.opening_crawl,
21        releaseDate: movieData.release_date,
22      };
23    });
24    setMovies(transformedMovies);
25    setIsLoading(false);
26  }
27
28  return (
29    <React.Fragment>
30      <section>
31        <button onClick={fetchMoviesHandler}>Fetch Movies</button>
32      </section>
33      <section>
34        {!isLoading && movies.length > 0 && <MoviesList movies={movies} />}
35        {!isLoading && movies.length === 0 && <p>Found no movies</p>}
36        {isLoading && <p>Loading...</p>}
37      </section>
38    </React.Fragment>
39  );
40}
```

Handling Http Errors:

```
  JS App.js   X  JS Movie.js
src > JS App.js > ...
5  function App() {
6    const [movies, setMovies] = useState([]);
7    const [isLoading, setIsLoading] = useState(false);
8    const [error, setError] = useState(null);
9    // Also can send a additional info with fetch('url', {})
10   async function fetchMoviesHandler() {
11     setIsLoading(true);
12     setError(null);
13     try {
14       // then takes a function that will be called whenever we get a response.
15       // we can also add catch to handle errors by fetch('url').then( () =>{}).catch(() =>{})
16       const responses = await fetch("https://swapi.dev/api/films/");
17       if (!responses.ok) {
18         throw new Error("Something went wrong!");
19       }
20       const data = await responses.json();
21       // Could also check the status of the request by responses.status (returns the actual status code)
22       const transformedMovies = data.results.map((movieData) => {
23         return {
24           id: movieData.episode_id,
25           title: movieData.title,
26           openingText: movieData.opening_crawl,
27           releaseDate: movieData.release_date,
28         };
29       });
30       setMovies(transformedMovies);
31     } catch (error) {
32       setError(error.message);
33     }
34     setIsLoading(false);
35   }
36
37   return (
38     <React.Fragment>
39       <section>
40         <button onClick={fetchMoviesHandler}>Fetch Movies</button>
41       </section>
42       <section>
43         {!isLoading && movies.length > 0 && <MoviesList movies={movies} />}
44         {!isLoading && movies.length === 0 && <p>Found no movies</p>}
45         {isLoading && <p>Loading...</p>}
46         {!isLoading && error && <p>{error}</p>}
47       </section>
48     </React.Fragment>
49   );
50 }
51
52 export default App;
```

Using useEffect() For Requests:



```
App.js      X  Movie.js
src > App.js > App
1 import React, { useState, useEffect, useCallback } from "react";
2 import MoviesList from "./components/MoviesList";
3 import "./App.css";
4
5 function App() {
6   const [movies, setMovies] = useState([]);
7   const [isLoading, setIsLoading] = useState(false);
8   const [error, setError] = useState(null);
9
10  // Also can send a additional info with fetch('url', {})
11  const fetchMoviesHandler = useCallback(async () => {
12    setIsLoading(true);
13    setError(null);
14    try {
15      // then takes a function that will be called whenever we get a response.
16      // we can also add catch to handle errors by fetch('url').then( () =>{}).catch(() =>{})
17      const responses = await fetch("https://swapi.dev/api/films/");
18      if (!responses.ok) {
19        throw new Error("Something went wrong!");
20      }
21      const data = await responses.json();
22      // Could also check the status of the request by responses.status (returns the actual status code)
23      const transformedMovies = data.results.map((movieData) => {
24        return {
25          id: movieData.episode_id,
26          title: movieData.title,
27          openingText: movieData.opening_crawl,
28          releaseDate: movieData.release_date,
29        };
30      });
31      setMovies(transformedMovies);
32    } catch (error) {
33      setError(error.message);
34    }
35    setIsLoading(false);
36  }, []);
37
38  useEffect(() => {
39    fetchMoviesHandler();
40  }, [fetchMoviesHandler]);
41
42  let contentToShow = <p>Found no movies</p>;
43  if (!isLoading && movies.length > 0) {
44    contentToShow = <MoviesList movies={movies} />;
45  }
46  if (isLoading) {
47    contentToShow = <p>Loading...</p>;
48  }
49  if (!isLoading && error) {
50    contentToShow = <p>{error}</p>;
51  }
52  return (
53    <React.Fragment>
54      <section>
55        <button onClick={fetchMoviesHandler}>Fetch Movies</button>
56      </section>
57      <section>{contentToShow}</section>
58    </React.Fragment>
59  );
60}
61
62 export default App;
63
```

Sending a POST Request:

```

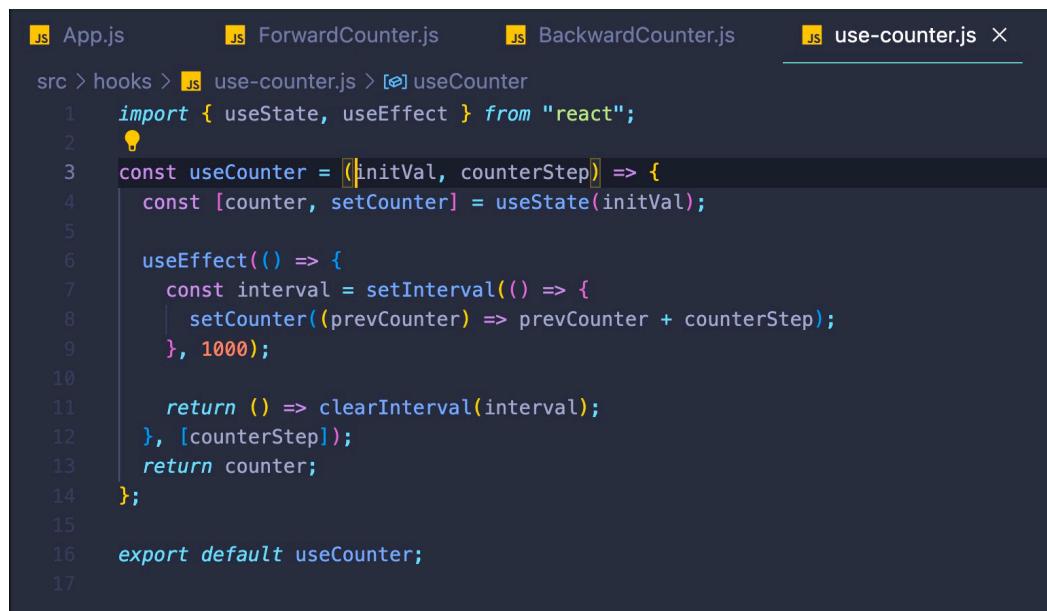
async function addMovieHandler(movie) {
  const response = await fetch(
    "https://react-udemy-course-http-71ec4-default-firebase.com.firebaseio.com/movies.json",
    {
      method: "POST",
      body: JSON.stringify(movie), // takes a JavaScript object/Array and transform it to JSON format
     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

```

Section 15: Building Custom React Hooks:

Creating a Custom React Hook Function:

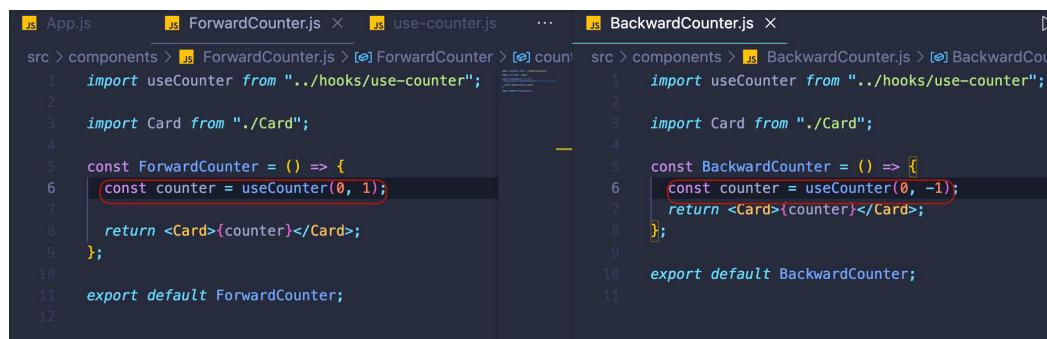
We have to start the name of the function with "use".



```

src > hooks > use-counter.js > [e] useCounter
1 import { useState, useEffect } from "react";
2
3 const useCounter = ([initVal, counterStep]) => {
4   const [counter, setCounter] = useState(initVal);
5
6   useEffect(() => {
7     const interval = setInterval(() => {
8       setCounter((prevCounter) => prevCounter + counterStep);
9     }, 1000);
10
11   return () => clearInterval(interval);
12 }, [counterStep]);
13   return counter;
14 };
15
16 export default useCounter;
17

```



```

src > components > ForwardCounter.js > [e] ForwardCounter > [e] count
1 import useCounter from "../hooks/use-counter";
2
3 import Card from "./Card";
4
5 const ForwardCounter = () => {
6   const counter = useCounter(0, 1);
7
8   return <Card>{counter}</Card>;
9 };
10
11 export default ForwardCounter;
12

src > components > BackwardCounter.js > [e] BackwardCounter
1 import useCounter from "../hooks/use-counter";
2
3 import Card from "./Card";
4
5 const BackwardCounter = () => {
6   const counter = useCounter(0, -1);
7
8   return <Card>{counter}</Card>;
9 };
10
11 export default BackwardCounter;
12

```

Building & Using the Custom Http Hook:

```
App.js          NewTask.js      use-http.js ×
src > hooks > use-http.js > [o] useHttp
1 import { useState } from "react";
2 const useHttp = (requestConfig, dataTransformer) => {
3   const [isLoading, setIsLoading] = useState(false);
4   const [error, setError] = useState(null);
5
6   const sendRequest = async (taskText) => {
7     setIsLoading(true);
8     setError(null);
9     try {
10       const response = await fetch(requestConfig.url, {
11         method: requestConfig.method ? requestConfig.method : "GET",
12         headers: requestConfig.headers ? requestConfig.headers : {},
13         body: requestConfig.body ? JSON.stringify(requestConfig.body) : null,
14       });
15
16       if (!response.ok) {
17         throw new Error("Request failed!");
18       }
19
20       const data = await response.json();
21
22       // const loadedTasks = [];
23
24       // for (const taskKey in data) {
25       //   loadedTasks.push({ id: taskKey, text: data[taskKey].text });
26       // }
27       dataTransformer(data);
28     } catch (err) {
29       setError(err.message || "Something went wrong!");
30     }
31     setIsLoading(false);
32   };
33   return {
34     isLoading,
35     error,
36     sendRequest,
37   };
38 };
39
40 export default useHttp;
```

```
App.js  NewTask.js  use-http.js
src > App.js > App
1 import React, { useEffect, useState } from "react";
2
3 import Tasks from "./components/Tasks/Tasks";
4 import NewTask from "./components/NewTask/NewTask";
5 import useHttp from "./hooks/use-http";
6
7 function App() {
8   const [tasks, setTasks] = useState([]);
9   const transformTask = (transformObject) => {
10     const loadedTasks = [];
11     for (const taskKey in transformObject) {
12       loadedTasks.push({ id: taskKey, text: transformObject[taskKey].text });
13     }
14     setTasks(loadedTasks);
15   };
16   const {
17     isLoading,
18     error,
19     sendRequest: fetchTasks,
20   } = useHttp(
21     {
22       url: "https://react-udemy-course-http-71ec4-default-firebaseio.com/tasks.json",
23     },
24     trnsformTask
25   );
26
27   useEffect(() => {
28     fetchTasks();
29   }, []);
30
31   const taskAddHandler = (task) => {
32     setTasks((prevTasks) => prevTasks.concat(task));
33   };
34
35   return (
36     <React.Fragment>
37       <NewTask onAddTask={taskAddHandler} />
38       <Tasks
39         items={tasks}
40         loading={isLoading}
41       </Tasks>
42     </React.Fragment>
43   );
44 }
```

Section 16: Working with Forms & User Input:

Handling the "was touched" State:

```
src > components > SimpleInput.js > SimpleInput.js > SimpleInput > formSubmitHandler
1 import { useState } from "react";
2 const SimpleInput = (props) => {
3   const [enteredName, setEnteredName] = useState("");
4   const [enteredNameTouch, setEnteredNameTouch] = useState(false);
5
6   const enteredNameIsValid = enteredName.trim() !== "";
7   const nameInputIsInvalid = !enteredNameIsValid && enteredNameTouch;
8
9   const nameInputChangeHandler = (event) => {
10     setEnteredName(event.target.value);
11   };
12
13   const nameInputBlurHandler = (event) => {
14     setEnteredNameTouch(true);
15   };
16
17   const formSubmitHandler = (event) => {
18     event.preventDefault();
19     setEnteredNameTouch(true);
20     if (!enteredNameIsValid) {
21       return;
22     }
23     setEnteredName("");
24     setEnteredNameTouch(false);
25   };
26   const nameEnteredValid = nameInputIsInvalid
27     ? "form-control invalid"
28     : "form-control";
29
30   return (
31     <form onSubmit={formSubmitHandler}>
32       <div className={nameEnteredValid}>
33         <label htmlFor="name">Your Name</label>
34         <input
35           type="text"
36           id="name"
37           onChange={nameInputChangeHandler}
38           onBlur={nameInputBlurHandler}
39         />
40         {nameInputIsInvalid && (
41           <p className="error-text">Can't submit an empty form</p>
42         )}
43       </div>
44       <div className="form-actions">
45         <button>Submit</button>
46       </div>
47     </form>
48   );
49 };
50
```

Adding A Custom Input Hook:

```
SimpleInput.js      use-input.js ×  index.css
src > hooks > use-input.js > [o] useInput
1 import { useState } from "react";
2 // This hook will manage the state of an input
3
4 const useInput = (validateValueFunc) => {
5   const [enteredValue, setEnteredValue] = useState("");
6   const [isTouched, setIsTouched] = useState(false);
7
8   const valueIsValid = validateValueFunc(enteredValue);
9   const hasError = !valueIsValid && isTouched;
10
11  const valueChangeHandler = (event) => {
12    setEnteredValue(event.target.value);
13  };
14  const inputBlurHandler = (event) => {
15    setIsTouched(true);
16  };
17
18  const rest = () => {
19    setEnteredValue("");
20    setIsTouched(false);
21  };
22
23  return [
24    value: enteredValue,
25    isValid: valueIsValid,
26    hasError,
27    valueChangeHandler,
28    inputBlurHandler,
29    rest,
30  ];
31};
32 export default useInput;
33
```

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").

```
JS redux-demo.js ×
JS redux-demo.js > ...
1  const redux = require("redux");
2
3  const counterReducer = (currentState = { counter: 0 }, dispatchAction) => {
4    return {
5      counter: currentState.counter + 1,
6    };
7  };
8
9  const store = redux.createStore(counterReducer);
10
11 const counterSubscriber = () => {
12   const latestStore = store.getState();
13   console.log(latestStore);
14 };
15
16 store.subscribe(counterSubscriber);
17 store.dispatch({ type: "increment" });
18
```

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

Another example using redux in react components:

```
JS my-redux.js ×    JS index.js
src > store > JS my-redux.js > [o] default
1  const redux = require("redux");
2
3  const counterReducer = (currentState = { counter: 0 }, action) => {
4    let value = currentState;
5    if (action.type === "increment") {
6      value = {
7        counter: currentState.counter + 1,
8      };
9    }
10   if (action.type === "decrement") {
11     value = {
12       counter: currentState.counter - 1,
13     };
14   }
15   return value;
16 };
17
18 const store = redux.createStore(counterReducer);
19
20 export default store;
```

```
JS my-redux.js      JS index.js ×
src > JS index.js > ...
1 import React from "react";
2 import ReactDOM from "react-dom/client";
3 import { Provider } from "react-redux";
4
5 import "./index.css";
6 import App from "./App";
7 import store from "./store/my-redux";
8
9 const root = ReactDOM.createRoot(document.getElementById("root"));
10 root.render(
11   <Provider store={store}>
12     <App />
13   </Provider>
14 );
15
```

Using Redux Data in React Components:

```
JS my-redux.js      JS Counter.js ×  JS index.js
src > components > JS Counter.js > [?] Counter
1 import classes from "./Counter.module.css";
2 import { useSelector } from "react-redux";
3 const Counter = () => {
4   const counterState = useSelector(state => state.counter);
5   const toggleCounterHandler = () => {};
6
7   return (
8     <main className={classes.counter}>
9       <h1>Redux Counter</h1>
10      <div className={classes.value}>{counterState}</div>
11      <button onClick={toggleCounterHandler}>Toggle Counter</button>
12    </main>
13  );
14};
15
16 export default Counter;
17
```

Dispatching Actions From Inside Components:

```
my-redux.js  Counter.js  index.js
src > components > Counter.js > [e] Counter
1 import classes from "./Counter.module.css";
2 import { useSelector, useDispatch } from "react-redux";
3 import { useState } from "react";
4 const Counter = () => {
5   const counterState = useSelector((state) => state.counter);
6   const [toggleCounterState, setToggleCounterState] = useState(false);
7   const dispatchAction = useDispatch();
8   const toggleCounterHandler = () => {
9     setToggleCounterState((prevState) => {
10       return !prevState;
11     });
12   };
13   const incrementHandler = () => {
14     dispatchAction({ type: "increment" });
15   };
16   const decrementHandler = () => {
17     dispatchAction({ type: "decrement" });
18   };
19
20   return (
21     <main className={classes.counter}>
22       <h1>Redux Counter</h1>
23       {toggleCounterState && [
24         <div className={classes.value}>{counterState}</div>
25       ]}
26       <div>
27         <button onClick={incrementHandler}>Increment</button>
28         <button onClick={decrementHandler}>Decrement</button>
29       </div>
30       <button onClick={toggleCounterHandler}>Toggle Counter</button>
31     </main>
32   );
33 };
34
35 export default Counter;
```

Redux toolkit:

```
my-redux.js × Counter.js index.js
src > store > my-redux.js > [?] store
1 import { createSlice, configureStore } from "@reduxjs/toolkit";
2
3 const initState = { counter: 0, showCounter: false };
4
5 const counterSlice = createSlice({
6   name: "counter",
7   initialState: initState,
8   reducers: {
9     increment(state) {
10       state.counter++;
11     },
12     decrement(state) {
13       state.counter--;
14     },
15     increase(state, action) {
16       state.counter += action.value;
17     },
18     toggle(state) {
19       state.showCounter = !state.showCounter;
20     },
21   },
22 });
23
24 const store = configureStore([
25   reducer: counterSlice.reducer,
26 ]);
27
28 export default store;
29
```

Migrating Everything To Redux Toolkit:

```
my-redux.js Counter.js index.js
src > components > Counter.js > [x] Counter > [x] increaseHandler
1 import classes from "./Counter.module.css";
2 import { useSelector, useDispatch } from "react-redux";
3 import { counterActions } from "../store/my-redux";
4 const Counter = () => {
5   const counterValue = useSelector((state) => state.counter);
6   const counterState = useSelector((state) => state.showCounter);
7
8   const dispatchAction = useDispatch();
9   const toggleCounterHandler = () => {
10     dispatchAction(counterActions.toggle());
11   };
12   const incrementHandler = () => {
13     dispatchAction(counterActions.increment());
14   };
15   const increaseHandler = () => [
16     dispatchAction(counterActions.increase(5)), // Creates - {type: "Unique", payload: 5}
17   ];
18   const decrementHandler = () => [
19     dispatchAction(counterActions.decrement());
20   ];
21
22   return (
23     <main className={classes.counter}>
24       <h1>Redux Counter</h1>
25       {counterState && <div className={classes.value}>{counterValue}</div>}
26       <div>
27         <button onClick={increaseHandler}>Increment by 5</button>
28         <button onClick={incrementHandler}>Increment</button>
29         <button onClick={decrementHandler}>Decrement</button>
30       </div>
31       <button onClick={toggleCounterHandler}>Toggle Counter</button>
32     </main>
33   );
34 };
35
36 export default Counter;
37
```

Working with Multiple Slices:

```
my-redux.js counter-slice.js auth-slice.js
src > store > counter-slice.js ...
1 import { createSlice } from "@reduxjs/toolkit";
2 const initialCounterState = { counter: 0, showCounter: false };
3
4 const counterSlice = createSlice({
5   name: "counter",
6   initialState: initialCounterState,
7   reducers: {
8     increment(state) {
9       state.counter++;
10    },
11     decrement(state) {
12       state.counter--;
13    },
14     increase(state, action) {
15       state.counter += action.payload;
16    },
17     toggle(state) {
18       state.showCounter = !state.showCounter;
19    },
20  },
21 });
22 export default counterSlice.reducer;
23 export const counterActions = counterSlice.actions;
```

```
JS my-redux.js      JS counter-slice.js      JS auth-slice.js X
src > store > JS auth-slice.js > ...
● 1 import { createSlice } from "@reduxjs/toolkit";
2
3   ↘ const initialAuthState = {
4     isAuthenticated: false,
5   };
6
7   ↘ const authSlice = createSlice({
8     name: "authentication",
9     initialState: initialAuthState,
10    ↘ reducers: {
11      ↘ login(state) {
12        state.isAuthenticated = true;
13      },
14      ↘ logout(state) {
15        state.isAuthenticated = false;
16      },
17    },
18  });
19  export default authSlice.reducer;
20  export const authActions = authSlice.actions;
21
```

```
JS my-redux.js X      JS counter-slice.js      JS auth-slice.js
src > store > JS my-redux.js > ↗ store > ↗ reducer > ↗ auth
1 import { configureStore } from "@reduxjs/toolkit";
2 import counterReducer from "./counter-slice";
3 import authReducer from "./auth-slice";
4 const store = configureStore({
5   reducer: [
6     counter: counterReducer,
7     auth: authReducer,
8   ],
9 });
10
11 export default store;
12
```

```

1 import Counter from "./components/Counter";
2 import { Fragment } from "react";
3 import Header from "./components/Header";
4 import Auth from "./components/Auth";
5 import { useSelector } from "react-redux";
6 import UserProfile from "./components/UserProfile";
7
8 function App() {
9   const userStatus = useSelector((state) => state.auth.isAuthenticated);
10
11   return (
12     <Fragment>
13       <Header />
14       {!userStatus && <Auth />}
15       {userStatus && <UserProfile />}
16       {userStatus && <Counter />}
17     </Fragment>
18   );
19 }
20
21 export default App;
22

```

```

1 import classes from "./Auth.module.css";
2 import { useRef } from "react";
3 import { useDispatch } from "react-redux";
4 import { authActions } from "../store/auth-slice";
5
6 const Auth = (props) => {
7   const dispatchAction = useDispatch();
8   const passwordInputRef = useRef();
9   const emailInputRef = useRef();
10  const formSubmitHandler = (event) => {
11    event.preventDefault();
12    dispatchAction(authActions.login());
13  };
14
15  return (
16    <main className={classes.auth}>
17      <section>
18        <form onSubmit={formSubmitHandler}>
19          <div className={classes.control}>
20            <label htmlFor="email">Email</label>
21            <input type="email" id="email" ref={emailInputRef} />
22          </div>
23          <div className={classes.control}>
24            <label htmlFor="password">Password</label>
25            <input type="password" id="password" ref={passwordInputRef} />
26          </div>
27        <button>Login</button>
28      </form>
29    </section>
30  </main>
31}
32
33 export default Auth;
34
35

```

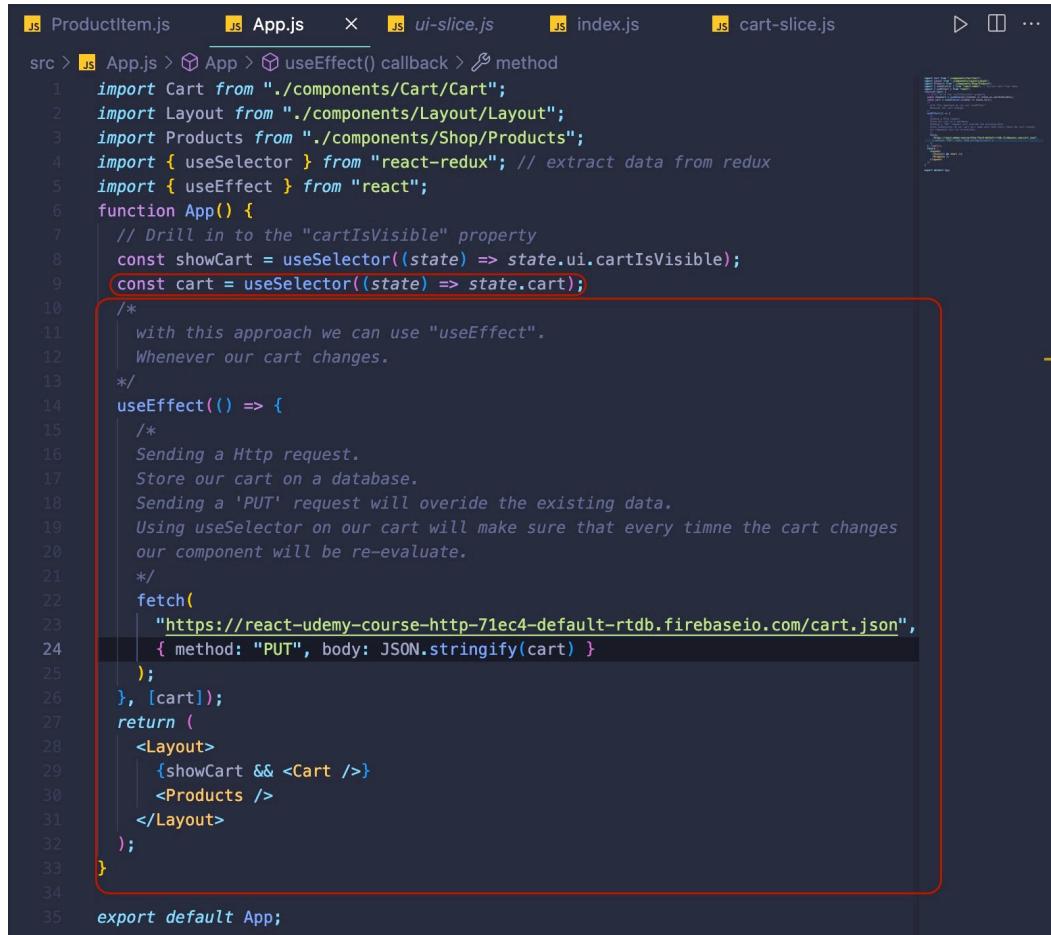
Clone from a branch - **git clone --single-branch --branch <branchname> <remote-repo>**

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:



```
src > JS App.js > ↗ App > ↗ useEffect() callback > ↗ method
1 import Cart from "./components/Cart/Cart";
2 import Layout from "./components/Layout/Layout";
3 import Products from "./components/Shop/Products";
4 import { useSelector } from "react-redux"; // extract data from redux
5 import { useEffect } from "react";
6 function App() {
7   // Drill in to the "cartIsVisible" property
8   const showCart = useSelector((state) => state.ui.cartIsVisible);
9   const cart = useSelector((state) => state.cart);
10  /*
11    with this approach we can use "useEffect".
12    Whenever our cart changes.
13  */
14  useEffect(() => {
15    /*
16      Sending a Http request.
17      Store our cart on a database.
18      Sending a 'PUT' request will override the existing data.
19      Using useSelector on our cart will make sure that every time the cart changes
20      our component will be re-evaluate.
21    */
22    fetch(
23      "https://react-udemy-course-http-71ec4-default-firebase.com/cart.json",
24      { method: "PUT", body: JSON.stringify(cart) }
25    );
26  }, [cart]);
27  return (
28    <Layout>
29      {showCart && <Cart />}
30      <Products />
31    </Layout>
32  );
33}
34
35 export default App;
```

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:

```

src > App.js > App > useEffect() callback > sendCartData > response
1 import Cart from "./components/Cart/Cart";
2 import Layout from "./components/Layout/Layout";
3 import Products from "./components/Shop/Products";
4 import {useSelector, useDispatch} from "react-redux"; // extract data from redux
5 import {useEffect, Fragment} from "react";
6 import {uiActions} from "./store/ui-slice";
7 import Notification from "./components/UI/Notification";
8
9 let firstLoad = true;
10
11 function App() {
12   // Drill in to the "cartIsVisible" property
13   const showCart = useSelector((state) => state.ui.cartIsVisible);
14   const cart = useSelector((state) => state.cart);
15   const dispatch = useDispatch();
16   const notification = useSelector((state) => state.ui.notification);
17
18   /*
19    with this approach we can use "useEffect".
20    Whenever our cart changes.
21   */
22   useEffect(() => {
23     /*
24      Sending a Http request.
25      Store our cart on a database.
26      Sending a 'PUT' request will override the existing data.
27      Using useSelector on our cart will make sure that every time the cart changes
28      our component will be re-evaluate.
29   */
30   const sendCartData = async () => {
31     dispatch(
32       uiActions.showNotification({
33         status: "pending",
34         title: "Sending...",
35         message: "Sending cart data",
36       })
37     );
38     const response = await fetch(
39       "https://react-udemy-course-http-71ec4-default-firebase.com/cart.json",
40       { method: "PUT", body: JSON.stringify(cart) }
41     );
42     if (!response.ok) {
43       throw new Error("Something went wrong!");
44     }
45     dispatch(
46       uiActions.showNotification({
47         status: "success",
48         title: "Success",
49         message: "Cart data sent!",
50       })
51     );
52   };
53   if (firstLoad) {
54     firstLoad = false;
55     return;
56   }
57   sendCartData().catch((error) => {
58     dispatch(
59       uiActions.showNotification({
60         status: "error",
61         title: "Error!",
62         message: "Sending cart data failed!",
63       })
64     );
65   });
66 }, [cart, dispatch]);
67
68 <Fragment>
69   {notification && (
70     <Notification
71       status={notification.status}
72       title={notification.title}
73       message={notification.message}
74     />
75   )}
76   <Layout>
77     {showCart && <Cart />}
78     <Products />
79   </Layout>
80

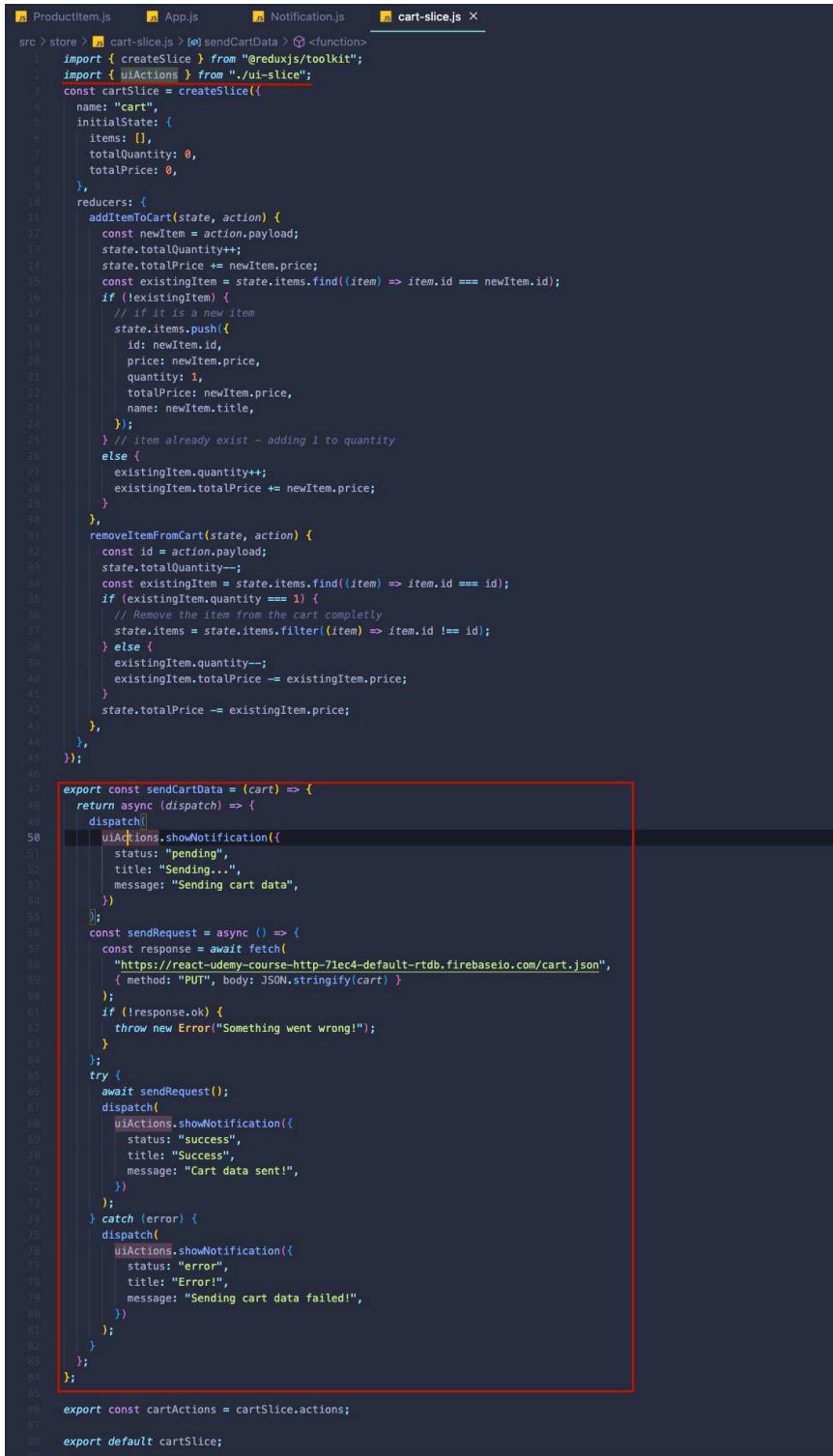
```

Another way of accomplishing this is by using "action creators":
App.js a lot slimmer

The screenshot shows a code editor with several tabs at the top: ProductItem.js, App.js (which is the active tab), Notification.js, and cart-slice.js. The code in App.js is as follows:

```
src > App.js > App > useEffect() callback
1 import Cart from "./components/Cart/Cart";
2 import Layout from "./components/Layout/Layout";
3 import Products from "./components/Shop/Products";
4 import { useSelector, useDispatch } from "react-redux"; // extract data from store
5 import { useEffect, Fragment } from "react";
6 import Notification from "./components/UI/Notification";
7 import sendCartData from "store/cart-slice";
8 let firstLoad: boolean
9
10 let firstLoad = true;
11
12 function App() {
13     // Drill in to the "cartIsVisible" property
14     const showCart = useSelector((state) => state.ui.cartIsVisible);
15     const cart = useSelector((state) => state.cart);
16     const dispatch = useDispatch();
17     const notification = useSelector((state) => state.ui.notification);
18     /*
19         with this approach we can use "useEffect".
20         Whenever our cart changes.
21     */
22     useEffect(() => {
23         if (firstLoad) {
24             firstLoad = false;
25             return;
26         }
27         dispatch(sendCartData(cart));
28     }, [cart, dispatch]);
29
30     return (
31         <Fragment>
32             {notification && (
33                 <Notification
34                     status={notification.status}
35                     title={notification.title}
36                     message={notification.message}
37                 />
38             )}
39             <Layout>
40                 {showCart && <Cart />}
41                 <Products />
42             </Layout>
43         </Fragment>
44     );
45 }
46
47 export default App;
48
```

A red box highlights the code block starting with `useEffect(() => {` and ending with `}, [cart, dispatch]);`. This highlights the logic that runs whenever the `cart` or `dispatch` dependencies change.



```

Productitem.js   App.js      Notification.js  cart-slice.js ×
src > store > cart-slice.js > sendCartData > <function>
1 import { createSlice } from "@reduxjs/toolkit";
2 import { uiActions } from "./ui-slice";
3 const cartSlice = createSlice({
4   name: "cart",
5   initialState: {
6     items: [],
7     totalQuantity: 0,
8     totalPrice: 0,
9   },
10  reducers: {
11    addItemToCart(state, action) {
12      const newItem = action.payload;
13      state.totalQuantity++;
14      state.totalPrice += newItem.price;
15      const existingItem = state.items.find((item) => item.id === newItem.id);
16      if (!existingItem) {
17        // If it is a new item
18        state.items.push({
19          id: newItem.id,
20          price: newItem.price,
21          quantity: 1,
22          totalPrice: newItem.price,
23          name: newItem.title,
24        });
25      } // item already exist - adding 1 to quantity
26      else {
27        existingItem.quantity++;
28        existingItem.totalPrice += newItem.price;
29      }
30    },
31    removeItemFromCart(state, action) {
32      const id = action.payload;
33      state.totalQuantity--;
34      const existingItem = state.items.find((item) => item.id === id);
35      if (existingItem.quantity === 1) {
36        // Remove the item from the cart completely
37        state.items = state.items.filter((item) => item.id !== id);
38      } else {
39        existingItem.quantity--;
40        existingItem.totalPrice -= existingItem.price;
41      }
42      state.totalPrice -= existingItem.price;
43    },
44  },
45 });
46
47 export const sendCartData = (cart) => {
48   return async (dispatch) => {
49     dispatch(uiActions.showNotification({
50       status: "pending",
51       title: "Sending...",
52       message: "Sending cart data",
53     }));
54   };
55   const sendRequest = async () => {
56     const response = await fetch(
57       "https://react-udemy-course-http-71ec4.firebaseio.com/cart.json",
58       { method: "PUT", body: JSON.stringify(cart) }
59     );
60     if (!response.ok) {
61       throw new Error("Something went wrong!");
62     }
63   };
64   try {
65     await sendRequest();
66     dispatch(uiActions.showNotification({
67       status: "success",
68       title: "Success",
69       message: "Cart data sent!",
70     }));
71   } catch (error) {
72     dispatch(uiActions.showNotification({
73       status: "error",
74       title: "Error!",
75       message: "Sending cart data failed!",
76     }));
77   }
78 };
79
80
81
82
83
84
85
86 export const cartActions = cartSlice.actions;
87
88 export default cartSlice;

```

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 screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor pane on the right.

EXPLORER

- OPEN EDITORS
 - App.js
 - cart-actions.js
- REDUX AGAIN
 - node_modules
 - public
 - src
 - components
 - Cart
 - Layout
 - Shop
 - ProductItem.js
 - ProductItem....
 - Products.js
 - Products.m...
 - UI
 - Card.js
 - Card.module...
 - Notification.js
 - Notification....
 - store
 - cart-actions.js
 - cart-slice.js
 - index.js
 - ui-slice.js
- App.js
- index.css
- index.js
- package-lock.json
- package.json

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.

```
src > App.js > App
1 import Cart from "./components/Cart/Cart";
2 import Layout from "./components/Layout/Layout";
3 import Products from "./components/Shop/Products";
4 import { useSelector, useDispatch } from "react-redux"; // extract data from state
5 import { useEffect, Fragment } from "react";
6 import Notification from "./components/UI/Notification";
7 import { sendCartData, fetchCartData } from "./store/cart-actions";
8
9 let firstLoad = true;
10
11 function App() {
12     // Drill in to the "cartIsVisible" property
13     const showCart = useSelector((state) => state.ui.cartIsVisible);
14     const cart = useSelector((state) => state.cart);
15     const dispatch = useDispatch();
16     const notification = useSelector((state) => state.ui.notification);
17     /*
18         with this approach we can use "useEffect".
19         Whenever our cart changes.
20     */
21     useEffect(() => {
22         dispatch(fetchCartData());
23     }, [dispatch]);
24     useEffect(() => {
25         if (firstLoad) {
26             firstLoad = false;
27             return;
28         }
29         dispatch(sendCartData(cart));
30     }, [cart, dispatch]);
31     return (
32         <Fragment>
33             {notification && [
34                 <Notification
35                     status={notification.status}
36                     title={notification.title}
37                     message={notification.message}
38                 />
39             ]}
40             <Layout>
41                 {showCart && <Cart />}
42                 <Products />
43             </Layout>
44         </Fragment>
45     );
46 }
47
48 export default App;
49
```

With this fix its complete:

The screenshot shows a code editor with three tabs: 'App.js', 'cart-slice.js X', and 'cart-actions.js'. The 'cart-slice.js' tab is active and displays the following code:

```
src > store > cart-slice.js > cartSlice > reducers > removeItemFromCart
1 import { createSlice } from "@reduxjs/toolkit";
2
3 const cartSlice = createSlice({
4   name: "cart",
5   initialState: {
6     items: [],
7     totalQuantity: 0,
8     totalPrice: 0,
9     changed: false,
10 },
11 reducers: {
12   replaceCart(state, action) {
13     state.items = action.payload.items;
14     state.totalQuantity = action.payload.totalQuantity;
15     state.totalPrice = action.payload.totalPrice;
16   },
17   addItemToCart(state, action) {
18     const newItem = action.payload;
19     state.totalQuantity++;
20     state.totalPrice += newItem.price;
21     state.changed = true;
22     const existingItem = state.items.find(item => item.id === newItem.i
23     if (!existingItem) {
24       // if it is a new item
25       state.items.push({
26         id: newItem.id,
27         price: newItem.price,
28         quantity: 1,
29         totalPrice: newItem.price,
30         name: newItem.title,
31       });
32     } // item already exist - adding 1 to quantity
33     else {
34       existingItem.quantity++;
35       existingItem.totalPrice += newItem.price;
36     }
37   },
38   removeItemFromCart(state, action) {
39     const id = action.payload;
40     state.totalQuantity--;
41     state.changed = true;
42     const existingItem = state.items.find(item => item.id === id);
43     if (existingItem.quantity === 1) {
44       // Remove the item from the cart completely
45       state.items = state.items.filter(item => item.id !== id);
46     } else {
47       existingItem.quantity--;
48       existingItem.totalPrice -= existingItem.price;
49     }
50     state.totalPrice -= existingItem.price;
51   },
52 },
53 
```

```
src > App.js > App > useEffect() callback
  1 import Cart from "./components/Cart/Cart";
  2 import Layout from "./components/Layout/Layout";
  3 import Products from "./components/Shop/Products";
  4 import { useSelector, useDispatch } from "react-redux"; // extract data from state
  5 import { useEffect, Fragment } from "react";
  6 import Notification from "./components/UI/Notification";
  7 import { sendCartData, fetchCartData } from "./store/cart-actions";
  8
  9 let firstLoad = true;
10
11 function App() {
12   // Drill in to the "cartIsVisible" property
13   const showCart = useSelector((state) => state.ui.cartIsVisible);
14   const cart = useSelector((state) => state.cart);
15   const dispatch = useDispatch();
16   const notification = useSelector((state) => state.ui.notification);
17   /*
18    | with this approach we can use "useEffect".
19    | Whenever our cart changes.
20   */
21   useEffect(() => {
22     dispatch(fetchCartData());
23   }, [dispatch]);
24   useEffect(() => {
25     if (firstLoad) {
26       firstLoad = false;
27       return;
28     }
29     if (cart.changed) [
30       dispatch(sendCartData(cart));
31     ]
32   }, [cart, dispatch]);
33   return (
34     <Fragment>
35       {notification && (
36         <Notification
37           status={notification.status}
38           title={notification.title}
39           message={notification.message}
40         />
41       )}
42       <Layout>
43         {showCart && <Cart />}
44         <Products />
45       </Layout>
46     </Fragment>
47   );
48 }
49
```

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):

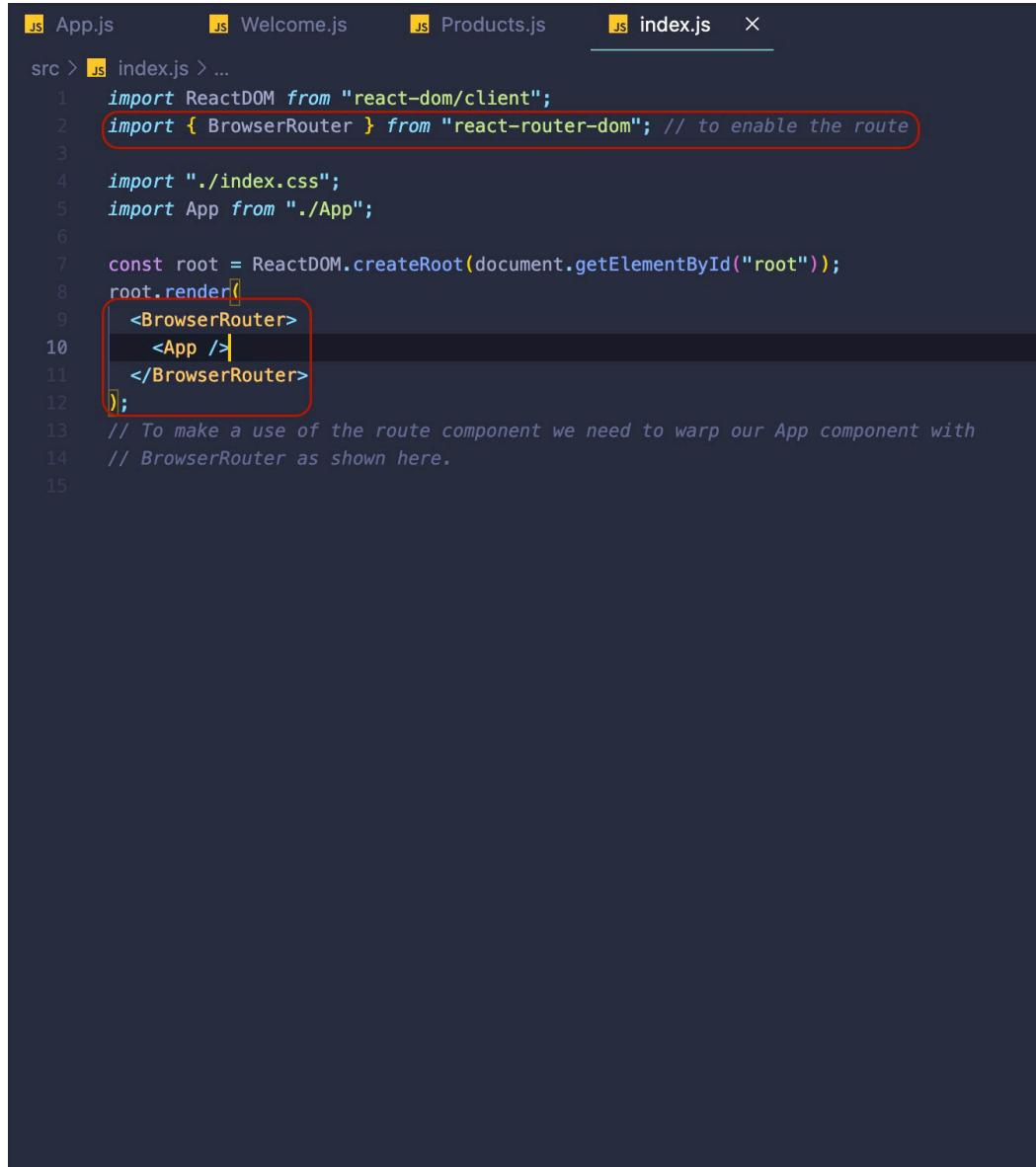
In terminal: npm install react-router-dom@5

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



```
src > index.js > ...
1 import ReactDOM from "react-dom/client";
2 import { BrowserRouter } from "react-router-dom"; // to enable the route
3
4 import "./index.css";
5 import App from "./App";
6
7 const root = ReactDOM.createRoot(document.getElementById("root"));
8 root.render([
9   <BrowserRouter>
10   <App />
11   </BrowserRouter>
12 ]);
13 // To make a use of the route component we need to warp our App component with
14 // BrowserRouter as shown here.
15
```

```

App.js          Welcome.js      Products.js    index.js
src > App.js > App
1 import { Route } from "react-router-dom"; // A component that let us define a certain path and the component that should be rendered
2 import Products from "./Components/Products";
3 import Welcome from "./Components/Welcome";
4 function App() {
5   return (
6     <div>
7       <Route path="/welcome">
8         <Welcome />
9       </Route>
10      <Route path="/products">
11        <Products />
12      </Route>
13    </div>
14  );
15}
16
17 export default App;
18 /*
19 <Route path="/welcome"/> - with that we saying - this route should be active when this:
20 our-domain.com/welcome will be active
21
22 */
23

```

Working with Links:

```

App.js          MainHeader.js > MainHeader
src > Components > MainHeader.js > MainHeader
1 import { Link } from "react-router-dom"; // Unlike the regular <a> tag this component prevents the browser default of loading another html page.
2 const MainHeader = () => {
3   return (
4     <header>
5       <nav>
6         <ul>
7           <li>
8             <Link to="/welcome">welcome</Link>
9           </li>
10          <li>
11            <Link to="/products">products </Link>
12          </li>
13        </ul>
14      </nav>
15    </header>
16  );
17}
18
19 export default MainHeader;

```

Using NavLink:

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 > MainHeader
src > Components > MainHeader.js > MainHeader
1 import { NavLink } from "react-router-dom"; // Unlike the regular <a> tag this component prevents the browser default of loading another html page.
2 import classes from "./MainHeader.module.css";
3 const MainHeader = () => {
4   return (
5     <header className={classes.header}>
6       <nav>
7         <ul>
8           <li>
9             <NavLink activeClassName={classes.active} to="/welcome">
10               Welcome
11             </NavLink>
12           </li>
13           <li>
14             <NavLink activeClassName={classes.active} to="/products">
15               Products(" ")
16             </NavLink>
17           </li>
18         </ul>
19       </nav>
20     </header>
21   );
22}
23
24 export default MainHeader;

```

Adding Dynamic Routes with Params:

```

src > App.js < MainHeader.js < Welcome.js < Products.js < ProductDetail.js < index.js
1 import { Route } from "react-router-dom"; // A component that let us define a certine path and the component that should be render
2 import Products from "./Pages/Products";
3 import Welcome from "./Pages/Welcome";
4 import MainHeader from "./Components/MainHeader";
5 import ProductDetail from "./Pages/ProductDetail";
6 function App() {
7   return (
8     <div>
9       <MainHeader />
10      <main>
11        <Route path="/welcome">
12          <Welcome />
13        </Route>
14        <Route path="/products">
15          <Products />
16        </Route>
17        <Route path="/product-detail/:productId">
18          /* This is a dynamic path segment -> tells react router that the overall path that of this page could be loaded
19          Will be something like this: our-domain.com/product-detail/anything (can be p1 p2 or whatever)
20        </>;
21        <ProductDetail />
22      </Route>
23    </main>
24  </div>
25);
26
27 export default App;
28 /*
29 <Route path="/welcome"/> - with that we saying - this route should be active when this:
30 our-domain.com/welcome will be active
31
32 */
33
34

```

Extracting Route Params:

```

src > Pages > ProductDetail.js > ProductDetail
1 import { useParams } from "react-router";
2 const ProductDetail = () => {
3   const params = useParams();
4   console.log([params.productId]);
5   return (
6     <section>
7       <h1>Product Detail</h1>
8       <p>{params.productId}</p>
9     </section>
10   );
11 };
12 export default ProductDetail;
13

```

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.

A screenshot of a code editor showing the file `App.js`. The code defines a `Switch` component with two nested `Route` components. The first nested route has a path of `/welcome` and points to a component named `Welcome`. The second nested route has a path of `/products` and points to a component named `Products`. A red box highlights the `<Switch>` tag.

```
1 import { Route, Switch } from "react-router-dom"; // A component that let us define a certine path and the component that should be render
2 import Products from "./Pages/Products";
3 import Welcome from "./Pages/Welcome";
4 import MainHeader from "./Components/MainHeader";
5 import ProductDetail from "./Pages/ProductDetail";
6 function App() {
7   return (
8     <div>
9       <MainHeader />
10      <main>
11        <Switch>
12          <Route path="/welcome">
13            <Welcome />
14          </Route>
15          <Route path="/products" exact>
16            <Products />
17          </Route>
18          <Route path="/products/:productId">
19            /* This is a dynamic path segment -> tells react router that the overall path that of this page sould be loaded
20             Will be something like this: our-domain.com/product-detail/anything (can be p1 p2 or whatever)
21           */
22            <ProductDetail />
23          </Route>
24        </Switch>
25      </main>
26    </div>
27  );
28}
29
30 export default App;
31 /*
32 <Route path="/welcome"/> - with that we saying - this route should be active when this:
33 our-domain.com/welcome will be active
34
35 */
36
```

Working with Nested Routes:

A screenshot of a code editor showing the file `Welcome.js`. It contains a `Route` component with a nested `Route` component. The outer route has a path of `/welcome` and points to a component named `NewUser`. The inner route has a path of `/new-user` and points to a component named `NewUser`. A red box highlights the inner `<Route>` tag.

```
1 import { Route } from "react-router";
2 const Welcome = () => {
3   return [
4     <section>
5       <h1>Welcome to the main page!</h1>
6       <Route path="/welcome/new-user">
7         <p>Welcome my new FRIEND!</p>
8       </Route>
9     </section>
10   ];
11 }
12
13 export default Welcome;
14
```

Redirecting The User:

. The code also includes comments explaining the purpose of dynamic path segments and the overall path loading logic."/>

```

src > App.js > App
1 import { Route, Switch, Redirect } from "react-router-dom"; // A component that let us define a certain path and the component that should be render
2 import Products from "./Pages/Products";
3 import Welcome from "./Pages/Welcome";
4 import MainHeader from "./Components/MainHeader";
5 import ProductDetail from "./Pages/ProductDetail";
6
7 function App() {
8   return (
9     <div>
10       <MainHeader />
11       <main>
12         <Switch>
13           <Route path="/" exact>
14             <Redirect to="/welcome"/></Redirect>
15           </Route>
16           <Route path="/welcome">
17             <Welcome />
18           </Route>
19           <Route path="/products" exact>
20             <Products />
21           </Route>
22           <Route path="/products/:productId">
23             /* This is a dynamic path segment -> tells react router that the overall path of this page should be loaded
24             Will be something like this: our-domain.com/product-detail/anything (can be p1 p2 or whatever)
25           */
26             <ProductDetail />
27           </Route>
28         </Switch>
29       </main>
30     </div>
31   );
32 }
33
34 export default App;
35 /*
36 <Route path="/welcome"/> - with that we saying - this route should be active when this:
37 our-domain.com/welcome will be active
38 */
39

```

Implementing Programmatic (Imperative) Navigation:

```

src > pages > NewQuote.js > ...
1 import QuoteForm from "../components/quotes/QuoteForm";
2 import { useHistory } from "react-router-dom";
3
4 const NewQuote = () => {
5   const history = useHistory();
6   const addQuoteHandler = (quote) => {
7     console.log(quote);
8     history.push("/quotes");
9   };
10  return <QuoteForm onAddQuote={addQuoteHandler} />;
11};
12
13 export default NewQuote;

```

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

```
src > components > quotes > QuoteForm.js > [o] QuoteForm
  1 import { Fragment, useRef, useState } from "react";
  2 import Card from "../UI/Card";
  3 import LoadingSpinner from "../UI>LoadingSpinner";
  4 import classes from "./QuoteForm.module.css";
  5 import { Prompt } from "react-router-dom";
  6
  7 const QuoteForm = (props) => {
  8   const authorInputRef = useRef();
  9   const textInputRef = useRef();
 10   const [isEntered, setIsEntered] = useState(false);
 11
 12   function submitFormHandler(event) {
 13     event.preventDefault();
 14
 15     const enteredAuthor = authorInputRef.current.value;
 16     const enteredText = textInputRef.current.value;
 17
 18     // optional: Could validate here
 19
 20     props.onAddQuote({ author: enteredAuthor, text: enteredText });
 21   }
 22
 23   const formFocusHandler = () => {
 24     setIsEntered(true);
 25   };
 26   const finishEnteringHandler = () => {
 27     setIsEntered(false);
 28   };
 29
 30   return (
 31     <Fragment>
 32       <Prompt
 33         when={isEntered}
 34         message={(location) => {
 35           return "Are you sure boy?";
 36         }}
 37       />
 38       <Card>
 39         <form
 40           onFocus={formFocusHandler}
 41           className={classes.form}
 42           onSubmit={submitFormHandler}>
```

Working with Query Parameters:

```
src > components > quotes > js QuoteList.js > [x] QuoteList
1 import { Fragment } from "react";
2 import { useHistory, useLocation } from "react-router-dom";
3 import QuoteItem from "./QuoteItem";
4 import classes from "./QuoteList.module.css";
5
6 const sortQuotes = (quotes, ascending) => {
7   return quotes.sort((quoteA, quoteB) => {
8     if (ascending) {
9       return quoteA.id > quoteB.id ? 1 : -1;
10    } else {
11      return quoteA.id < quoteB.id ? 1 : -1;
12    }
13  });
14};
15
16 const QuoteList = (props) => {
17   // return a 'history' object that allows us to change and manage the url
18   const history = useHistory();
19   // return a 'location' object that have information about the current loaded url
20   const location = useLocation();
21   const queryParams = new URLSearchParams(location.search);
22   // if the sort param is equal to asc or not
23   const isSortingAscending = queryParams.get("sort") === "asc";
24
25   const sortedQuotes = sortQuotes(props.quotes, isSortingAscending);
26
27   const changeSortingHandler = () => {
28     history.push("/quotes?sort=" + (isSortingAscending ? "desc" : "asc"));
29   };
30
31   return (
32     <Fragment>
33       <div className={classes.sorting}>
34         <button onClick={changeSortingHandler}>
35           Sort {isSortingAscending ? "Descending" : "Ascending"}
36         </button>
37       </div>
38       <ul className={classes.list}>
39         {sortedQuotes.map((quote) => (
40           <QuoteItem
41             key={quote.id}
42             id={quote.id}
43             author={quote.author}
44             text={quote.text}
45           />
46         )));
47       </ul>
48     </Fragment>
49   );
50
51   export default QuoteList;
52
```

Getting Creative With Nested Routes:

Here is an example on how 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.

```
src > pages > [js] QuoteDetail.js > [x] QuoteDetail
1 import { Route, useParams } from "react-router";
2 import { Link } from "react-router-dom";
3 import { Fragment } from "react";
4 import Comments from "../components/comments/Comments";
5 import HighlightedQuote from "../components/quotes/HighlightedQuote"
6 const DUMMY_QUOTES = [
7   { id: "q1", author: "Remez", text: "Remez is a big gamer" },
8   { id: "q2", author: "Roei", text: "Roei is a shooter" },
9   { id: "q3", author: "Dor", text: "Dor is crazy" },
10 ];
11 const QuoteDetail = () => {
12   const params = useParams();
13   const quote = DUMMY_QUOTES.find((quote) => quote.id === params.quoteId);
14   if (!quote) {
15     return <p>No quote found!</p>;
16   }
17   return (
18     <Fragment>
19       <h1>Quote Detail</h1>
20       <HighlightedQuote text={quote.text} author={quote.author} />
21       <Route path={`/quotes/${params.quoteId}/comments`} exact>
22         <div className="centered">
23           <Link
24             to={`/quotes/${params.quoteId}/comments`}
25             className="btn--flat"
26           >
27             Load Comments
28           </Link>
29         </div>
30       </Route>
31       <Route path={`/quotes/${params.quoteId}/comments`}>
32         <Comments />
33       </Route>
34     </Fragment>
35   );
36 };
37 export default QuoteDetail;
38
```

Writing More Flexible Routing Code:

```
.js App.js .js QuoteForm.js .js QuoteList.js .js QuoteDetail.js ×
src > pages > .js QuoteDetail.js > [⊗] QuoteDetail
1   import { Route, useParams, Link, useRouteMatch } from "react-router-dom";
2   import { Fragment } from "react";
3   import Comments from "../components/comments/Comments";
4   import HighlightedQuote from "../components/quotes/HighlightedQuote";
5   const DUMMY_QUOTES = [
6     { id: "q1", author: "Remez", text: "Remez is a big gamer" },
7     { id: "q2", author: "Roei", text: "Roei is a shooter" },
8     { id: "q3", author: "Dor", text: "Dor is crazy" },
9   ];
10  const QuoteDetail = () => [
11    const match = useRouteMatch();
12    const params = useParams();
13    const quote = DUMMY_QUOTES.find(quote => quote.id === params.quoteId);
14    if (!quote) {
15      return <p>No quote found!</p>;
16    }
17    return (
18      <Fragment>
19        <h1>Quote Detail</h1>
20        <HighlightedQuote text={quote.text} author={quote.author} />
21        <Route path={match.path} exact>
22          <div className="centered">
23            <Link to={match.url + "/comments"} className="btn--flat">
24              Load Comments
25            </Link>
26            </div>
27          </Route>
28          <Route path={match.path + "/comments"}>
29            <Comments />
30          </Route>
31        </Fragment>
32      );
33    };
34    export default QuoteDetail;
35
```

Sending & Getting Quote Data via Http:

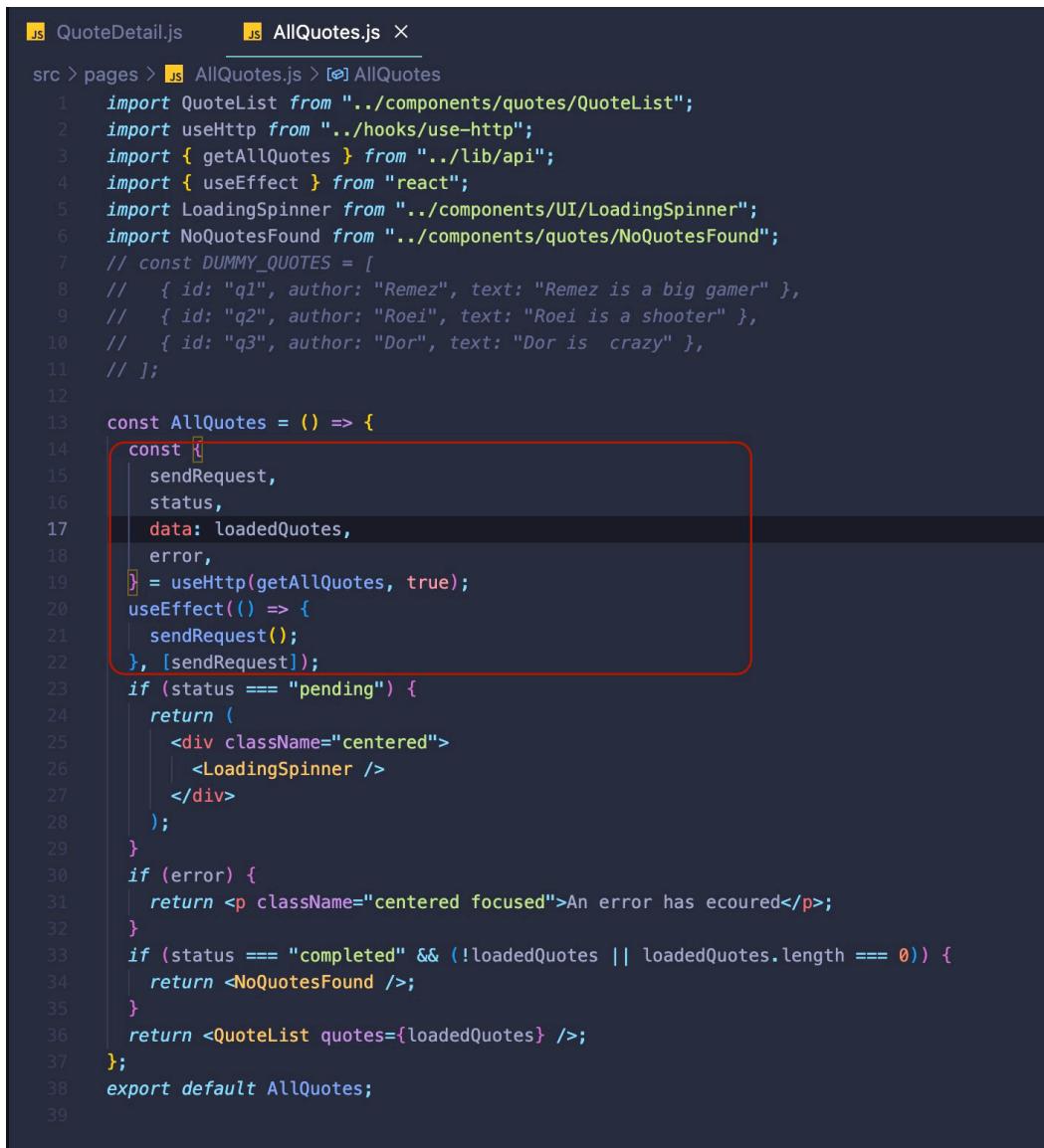
Using a custom http hook and a custom methods to fetch a single quote from firebase.

```
src > pages > [js] QuoteDetail.js > [o] QuoteDetail
1 import { Route, useParams, Link, useRouteMatch } from "react-router-dom";
2 import { Fragment, useEffect } from "react";
3 import Comments from "../components/comments/Comments";
4 import HighlightedQuote from "../components/quotes/HighlightedQuote";
5 import useHttp from "../hooks/use-http";
6 import { getSingleQuote } from "../lib/api";
7 import LoadingSpinner from "../components/UI>LoadingSpinner";
```

```
9 const QuoteDetail = () => {
10   const match = useRouteMatch();
11   const params = useParams();
12   const {
13     sendRequest,
14     status,
15     data: loadedQuote,
16     error,
17   } = useHttp(getSingleQuote, true);
18   const { quoteId } = params;
19   useEffect(() => {
20     sendRequest(quoteId);
21   }, [sendRequest, quoteId]);
22   if (status === "pending") {
23     return (
24       <div className="centered">
25         <LoadingSpinner />
26       </div>
27     );
28   }
29   if (error) {
30     return <p className="centered">(error)</p>;
31   }
32   if (!loadedQuote.text) {
33     return <p className="centered">No Quotes Found</p>;
34   }
35   return (
36     <Fragment>
37       <h1>Quote Detail</h1>
38       <HighlightedQuote text={loadedQuote.text} author={loadedQuote.author} />
39       <Route path={match.path} exact>
40         <div className="centered">
41           <Link to={match.url + "/comments"} className="btn--flat">
42             Load Comments
43           </Link>
44         </div>
45       </Route>
46       <Route path={(match.path + "/comments")}>
47         <Comments />
48       </Route>
49     </Fragment>
50   );
51 }
```

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)



```

src > pages > AllQuotes.js > [AllQuotes]
1 import QuoteList from "../components/quotes/QuoteList";
2 import useHttp from "../hooks/use-http";
3 import { getAllQuotes } from "../lib/api";
4 import { useEffect } from "react";
5 import LoadingSpinner from "../components/UI>LoadingSpinner";
6 import NoQuotesFound from "../components/quotes/NoQuotesFound";
7 // const DUMMY_QUOTES = [
8 //   { id: "q1", author: "Remez", text: "Remez is a big gamer" },
9 //   { id: "q2", author: "Roei", text: "Roei is a shooter" },
10 //   { id: "q3", author: "Dor", text: "Dor is crazy" },
11 // ];
12
13 const AllQuotes = () => {
14   const [ {
15     sendRequest,
16     status,
17     data: loadedQuotes,
18     error,
19   } ] = useHttp(getAllQuotes, true);
20   useEffect(() => {
21     sendRequest();
22   }, [sendRequest]);
23   if (status === "pending") {
24     return (
25       <div className="centered">
26         <LoadingSpinner />
27       </div>
28     );
29   }
30   if (error) {
31     return <p className="centered focused">An error has occurred</p>;
32   }
33   if (status === "completed" && (!loadedQuotes || loadedQuotes.length === 0)) {
34     return <NoQuotesFound />;
35   }
36   return <QuoteList quotes={loadedQuotes} />;
37 };
38 export default AllQuotes;
39

```

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:

```

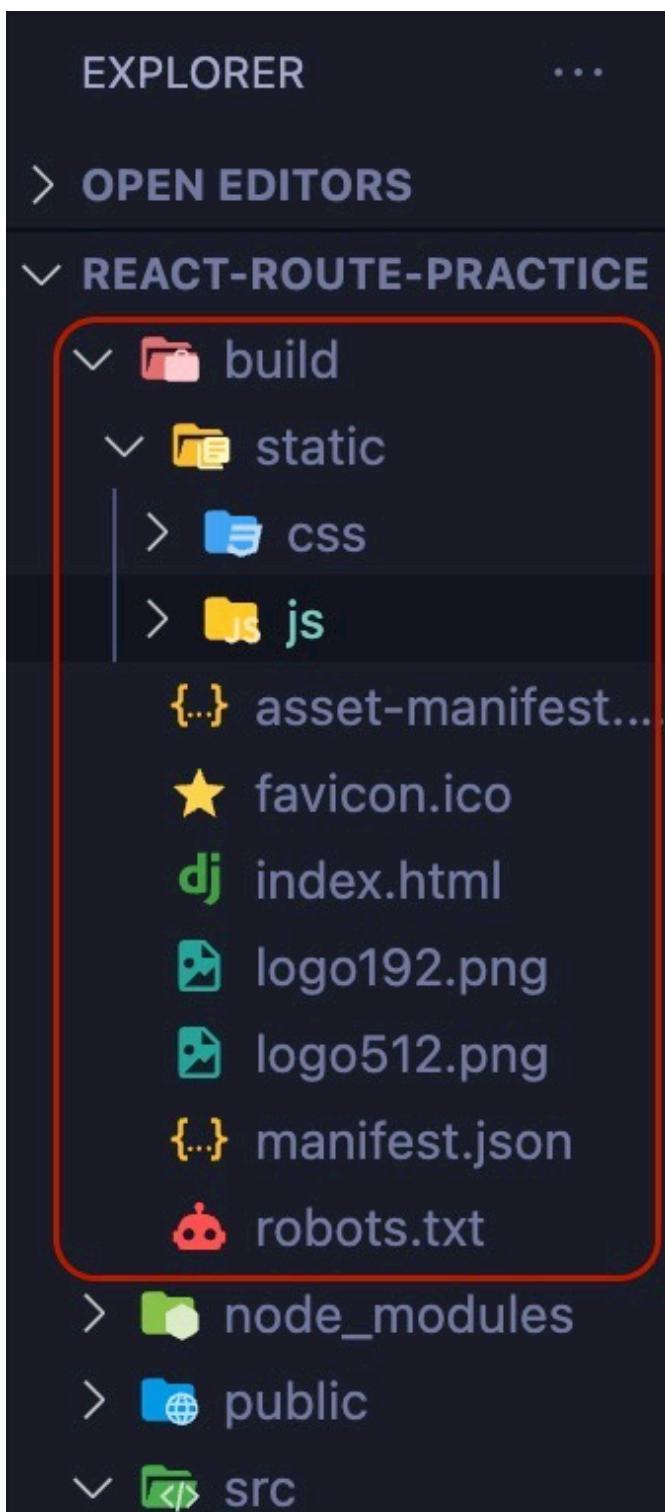
1 import React, { Suspense } from "react";
2 import { Route, Switch, Redirect } from "react-router-dom";
3
4 import AllQuotes from "./pages/AllQuotes";
5 import Layout from "./components/layout/Layout";
6 import LoadingSpinner from "./components/UI>LoadingSpinner";
7
8 /*
9 A method that helps us splitting our code.
10 The React.lazy will be called only when the page is needed.
11 To make this work we need to define a 'fall-back ui'
12 the code that we are downloading with:
13 React.lazy(() => {
14   import("./pages/NewQuote");
15 });
16 can take some time therefore we need to use special react component
17 called "Suspense" and warp all the react code that use React.lazy.
18 */
19 const NewQuote = React.lazy(() => import("./pages/NewQuote"));
20 const QuoteDetail = React.lazy(() => import("./pages/QuoteDetail"));
21 const NotFound = React.lazy(() => import("./pages/NotFound"));
22
23 function App() {
24   return (
25     <Layout>
26       <Suspense
27         fallback={
28           <div className="centered">
29             <LoadingSpinner />
30           </div>
31         }
32       >
33         <Switch>
34           <Route path="/" exact>
35             <Redirect to="/quotes"/></Redirect>
36           </Route>
37           <Route path="/quotes" exact>
38             <AllQuotes />
39           </Route>
40           <Route path="/quotes/:quoteId">
41             <QuoteDetail />
42           </Route>
43           <Route path="/new-quote">
44             <NewQuote />
45           </Route>
46           <Route path="*>
47             <NotFound />
48           </Route>
49         </Switch>
50       </Suspense>
51     </Layout>
52   );
53 }
54 //<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:



```
src > components > Auth > AuthForm.js > [o] AuthForm > [o] submitHandler
1 import { useState, useRef } from "react";
2
3 import classes from "./AuthForm.module.css";
4
5 const API_KEY = "AIzaSyDVxBZnx2-yQwfW4crmWoyjgnxrS4gEGn4";
6
7 const AuthForm = () => {
8   const [isLogin, setIsLogin] = useState(true);
9   const emailInputRef = useRef();
10  const passwordInputRef = useRef();
11  const switchAuthModeHandler = () => {
12    setIsLogin((prevState) => !prevState);
13  };
14  const submitHandler = (event) => {
15    event.preventDefault();
16
17    const userEmail = emailInputRef.current.value;
18    const userPassword = passwordInputRef.current.value;
19    // Can check for valid input
20
21    if (isLogin) {
22    } else {
23      // Sending a sign-up request
24      fetch(
25        `https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=${API_KEY}`,
26        {
27          method: "POST",
28          body: JSON.stringify({
29            email: userEmail,
30            password: userPassword,
31            returnSecureToken: true,
32          }),
33          headers: { "Content-Type": "application/json" },
34        }
35      ).then((response) => {
36        if (response.ok) {
37          // Success case
38        } else {
39          response.json().then((data) => {
40            // show error modal
41            console.log(data);
42          });
43        }
44      });
45    }
46  };
47
48  return (
49    <section className={classes.auth}>
50      <h1>{isLogin ? "Login" : "Sign Up"}</h1>
51      <form onSubmit={submitHandler}>
52        <div className={classes.control}>
53          <label htmlFor="email">Your Email</label>
54          <input type="email" id="email" required ref={emailInputRef} />
55        </div>
56        <div className={classes.control}>
57          <button type="submit" value="Submit">Submit</button>
58        </div>
59      </form>
60    </section>
61  );
62}
63
```

Adding User Login:

```
AuthForm.js X
src > components > Auth > AuthForm.js > [AuthForm]
18
19 const submitHandler = (event) => {
20   event.preventDefault();
21
22   const userEmail = emailInputRef.current.value;
23   const userPassword = passwordInputRef.current.value;
24
25   let url = isLoggedIn
26     ? `https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=${API_KEY}`
27     : `https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=${API_KEY}`;
28
29   // Can check for valid input
30   setIsLoading(true);
31   fetch(url, {
32     method: "POST",
33     body: JSON.stringify({
34       email: userEmail,
35       password: userPassword,
36       returnSecureToken: true,
37     }),
38     headers: { "Content-Type": "application/json" },
39   })
40     .then((response) => {
41       setIsLoading(false);
42       if (response.ok) {
43         // Success case
44         return response.json();
45       } else {
46         return response.json().then((data) => {
47           // show error modal
48           // console.log(data);
49           let errorMessage = isLoggedIn
50             ? "Failed to Login!"
51             : "Authentication Failed!";
52           if (data && data.error && data.error.message) {
53             errorMessage = data.error.message;
54           }
55           throw new Error(errorMessage);
56         });
57       }
58     })
59     .then((data) => {
60       console.log(data);
61     })
62     .catch((error) => {
63       alert(error.message);
64     });
65   };
66   > return (...);
67   );
68 }
69
70 export default AuthForm;
```

Managing The Auth State With Context:

```
JS AuthForm.js      JS MainNavigation.js      JS auth-context.js X
src > store > JS auth-context.js > [o] AuthContextProvider > [o] loginHandler
1   import React, { useState } from "react";
2
3   const AuthContext = React.createContext({
4     token: "",
5     isLogin: false,
6     login: (token) => {},
7     logout: () => {},
8   });
9
10  export const AuthContextProvider = (props) => {
11    const [token, setToken] = useState(null);
12
13    const userIsLoggedIn = !!token;
14
15    const loginHandler = (token) => [
16      setToken(token);
17    ];
18    const logeOutHandler = () => {
19      setToken(null);
20    };
21    const contextValue = {
22      token,
23      isLoggedIn: userIsLoggedIn,
24      login: loginHandler,
25      logout: logeOutHandler,
26    };
27    return (
28      <AuthContext.Provider value={contextValue}>
29        {props.children}
30      </AuthContext.Provider>
31    );
32  };
33
34  export default AuthContext;
35
```

A screenshot of a code editor showing the file `index.js`. The code is a React application structure. It imports `ReactDOM`, `BrowserRouter`, and `AuthContextProvider`. It then creates a root element and renders a component tree. The `AuthContextProvider` components are highlighted with red boxes.

```
src > index.js > ...
1 import ReactDOM from "react-dom/client";
2 import { BrowserRouter } from "react-router-dom";
3 import { AuthContextProvider } from "./store/auth-context";
4
5 import "./index.css";
6 import App from "./App";
7
8 const root = ReactDOM.createRoot(document.getElementById("root"));
9 root.render(
10   <AuthContextProvider>
11     <BrowserRouter>
12       <App />
13     </BrowserRouter>
14   </AuthContextProvider>
15 );
16
```

```
JS AuthForm.js      JS MainNavigation.js X  JS index.js      JS auth-context.js
src > components > Layout > JS MainNavigation.js > [o] MainNavigation
1 import { Link } from "react-router-dom";
2 import { useContext } from "react";
3 import AuthContext from "../../store/auth-context";
4 import classes from "./MainNavigation.module.css";
5
6 const MainNavigation = () => {
7   const authContext = useContext(AuthContext);
8   const isLoggedIn = authContext.isLoggedIn;
9
10  const logoutHandler = () => {
11    authContext.logout();
12  };
13
14  return (
15    <header className={classes.header}>
16      <Link to="/">
17        <div className={classes.logo}>React Auth</div>
18      </Link>
19      <nav>
20        <ul>
21          {!isLoggedIn && (
22            <li>
23              <Link to="/auth">Login</Link>
24            </li>
25          )}
26          {isLoggedIn && (
27            <li>
28              <Link to="/profile">Profile</Link>
29            </li>
30          )}
31          {isLoggedIn && (
32            <li>
33              <button onClick={logoutHandler}>Logout</button>
34            </li>
35          )}
36        </ul>
37      </nav>
38    </header>
39  );
40};
41
42 export default MainNavigation;
43
```

Protecting Frontend Pages:

```
js App.js X
src > js App.js > ⚡ App
1 import { Switch, Route, Redirect } from "react-router-dom";
2 import React, { Suspense } from "react";
3 import { useContext } from "react";
4 import AuthContext from "./store/auth-context";
5 import Layout from "./components/Layout/Layout";
6 // import UserProfile from "./components/Profile/UserProfile";
7 import LoadingSpinner from "./components/UI>LoadingSpinner";
8 // import AuthPage from "./pages/AuthPage";
9 import HomePage from "./pages/HomePage";
10
11 const AuthPage = React.lazy(() => import("./pages/AuthPage"));
12 const ProfilePage = React.lazy(() => import("./pages/ProfilePage"));
13 function App() {
14   const authContext = useContext(AuthContext);
15   return (
16     <Layout>
17       <Suspense fallback={<LoadingSpinner />}>
18         <Switch>
19           <Route path="/" exact>
20             <HomePage />
21           </Route>
22           { !authContext.isLoggedIn && (
23             <Route path="/auth">
24               <AuthPage />
25             </Route>
26           )}
27           {authContext.isLoggedIn && (
28             <Route path="/profile">
29               <ProfilePage />
30             </Route>
31           )}
32           <Route path="*>
33             <Redirect to="/" />
34           </Route>
35         </Switch>
36       </Suspense>
37     </Layout>
38   );
39 }
40
41 export default App;
42
```

Persisting The User Authentication Status:

```
App.js          auth-context.js X  AuthForm.js
src > store > auth-context.js > [o] AuthContextProvider > [o] loginHandler
1 import React, { useState } from "react";
2
3 const AuthContext = React.createContext({
4   token: "",
5   isLogin: false,
6   login: (token) => {},
7   logout: () => {},
8 });
9
10 export const AuthContextProvider = (props) => {
11   const initToken = localStorage.getItem("token");
12   const [token, setToken] = useState(initToken);
13
14   const userIsLoggedIn = !!token;
15
16   const loginHandler = [token] => {
17     // Saving the token in the browser local storage.
18     // that way once a user logged in and refresh the page he will be
19     // still logged in
20     localStorage.setItem("token", token);
21     setToken(token);
22   };
23
24   const logeOutHandler = () => {
25     // localStorage.clear() - clears all the local storage
26     localStorage.removeItem("token");
27     setToken(null);
28   };
29
30   const contextValue = {
31     token: token,
32     isLoggedIn: userIsLoggedIn,
33     login: loginHandler,
34     logout: logeOutHandler,
35   };
36
37   return (
38     <AuthContext.Provider value={contextValue}>
39       {props.children}
40     </AuthContext.Provider>
41   );
42
43   export default AuthContext;
44
```

Adding Auto-Logout:

```
JS App.js          JS auth-context.js      JS AuthForm.js ×
src > components > Auth > JS AuthForm.js > [AuthForm]
1 import { useState, useRef, useContext } from "react";
2 import LoadingSpinner from "../UI>LoadingSpinner";
3 import classes from "./AuthForm.module.css";
4 import AuthContext from "../../store/auth-context";
5 import { useHistory } from "react-router-dom";
6
7 const API_KEY = "AIzaSyDVxBZnx2-yQwfW4crmWoyjgnxrS4gEGn4";
8 /*
9 206383e7-4874-4d60-b968-e0ff8f7bd14a
10 remez@d.com
11 123456789
12 */
13 const AuthForm = () => {
14   const history = useHistory();
15   const [isLogin, setIsLogin] = useState(true);
16   const [isLoading, setIsLoading] = useState(false);
17   const emailInputRef = useRef();
18   const passwordInputRef = useRef();
19   const authContext = useContext(AuthContext);
20   const switchAuthModeHandler = () => {
21     setIsLogin((prevState) => !prevState);
22   };
23
24   const submitHandler = (event) => {
25     event.preventDefault();
26
27     const userEmail = emailInputRef.current.value;
28     const userPassword = passwordInputRef.current.value;
29
30     let url = isLogin
31       ? `https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword`
32       : `https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=${API_K
33
34     // Can check for valid input
35     setIsLoading(true);
36     > fetch(url, { ...
37       })
38         .then((response) => { ...
39       })
40         .then((data) => {
41           const tokenExpirationTime = new Date(
42             new Date().getTime() + +data.expiresIn * 1000
43           );
44           authContext.login(data.idToken, tokenExpirationTime.toISOString());
45           history.replace("/"); // Redirecting the user to the home page
46         })
47         .catch((error) => {
48           alert(error.message);
49         });
50       );
51     > return ( ...
52       );
53     );
54   };
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
```

```

  App.js          auth-context.js X  AuthForm.js
src > store > auth-context.js > [o] AuthContextProvider > [o] loginHandler
  1 import React, { useState } from "react";
  2
  3 const AuthContext = React.createContext({
  4   token: "",
  5   isLoggedIn: false,
  6   login: (token) => {},
  7   logout: () => {},
  8 });
  9
 10 const calcRemmainingTime = (expirationTime) => {
 11   const currentTime = new Date().getTime();
 12   const adjExperationTime = new Date(expirationTime).getTime();
 13   const remainTime = adjExperationTime - currentTime;
 14
 15   return remainTime;
 16 };
 17
 18 export const initToken: string | null = () => {
 19   const initToken = localStorage.getItem("token");
 20   const [token, setToken] = useState(initToken);
 21
 22   const userIsLoggedIn = !!token;
 23
 24   const logeOutHandler = () => {
 25     // localStorage.clear() - clears all the local storage
 26     localStorage.removeItem("token");
 27     setToken(null);
 28   };
 29   const loginHandler = (token, expirationTime) => {
 30     // Saving the token in the browser local storage.
 31     // that way once a user logged in and refresh the page he will be
 32     // still logged in
 33     localStorage.setItem("token", token);
 34     setToken(token);
 35
 36     const remainTime = calcRemmainingTime(expirationTime);
 37     setTimeout([logeOutHandler, remainTime]);
 38   };
 39   const contextValue = {
 40     token,
 41     isLoggedIn: userIsLoggedIn,
 42     login: loginHandler,
 43     logout: logeOutHandler,
 44   };
 45 }

```

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 Nextrjs 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 > ...

```
1 import classes from "./MeetupDetail.module.css";
2
3 const MeetupDetail = (props) => {
4   return (
5     <section className={classes.detail}>
6       <img src={props.img} alt={props.title} />
7       <h1>{props.title}</h1>
8       <address>{props.address}</address>
9       <p>{props.description}</p>
10      </section>
11    );
12  };
13 export default MeetupDetail;
14
```

MeetupDetail.module.css × MeetupDetail.js

components > meetups > MeetupDetail.module.css > ↗

```
1 .detail {
2   text-align: center;
3 }
4 .detail img {
5   width: 100%;
6 }
```

Data Fetching for Static Pages:

```

js index.js ×
pages > js index.js > ⚡ getStaticProps
1 import MeetupList from "../components/meetups/MeetupList";
2
3 const DUMMY_MEETUPS = [
4   {
5     id: "m1",
6     title: "Haifa City",
7     image:
8       "https://upload.wikimedia.org/wikipedia/commons/1/1d/Western_Haifa_from_",
9     address: "Haifa",
10    description: "The First meetup",
11  },
12  {
13    id: "m2",
14    title: "Haifa City",
15    image:
16      "https://upload.wikimedia.org/wikipedia/commons/1/1d/Western_Haifa_from_",
17    address: "Haifa-Center",
18    description: "The Second meetup",
19  },
20];
21
22 const HomePage = (props) => {
23   return <MeetupList meetups={props.meetups} />;
24 };
25 /*
26
only in component files that are in the pages folder.
Nextjs will look first for this kind of function
and will call it before the component function.
The code inside the function will not be executed on the client side and,
will never end up in the client side.
The function needs to always 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.
 kinda like in java or c (console arguments) main(argv, args)
*/
27 export async function getStaticProps() {
28   // fetch data from database/API/ or whatever
29   return {
30     props: {
31       meetups: DUMMY_MEETUPS,
32     },
33   };
34 }
35 export default HomePage;
36
37
38
39
40
41
42
43
44
45
46
47

```

More on Static Site Generation (SSG):

```

21
22 const HomePage = (props) => {
23   return <MeetupList meetups={props.meetups} />;
24 };
25 /*
26
27 only in component files that are in the pages folder.
28 Nextjs will look first for this kind of function
29 and will call it before the component function.
30 The code inside the function will not be executed on the client side and,
31 will never end up in the client side.
32 The function needs to always return an object.
33 The object must have a "props" "field" and this field must contain an object.
34 this object with the field "props" than will be sent as our props
35 in our component.
36 kinda like in java or c (console arguments) main(argv, args).
37 The revalidate field handy in cases where we want to revalidate (get the data again).
38 sometimes the data we are working with can be out-dated quickly therefore
39 if we use revalidate (and as a value we need to give it a number)
40 we tell nextjs that we want to call getStaticProps every number of seconds
41 (the number we set as an argument. here we set it to 10 - every 10 seconds)
42 (only if requests come's to this path)
43 */
44 export async function getStaticProps() {
45   // fetch data from database/API/ or whatever
46   return {
47     props: {
48       meetups: DUMMY_MEETUPS,
49     },
50     revalidate: 10,
51   };
52 }
53 export default HomePage;
54

```

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

```

54 /*
55 this function will always run on the server after deployment.
56 use it in cases where we want to revalidate for every request.
57 */
58
59 export async function getServerSideProps(context) {
60   // with the context we get access to the request:
61   const req = context.req;
62   // with the context we get access to the response:
63   const res = context.res;
64   // fetch data or anything we want to do.
65   return {
66     props: {
67       meetups: DUMMY_MEETUPS,
68     },
69   };
70 }
71
72 export default HomePage;
73

```

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

```
js index.js pages      js index.js .../[meetupId] X
pages > [meetupId] > js index.js > ⚡ getStaticPaths > ⚡ paths > ⚡ params > ⚡ meetupId
1  import { Fragment } from "react";
2  import MeetupDetail from "../../components/meetups/MeetupDetail";
3  const MeetupDetails = () => {
4    return (
5      <MeetupDetail
6        img="https://upload.wikimedia.org/wikipedia/commons/1/1d/Western_Haifa.jpg"
7        alt="Haifa City"
8        address="Haifa"
9        description="The First meetup"
10       />
11    );
12  };
13
14 // Needed because its a dynamic page component and we using
15 // getStaticProps
16 export function getStaticPaths() {
17   return {
18     fallback: false,
19     paths: [
20       {
21         params: {
22           meetupId: "m1",
23         },
24       },
25       {
26         params: {
27           meetupId: "m2",
28         },
29       },
30     ],
31   };
32 }
33
34 export async function getStaticProps(context) {
35   const meetupId = context.params.meetupId;
```

Introducing API Routes:

The screenshot shows the VS Code interface with the Explorer and Editor panes. The Explorer pane displays the project structure:

- OPEN EDITORS
- REACT-NEXTJS-PROJECT
 - .next
 - components
 - layout
 - meetups
 - ui
 - node_modules
 - pages
 - [meetupId]
 - index.js
 - api
 - new-meetup.js
 - new-meetup
 - index.js
 - _app.js
 - index.js
 - public
 - styles
 - globals.css
- package-lock.json
- package.json

new-meetup.js ×

```
pages > api > new-meetup.js > handler
1  /**
2   * the url of the file:
3   * /api/new-meetup
4   */
5
6  function handler(req, res) {
7    if (req.method === "POST") {
8      const data = req.body;
9      const { title, image, address, description } = data
10     }
11   }
12
13 export default handler;
```

Working with MongoDB:

The screenshot shows the VS Code interface with the Explorer and Editor panes. The Explorer pane displays the project structure, identical to the previous screenshot.

The Editor pane shows the content of new-meetup.js with a red box highlighting the MongoDB import and connection code:

```
pages > api > new-meetup.js > handler > message
1  /**
2   * the url of the file:
3   * /api/new-meetup
4   */
5  import { MongoClient } from "mongodb";
6
7  async function handler(req, res) {
8    if (req.method === "POST") {
9      const data = req.body;
10
11      const client = await MongoClient.connect(
12        "mongodb+srv://Remez:vbMk3le70okvPy0@react-udemy.bb vem7r.mongodb.net"
13      );
14      const DataBase = client.db();
15      const meetupsCollection = DataBase.collection("meetups");
16      const result = await meetupsCollection.insertOne(data);
17      client.close();
18      res.status(201).json({ message: "New Meetup Inserted" });
19    }
20  }
21
22
23 export default handler;
```

Sending Http Requests To Our API Routes:

```
js new-meetup.js      js index.js  X  Mongo Db
pages > new-meetup > js index.js > [e] default
1 import NewMeetupForm from "../../components/meetups/NewMeetupForm";
2 import { useRouter } from "next/router";
3
4 const NewMeetupPage = () => {
5   const router = useRouter();
6   const addMeetupHandler = async (newMeetupData) => {
7     const response = await fetch("/api/new-meetup", {
8       method: "POST",
9       body: JSON.stringify(newMeetupData),
10      headers: {
11        "Content-Type": "application/json",
12      },
13    });
14    const data = await response.json();
15    console.log(data);
16    router.replace("/");
17  };
18  return <NewMeetupForm onAddMeetup={addMeetupHandler} />;
19};
20 export default NewMeetupPage;
21
```

Getting Data From The Database:

```
new-meetup.js      index.js  X
pages > index.js > ...
1 import MeetupList from "../components/meetups/MeetupList";
2 import { MongoClient } from "mongodb";
3 
4 const HomePage = (props) => {
5   return <MeetupList meetups={props.meetups} />;
6 };
7 /*
8 
9 only in component files that are in the pages folder.
10 Nextjs will look first for this kind of function
11 and will call it before the component function.
12 The code inside the function will not be executed on the client side and,
13 will never end up in the client side.
14 The function needs to always return an object.
15 The object must have a "props" field and this field must contain an object
16 this object with the field "props" than will be sent as our props
17 in our component.
18 kinda like in java or c (console arguments) main(argv, args).
19 The revalidate field handy in cases where we want to revalidate (get the data
20 sometimes the data we are working with can be out-dated quickly therefore
21 if we use revalidate (and as a value we need to give it a number)
22 we tell nextjs that we want to call getStaticProps every number of seconds
23 (the number we set as an argument. here we set it to 10 - every 10 seconds
24 (only if requests come's to this path)
25 */
26 export async function getStaticProps() {
27   // fetch data from database/api/ or whatever
28   const client = await MongoClient.connect(
29     "mongodb+srv://Remez:vbMk3le70okvPy0r@react-udemy.bbven7r.mongodb.net/"
30   );
31   const DataBase = client.db();
32   const meetupsCollection = DataBase.collection("meetups");
33   const meetups = await meetupsCollection.find().toArray();
34   client.close();
35 
36   return {
37     props: {
38       meetups: meetups.map((meetup) => {
39         return {
40           title: meetup.title,
41           address: meetup.address,
42           description: meetup.description,
43           image: meetup.image,
44           id: meetup._id.toString(),
45         };
46       }),
47     },
48     revalidate: 10,
49   };
50 }
```

Getting Meetup Details Data & Preparing Pages:

Adding "head" Metadata:

```
new-meetup.js      [js] index.js .../[meetupId]  X  [js] index.js pages      [js] index.js .../new-meetup
pages > [meetupId] > [js] index.js > [o] MeetupDetails
1  import { Fragment } from "react";
2  import { MongoClient, ObjectId } from "mongodb";
3  import Head from "next/head";
4  import MeetupDetail from "../../components/meetups/MeetupDetail";
5  const MeetupDetails = (props) => [
6    return (
7      <Fragment>
8        <Head>
9          <title>{props.meetupData.title}</title>
10         <meta name="description" content={props.meetupData.description} />
11       </Head>
12       <MeetupDetail
13         img={props.meetupData.image}
14         alt={props.meetupData.title}
15         address={props.meetupData.address}
16         description={props.meetupData.description}
17       />
18     </Fragment>
19   );
20 }
21
```

Section 25: Replacing Redux with React Hooks:

Finishing the Store Hook:



The screenshot shows a code editor window with a dark theme. The file is named 'store.js'. The code implements a global state management system using React's useState and useEffect hooks. It defines a 'useStore' hook that returns the current global state and a dispatch function. The dispatch function executes actions based on their action ID. It also maintains a list of listeners who receive updates via the 'useEffect' hook. The 'initStore' export initializes the global state with user actions and initial state.

```
src > hooks-store > store.js > ...
1 import { useState, useEffect } from "react";
2
3 let globalState = {};
4 let listeners = [];
5 /*
6 actions are in the form of:
7 {
8   |   actionId: (globalState) =>{} (function to execute according to the action)
9 }
10 */
11 let actions = {};
12
13 export const useStore = () => {
14   const setState = useState(globalState)[1];
15   listeners.push(setState);
16
17   const dispatch = (actionId) => {
18     const newState = actions[actionId](globalState);
19     globalState = { ...globalState, ...newState };
20
21     for (const listener of listeners) {
22       listener(globalState);
23     }
24   };
25
26   useEffect(() => {
27     listeners.push(setState);
28     return () => {
29       listeners = listeners.filter((listener) => listener !== setState);
30     };
31   }, [setState]);
32
33   return [globalState, dispatch];
34 };
35
36 export const initStore = (userActions, initState) => {
37   if (initState) {
38     globalState = { ...globalState, ...initState };
39   }
40   actions = { ...actions, ...userActions };
41 };
42
```

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.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** pane on the left:

 - OPEN EDITORS**: Shows 'App.test.js' and 'App.js'.
 - REACT_UNITTEST**: Shows the project structure under 'src': node_modules, public, components, App.css, App.js, App.test.js (highlighted with a red border), index.css, index.js, logo.svg, setupTests.js, package-lock.json, and package.json.

App.test.js content (highlighted area):

```
src > App.test.js > ...
1 import { render, screen } from "@testing-library/react";
2 import App from "./App";
3
4 /*
5  * test - takes two arguments.
6  * first argument: the name of the test (to identify the test).
7  * second argument a function that holds the code of the actual test:
8 */
9
10 test("renders learn react link", () => {
11   render(<App />);
12   const linkElement = screen.getByText(/learn react/i);
13   expect(linkElement).toBeInTheDocument();
14 });
15
```

Testing User Interaction & State:

```
JS Greeting.js      Greeting.test.js X
src > components > Greeting.test.js > describe("Greeting component") callback > test("renders hello world as a text")
1 import userEvent from "@testing-library/user-event";
2 import Greeting from "./Greeting";
3 import { render, screen } from "@testing-library/react";
4
5 // to describe our test suit we can give it a name
6 // Using "describe()"
7
8 describe("Greeting component", () => {
9   test("renders hello world as a text", () => {
10     // Arrange - Set up the test data, test conditions and test environment.
11     render(<Greeting />);
12
13     // Act - Run the logic that should be tested (e.g execute function).
14     // In this example there is no Act part.
15
16     // Assert - Compare execution results with expected results.
17     // To "getByText()" we can add another parameter { exact: false }.
18     const helloWorldElement = screen.getByText("Hello World!");
19     expect(helloWorldElement).toBeInTheDocument();
20   });
21
22   test("renders text before click", () => {
23     render(<Greeting />);
24     const pTextElement = screen.getByText("Its good to see", { exact: false });
25     expect(pTextElement).toBeInTheDocument();
26   });
27
28   test("render text after click", () => {
29     // Arrange
30     render(<Greeting />);
31
32     // Act
33     const buttonElement = screen.getByRole("button");
34     userEvent.click(buttonElement);
35
36     // Assert
37     const pTextElement = screen.getByText("Change!");
38     expect(pTextElement).toBeInTheDocument();
39   });
40 });
41
```

Testing Connected Components:

Regular - can test with render nested components !

The screenshot shows a code editor with the file `Greeting.js` open. The code uses the `useState` hook from React to manage state. It imports `Output` from `./Output`. The component `Greeting` returns a `div` containing an `h2` element with the text "Hello World!". It also contains two `<Output>` components. The first `<Output>` is conditionally rendered when `changedText` is false, displaying the message "Its good to see you!". The second `<Output>` is conditionally rendered when `changedText` is true, displaying the message "Change!". A button with the label "Change Text" has an `onClick` handler that calls the `changeTextHandler` function, which sets `changedText` to true.

```
src > components > Greeting.js > Greeting
1 import { useState } from "react";
2 import Output from "./Output";
3 const Greeting = () => {
4     const [changedText, setChangedText] = useState(false);
5     const changeTextHandler = () => {
6         setChangedText(true);
7     };
8     return (
9         <div>
10            <h2>Hello World!</h2>
11            { !changedText && <Output>Its good to see you!</Output> }
12            {changedText && <Output>Change!</Output> }
13            <button onClick={changeTextHandler}>Change Text</button>
14        </div>
15    );
16 }
17
18 export default Greeting;
19
```

Testing Asynchronous Code:

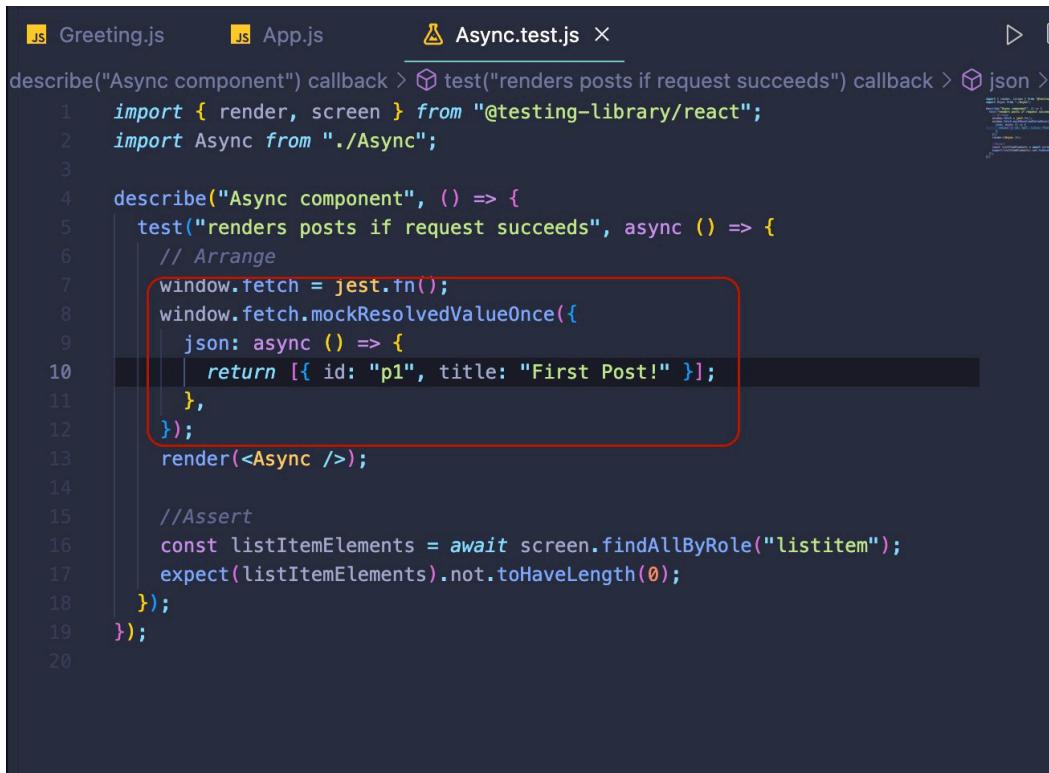
How to test when sending a Http request to an api (Asynchronous Code)

The screenshot shows a code editor with the file `Async.test.js` open. The code uses `@testing-library/react` and `Async` from `./Async`. It describes an "Async component" and tests it with a `test` block. The test checks if the component renders posts if the request succeeds. It uses `render` to arrange the component, and `expect` to assert that the list item elements have a length of 0.

```
describe("Async component") callback > test("renders posts if request succeeds") callback > listitem
1 import { render, screen } from "@testing-library/react";
2 import Async from "./Async";
3
4 describe("Async component", () => {
5     test("renders posts if request succeeds", async () => {
6         // Arrange
7         render(<Async />);
8
9         //Assert
10        const listItemElements = await screen.findAllByRole("listitem");
11        expect(listItemElements).toHaveLength(0);
12    });
13});
14
```

```
src > components > JS Async.js > Asynchronous
1 import { useEffect, useState } from "react";
2
3 const Async = () => [
4   const [posts, setPosts] = useState([]);
5
6   useEffect(() => {
7     fetch("https://jsonplaceholder.typicode.com/posts")
8       .then((response) => response.json())
9       .then((data) => {
10         setPosts(data);
11       });
12   }, []);
13
14   return (
15     <div>
16       <ul>
17         {posts.map((post) => (
18           <li key={post.id}>{post.title}</li>
19         ))}
20       </ul>
21     </div>
22   );
23 }
24
25 export default Async;
26
```

Working With Mocks:



```
js Greeting.js      js App.js      Async.test.js ×
describe("Async component") callback > ⚡ test("renders posts if request succeeds") callback > ⚡ json >
1   import { render, screen } from "@testing-library/react";
2   import Async from "./Async";
3
4   describe("Async component", () => {
5     test("renders posts if request succeeds", async () => {
6       // Arrange
7       window.fetch = jest.fn();
8       window.fetch.mockResolvedValueOnce({
9         json: async () => {
10           return [{ id: "p1", title: "First Post!" }];
11         },
12       });
13       render(<Async />);
14
15       //Assert
16       const listItemElements = await screen.findAllByRole("listitem");
17       expect(listItemElements).toHaveLength(0);
18     });
19   });
20
```

Section 27: React + TypeScript:

To install typeScript for a specific project run in terminal: "npm 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.

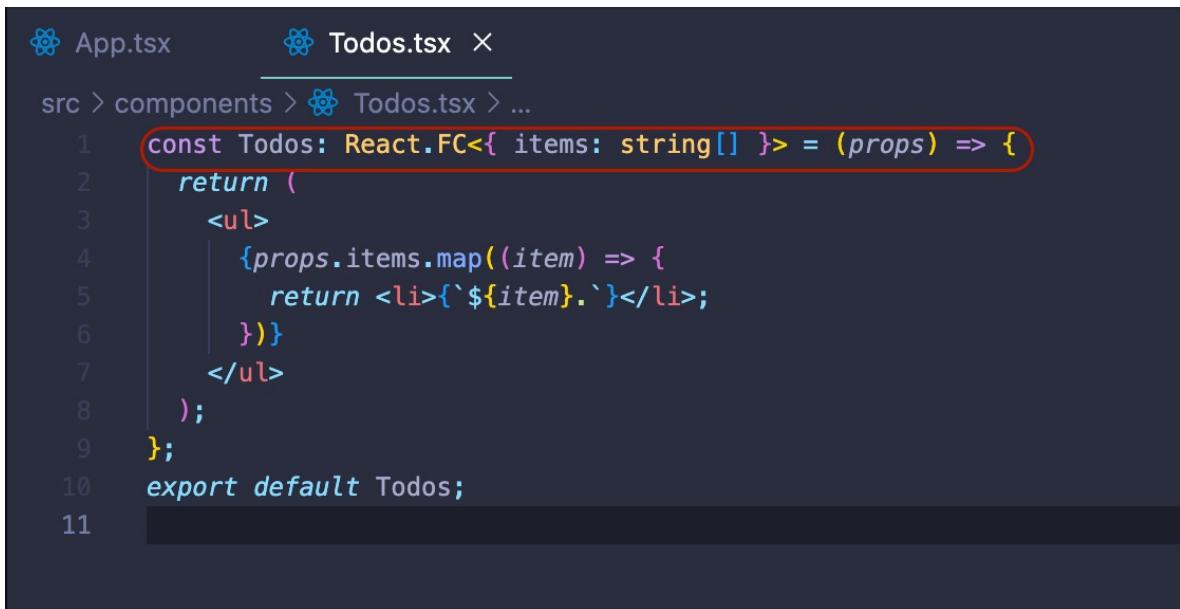
Installation

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 > ...
1 const Todos: React.FC<{ items: string[] }> = (props) => {
2   return (
3     <ul>
4       {props.items.map((item) => {
5         return <li>`${item}</li>;
6       })}
7     </ul>
8   );
9 }
10 export default Todos;
11
```

Adding a Data Model:



```
App.tsx Todos.tsx todo.ts
src > models > todo.ts > Todo > constructor
1 class Todo {
2   id: string;
3   text: string;
4   constructor(todoText: string) {
5     this.text = todoText;
6     this.id = new Date().toISOString();
7   }
8 }
9 export default Todo;
10
```

App.tsx X Todos.tsx todo.ts

```
src > App.tsx > ...
1 import Todos from "./components/Todos";
2 import Todo from "./models/todo";
3 function App() {
4     const todos = [new Todo("Remez"), new Todo("Ron"), new Todo("Avihai")];
5     // const dummy.todos = ["Remez", "Dor", "Avihai", "Ron"];
6     return (
7         <div>
8             <Todos items={todos} />
9         </div>
10    );
11 }
12
13 export default App;
```

App.tsx Todos.tsx todo.ts

```
src > components > Todos.tsx > Todos > props.items.map() callback
1 import Todo from "../models/todo";
2
3 const Todos: React.FC<{ items: Todo[] }> = (props) => {
4     return (
5         <ul>
6             {props.items.map((item) => {
7                 return <li key={item.id}>{`name: ${item.text}`}</li>;
8             })}
9         </ul>
10    );
11 };
12 export default Todos;
13
```