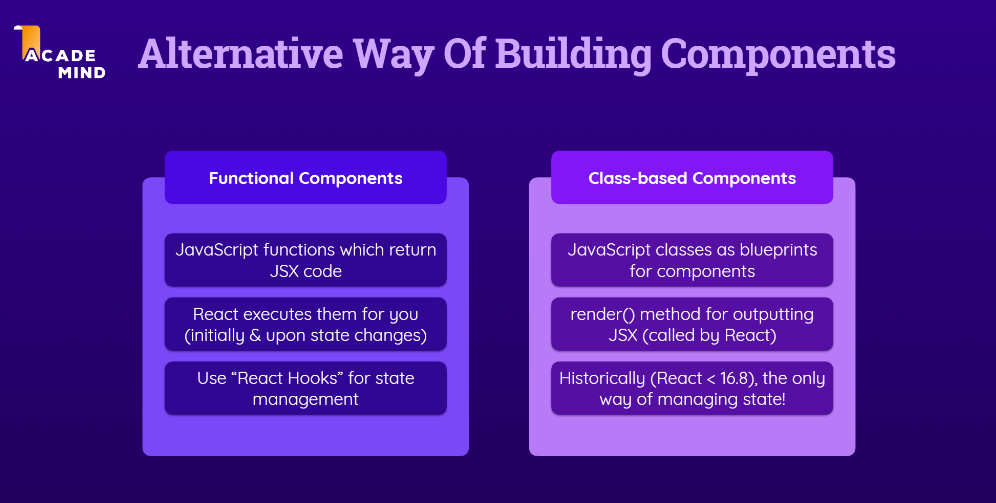
# Simple Components

# 

### Stateful vs Stateless Components



### Fragment In React, <Fragment> is a component that allows you to group multiple elements without adding an extra DOM element. This is useful when you don't want to create unnecessary HTML tags in the DOM structure. For example, if you want to return several elements from a component but don't want to wrap them in an additional div or another container, you can use <Fragment>.

| return (  <Fragment>  <AddUser onAddUser={addUserHandler} />  <UsersList users={usersList} />  </Fragment> ); |
| --- |

Or

| import React, { useContext } from "react"; import Login from "./components/Login/Login"; import Home from "./components/Home/Home"; import MainHeader from "./components/MainHeader/MainHeader"; import AuthContext from "./store/auth-context";  function App() {  const ctx = useContext(AuthContext);  return (  <React.Fragment>  <MainHeader />  <main>  {!ctx.isLoggedIn && <Login />}  {ctx.isLoggedIn && <Home />}  </main>  </React.Fragment>  ); }  export default App; |
| --- |

### 

### Props examples

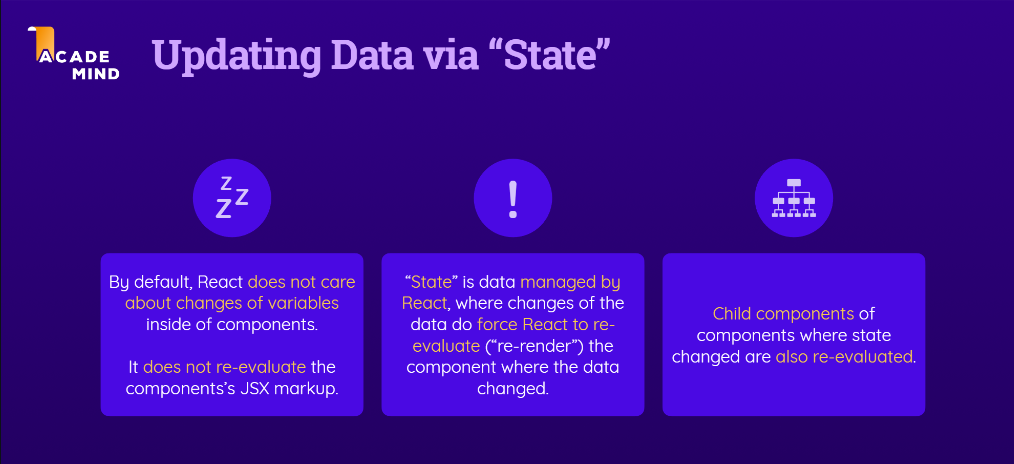
| <button  className={activeContentIndex === 0 ? "active" : "”}  onClick={() => setActiveContentIndex(0)} >  Why React? </button>; |
| --- |

### 

# State

| const [activeContentIndex, setActiveContentIndex] = useState(0); |
| --- |

### Updating State



| //Update the state  setUserInput({  ...userInput,  enteredTitle: event.target.value, }); |
| --- |

| //Updating State based on previous state  setUserInput((prevState) => {  return { ...prevState, enteredTitle: event.target.value }; }); |
| --- |

Another example

| const [usersList, setUsersList] = useState([]);   const addUserHandler = (uName, uAge) => {  setUsersList((prevUsersList) => {  return [  ...prevUsersList,  { name: uName, age: uAge, id: Math.random().toString() },  ];  });  }; |
| --- |

### 

### Lists Examples

| <div id="tab-content">  <ul>  {content[activeContentIndex].map((item) => (  <li key={item}>{item}</li>  ))}  </ul> </div>; |
| --- |

**Another example**

| **<ul>  {props.users.map((user) => (  <li key={user.id}>  {user.name} ({user.age} years old)  </li>  ))} </ul>** |
| --- |

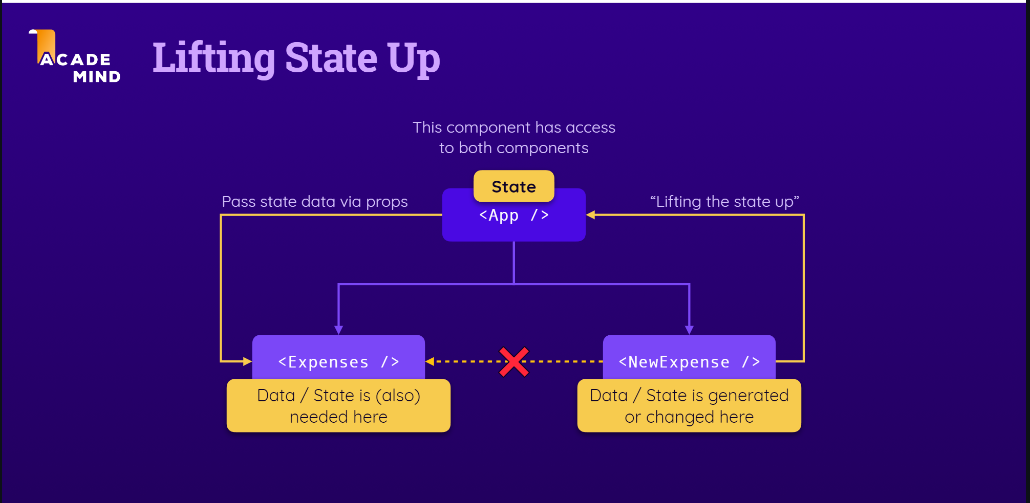
| const ResultsTable = (props) => {  return (  <table className={classes.result}>  <thead>  <tr>  <th>Year</th>  <th>Total Savings</th>  <th>Interest (Year)</th>  <th>Total Interest</th>  <th>Invested Capital</th>  </tr>  </thead>  <tbody>  {props.data.map((yearData) => (  <tr key={yearData.year}>  <td>{yearData.year}</td>  <td>{formatter.format(yearData.savingsEndOfYear)}</td>  <td>{formatter.format(yearData.yearlyInterest)}</td>  <td>  {formatter.format(  yearData.savingsEndOfYear -  props.initialInvestment -  yearData.yearlyContribution \* yearData.year,  )}  </td>  <td>  {formatter.format(  props.initialInvestment +  yearData.yearlyContribution \* yearData.year,  )}  </td>  </tr>  ))}  </tbody>  </table>  ); };  export default ResultsTable; |
| --- |

### 

### 

### 

### Lift the State UP



#### Example

| App.js  import React from 'react';  import NewExpense from './components/NewExpense/NewExpense'; import Expenses from './components/Expenses/Expenses';  const App = () => {  const expenses = [  {  id: 'e1',  title: 'Toilet Paper',  amount: 94.12,  date: new Date(2020, 7, 14),  },  { id: 'e2', title: 'New TV', amount: 799.49, date: new Date(2021, 2, 12) },  {  id: 'e3',  title: 'Car Insurance',  amount: 294.67,  date: new Date(2021, 2, 28),  },  {  id: 'e4',  title: 'New Desk (Wooden)',  amount: 450,  date: new Date(2021, 5, 12),  },  ];   const addExpenseHandler = expense => {  console.log('In App.js');  console.log(expense);  };   return (  <div>  <NewExpense onAddExpense={addExpenseHandler} />  <Expenses items={expenses} />  </div>  ); }  export default App; |
| --- |

| NewExpense.js |
| --- |

| import React from "react";  import ExpenseForm from "./ExpenseForm"; import "./NewExpense.css";  const NewExpense = (props) => {  const saveExpenseDataHandler = (enteredExpenseData) => {  const expenseData = {  ...enteredExpenseData,  id: Math.random().toString(),  };  props.onAddExpense(expenseData);  };   return (  <div className="new-expense">  <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />  </div>  ); };  export default NewExpense; |
| --- |

| ExpenseForm.js |
| --- |

| ExpenseForm.js  import React, { useState } from "react";  import "./ExpenseForm.css";  const ExpenseForm = (props) => {  const [enteredTitle, setEnteredTitle] = useState("");  const [enteredAmount, setEnteredAmount] = useState("");  const [enteredDate, setEnteredDate] = useState("");  // const [userInput, setUserInput] = useState({  // enteredTitle: '',  // enteredAmount: '',  // enteredDate: '',  // });   const titleChangeHandler = (event) => {  setEnteredTitle(event.target.value);  // setUserInput({  // ...userInput,  // enteredTitle: event.target.value,  // });  // setUserInput((prevState) => {  // return { ...prevState, enteredTitle: event.target.value };  // });  };   const amountChangeHandler = (event) => {  setEnteredAmount(event.target.value);  // setUserInput({  // ...userInput,  // enteredAmount: event.target.value,  // });  };   const dateChangeHandler = (event) => {  setEnteredDate(event.target.value);  // setUserInput({  // ...userInput,  // enteredDate: event.target.value,  // });  };   const submitHandler = (event) => {  event.preventDefault();   const expenseData = {  title: enteredTitle,  amount: enteredAmount,  date: new Date(enteredDate),  };   props.onSaveExpenseData(expenseData);  setEnteredTitle("");  setEnteredAmount("");  setEnteredDate("");  };   return (  <form onSubmit={submitHandler}>  <div className="new-expense\_\_controls">  <div className="new-expense\_\_control">  <label>Title</label>  <input  type="text"  value={enteredTitle}  onChange={titleChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Amount</label>  <input  type="number"  min="0.01"  step="0.01"  value={enteredAmount}  onChange={amountChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Date</label>  <input  type="date"  min="2019-01-01"  max="2022-12-31"  value={enteredDate}  onChange={dateChangeHandler}  />  </div>  </div>  <div className="new-expense\_\_actions">  <button type="submit">Add Expense</button>  </div>  </form>  ); };  export default ExpenseForm; |
| --- |

Same example with photos / more readable/  
  
App.js  


NewExpense.js  
  

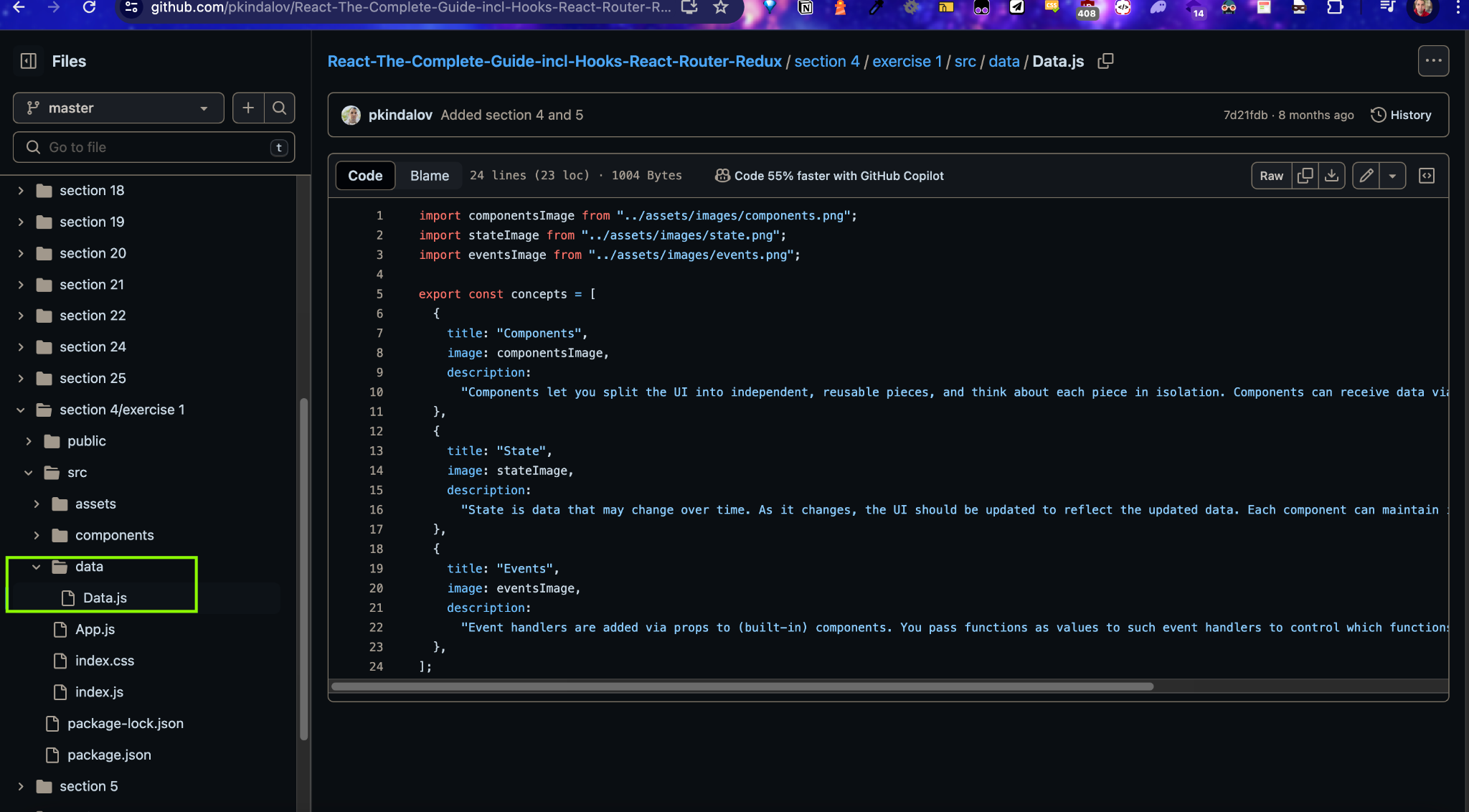

ExpenseForm.js - part 1



ExpenseForm.js - part 2



Where to put our Data



### How to use our Data

| import Header from "./components/Header/Header"; import ConceptItem from "./components/Concepts/ConceptItem"; import ConceptList from "./components/Concepts/ConceptList"; import { concepts } from "./data/Data";  function App() {  return (  <div>  <Header />  <ConceptList id="concepts">  {concepts.map((item, index) => (  <ConceptItem  key={index}  id={index}  image={item.image}  title={item.title}  description={item.description}  nameOfClass={"concept"}  />  ))}  </ConceptList>  </div>  ); }  export default App; |
| --- |

Other

### Currency formatter

| const formatter = new Intl.NumberFormat("en-En", {  style: "currency",  currency: "USD",  minimumFractionDigits: 2,  maximumFractionDigits: 2, }); |
| --- |

# CSS

### Modules

| import classes from "./ResultsTable.module.css";  <table className={classes.result}> |
| --- |

| <header className={classes["main-header"]}> |
| --- |

Forms

### Simple basic form

| import React, { useState } from "react";  import "./ExpenseForm.css";  const ExpenseForm = () => {  const [enteredTitle, setEnteredTitle] = useState("");  const [enteredAmount, setEnteredAmount] = useState("");  const [enteredDate, setEnteredDate] = useState("");   const titleChangeHandler = (event) => {  setEnteredTitle(event.target.value);  };   const amountChangeHandler = (event) => {  setEnteredAmount(event.target.value);  };   const dateChangeHandler = (event) => {  setEnteredDate(event.target.value);  };   return (  <form>  <div className="new-expense\_\_controls">  <div className="new-expense\_\_control">  <label>Title</label>  <input type="text" onChange={titleChangeHandler} />  </div>  <div className="new-expense\_\_control">  <label>Amount</label>  <input  type="number"  min="0.01"  step="0.01"  onChange={amountChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Date</label>  <input  type="date"  min="2019-01-01"  max="2022-12-31"  onChange={dateChangeHandler}  />  </div>  </div>  <div className="new-expense\_\_actions">  <button type="submit">Add Expense</button>  </div>  </form>  ); };  export default ExpenseForm; |
| --- |

### Simple Basic form 2

| import React, { useState } from "react";  import "./ExpenseForm.css";  const ExpenseForm = () => {  const [enteredTitle, setEnteredTitle] = useState("");  const [enteredAmount, setEnteredAmount] = useState("");  const [enteredDate, setEnteredDate] = useState("");  // const [userInput, setUserInput] = useState({  // enteredTitle: '',  // enteredAmount: '',  // enteredDate: '',  // });   const titleChangeHandler = (event) => {  setEnteredTitle(event.target.value);  // setUserInput({  // ...userInput,  // enteredTitle: event.target.value,  // });  // setUserInput((prevState) => {  // return { ...prevState, enteredTitle: event.target.value };  // });  };   const amountChangeHandler = (event) => {  setEnteredAmount(event.target.value);  // setUserInput({  // ...userInput,  // enteredAmount: event.target.value,  // });  };   const dateChangeHandler = (event) => {  setEnteredDate(event.target.value);  // setUserInput({  // ...userInput,  // enteredDate: event.target.value,  // });  };   return (  <form>  <div className="new-expense\_\_controls">  <div className="new-expense\_\_control">  <label>Title</label>  <input type="text" onChange={titleChangeHandler} />  </div>  <div className="new-expense\_\_control">  <label>Amount</label>  <input  type="number"  min="0.01"  step="0.01"  onChange={amountChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Date</label>  <input  type="date"  min="2019-01-01"  max="2022-12-31"  onChange={dateChangeHandler}  />  </div>  </div>  <div className="new-expense\_\_actions">  <button type="submit">Add Expense</button>  </div>  </form>  ); };  export default ExpenseForm; |
| --- |

### Simple Basic Form Submission

| import React, { useState } from "react";  import "./ExpenseForm.css";  const ExpenseForm = () => {  const [enteredTitle, setEnteredTitle] = useState("");  const [enteredAmount, setEnteredAmount] = useState("");  const [enteredDate, setEnteredDate] = useState("");  // const [userInput, setUserInput] = useState({  // enteredTitle: '',  // enteredAmount: '',  // enteredDate: '',  // });   const titleChangeHandler = (event) => {  setEnteredTitle(event.target.value);  // setUserInput({  // ...userInput,  // enteredTitle: event.target.value,  // });  // setUserInput((prevState) => {  // return { ...prevState, enteredTitle: event.target.value };  // });  };   const amountChangeHandler = (event) => {  setEnteredAmount(event.target.value);  // setUserInput({  // ...userInput,  // enteredAmount: event.target.value,  // });  };   const dateChangeHandler = (event) => {  setEnteredDate(event.target.value);  // setUserInput({  // ...userInput,  // enteredDate: event.target.value,  // });  };   const submitHandler = (event) => {  event.preventDefault();   const expenseData = {  title: enteredTitle,  amount: enteredAmount,  date: new Date(enteredDate),  };   console.log(expenseData);  };   return (  <form onSubmit={submitHandler}>  <div className="new-expense\_\_controls">  <div className="new-expense\_\_control">  <label>Title</label>  <input type="text" onChange={titleChangeHandler} />  </div>  <div className="new-expense\_\_control">  <label>Amount</label>  <input  type="number"  min="0.01"  step="0.01"  onChange={amountChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Date</label>  <input  type="date"  min="2019-01-01"  max="2022-12-31"  onChange={dateChangeHandler}  />  </div>  </div>  <div className="new-expense\_\_actions">  <button type="submit">Add Expense</button>  </div>  </form>  ); };  export default ExpenseForm; |
| --- |

### Two way binding form example

| import React, { useState } from "react";  import "./ExpenseForm.css";  const ExpenseForm = () => {  const [enteredTitle, setEnteredTitle] = useState("");  const [enteredAmount, setEnteredAmount] = useState("");  const [enteredDate, setEnteredDate] = useState("");  // const [userInput, setUserInput] = useState({  // enteredTitle: '',  // enteredAmount: '',  // enteredDate: '',  // });   const titleChangeHandler = (event) => {  setEnteredTitle(event.target.value);  // setUserInput({  // ...userInput,  // enteredTitle: event.target.value,  // });  // setUserInput((prevState) => {  // return { ...prevState, enteredTitle: event.target.value };  // });  };   const amountChangeHandler = (event) => {  setEnteredAmount(event.target.value);  // setUserInput({  // ...userInput,  // enteredAmount: event.target.value,  // });  };   const dateChangeHandler = (event) => {  setEnteredDate(event.target.value);  // setUserInput({  // ...userInput,  // enteredDate: event.target.value,  // });  };   const submitHandler = (event) => {  event.preventDefault();   const expenseData = {  title: enteredTitle,  amount: enteredAmount,  date: new Date(enteredDate),  };   console.log(expenseData);  setEnteredTitle("");  setEnteredAmount("");  setEnteredDate("");  };   return (  <form onSubmit={submitHandler}>  <div className="new-expense\_\_controls">  <div className="new-expense\_\_control">  <label>Title</label>  <input  type="text"  value={enteredTitle}  onChange={titleChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Amount</label>  <input  type="number"  min="0.01"  step="0.01"  value={enteredAmount}  onChange={amountChangeHandler}  />  </div>  <div className="new-expense\_\_control">  <label>Date</label>  <input  type="date"  min="2019-01-01"  max="2022-12-31"  value={enteredDate}  onChange={dateChangeHandler}  />  </div>  </div>  <div className="new-expense\_\_actions">  <button type="submit">Add Expense</button>  </div>  </form>  ); };  export default ExpenseForm; |
| --- |

HOOKS

### useRef

В React, useRef е hook, който предоставя начин да се създаде референция към DOM елементи или да се съхранява стойност, която няма да причини повторно рендиране на компонента при промяна. useRef връща обект с едно поле .current, което може да бъде използвано за достъп до DOM елемента или за съхранение на стойност.

In React, useRef is a hook that provides a way to create a reference to DOM elements or store a value that will not cause the component to re-render when it changes. useRef returns an object with a single field, .current, which can be used to access the DOM element or store a value.  
  
Some examples  
  
Here are two main scenarios where useRef is used:

1.Accessing DOM elements:  
  
С useRef можете да получите референция към DOM елемент и да манипулирате директно този елемент, без да използвате document.querySelector или други методи за селекция.

With useRef, you can get a reference to a DOM element and manipulate it directly without using document.querySelector or other selection methods.

| import React, { useRef, useEffect } from 'react';  function MyComponent() {  const inputRef = useRef(null);   useEffect(() => {  // Focus on the input element when the component mounts  inputRef.current.focus();  }, []);   return <input ref={inputRef} type="text" />; export default MyComponent; |
| --- |

2.Storing values between renders:

useRef може да се използва за съхранение на променливи, които няма да причинят повторно рендиране при промяна. Това е полезно за съхранение на предишни стойности или състояния, които не са необходими за рендерирането

useRef can be used to store variables that won't cause a re-render when changed. This is useful for storing previous values or states that are not needed for rendering.

| import React, { useRef, useState, useEffect } from 'react';  function TimerComponent() {  const [count, setCount] = useState(0);  const timerIdRef = useRef(null);   useEffect(() => {  timerIdRef.current = setInterval(() => {  setCount(prevCount => prevCount + 1);  }, 1000);   return () => {  clearInterval(timerIdRef.current);  };  }, []);   return <div>Count: {count}</div>; }  export default TimerComponent; |
| --- |

More examples

| import React, { useState, useRef } from "react";  import Card from "../UI/Card"; import Button from "../UI/Button"; import ErrorModal from "../UI/ErrorModal"; import Wrapper from "../Helpers/Wrapper"; import classes from "./AddUser.module.css";  const AddUser = (props) => {  const nameInputRef = useRef();  const ageInputRef = useRef();  const [error, setError] = useState();   const addUserHandler = (event) => {  event.preventDefault();  const enteredName = nameInputRef.current.value;  const enteredUserAge = ageInputRef.current.value;  if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {  setError({  title: "Invalid input",  message: "Please enter a valid name and age (non-empty values).",  });  return;  }  if (+enteredUserAge < 1) {  setError({  title: "Invalid age",  message: "Please enter a valid age (> 0).",  });  return;  }  props.onAddUser(enteredName, enteredUserAge);  nameInputRef.current.value = "";  ageInputRef.current.value = "";  };   const errorHandler = () => {  setError(null);  };   return (  <Wrapper>  {error && (  <ErrorModal  title={error.title}  message={error.message}  onConfirm={errorHandler}  />  )}  <Card className={classes.input}>  <form onSubmit={addUserHandler}>  <label htmlFor="username">Username</label>  <input id="username" type="text" ref={nameInputRef} />  <label htmlFor="age">Age (Years)</label>  <input id="age" type="number" ref={ageInputRef} />  <Button type="submit">Add User</Button>  </form>  </Card>  </Wrapper>  ); };  export default AddUser; |
| --- |

### useContext

В React, useContext е hook, който ви позволява да използвате контекста в рамките на функционален компонент. Контекстът в React предоставя начин за предаване на данни през компонентното дърво, без да е необходимо ръчно да се предават пропсове на всяко ниво.

In React, useContext is a hook that allows you to use context within a functional component. Context in React provides a way to pass data through the component tree without having to manually pass props down at every level

1Creating a context./ Създаване на контекст /:

Първо създавате контекст с помощта на React.createContext. Това създава обект с два компонента: Provider и Consumer.

First, you create a context using React.createContext. This creates an object with two components: Provider and Consumer.

| import React from 'react'; const MyContext = React.createContext(); |
| --- |

Example 2

| import { createContext, useState, useEffect } from "react";  const AuthContext = createContext({  isLoggedIn: false,  onLogout: () => {},  onLogin: (email, password) => {}, }); |
| --- |

2.Providing context: /Осигуряване на контекст:/

Използвате компонента Provider, за да предоставите стойност на контекста. Всички компоненти, които са деца на този Provider, ще имат достъп до стойността на контекст

You use the Provider component to supply a context value. All components that are children of this Provider will have access to the context value.

| import React from 'react'; import MyComponent from './MyComponent';  const MyContext = React.createContext();  function App() {  return (  <MyContext.Provider value={{ message: 'Hello from context!' }}>  <MyComponent />  </MyContext.Provider>  ); }  export default App; |
| --- |

Example 2

| export const AuthContextProvider = (props) => {  const [isLoggedIn, setIsLoggedIn] = useState(false);   useEffect(() => {  const storedUserLoggedInInformation = localStorage.getItem("isLoggedIn");  if (storedUserLoggedInInformation === "1") {  setIsLoggedIn(true);  }  }, []);   const logoutHandler = () => {  localStorage.removeItem("isLoggedIn");  setIsLoggedIn(false);  };   const loginHandler = () => {  localStorage.setItem("isLoggedIn", "1");  setIsLoggedIn(true);  };   return (  <AuthContext.Provider  value={{  isLoggedIn: isLoggedIn,  onLogout: logoutHandler,  onLogin: loginHandler,  }}  >  {props.children}  </AuthContext.Provider>  ); };  export default AuthContext; |
| --- |

3.Using context in a component /Използване на контекста в компонент/

Във функционален компонент използвате useContext, за да получите достъп до стойността на контекста.

Inside a functional component, you use useContext to access the context value.

| import React from 'react'; import MyComponent from './MyComponent';  const MyContext = React.createContext();  function App() {  return (  <MyContext.Provider value={{ message: 'Hello from context!' }}>  <MyComponent />  </MyContext.Provider>  ); }  export default App; |
| --- |

Example of a complete implementation: /Пример за пълна имплементация:/

| import React from "react";  const MyContext = React.createContext();  function App() {  return (  <MyContext.Provider value={{ message: "Hello from context!" }}>  <MyComponent />  </MyContext.Provider>  ); }  function MyComponent() {  const contextValue = useContext(MyContext);   return <div>{contextValue.message}</div>; }  export default App; |
| --- |

Other examples

#### Creating and exporting context and context provider

| import { createContext, useState, useEffect } from "react";  const AuthContext = createContext({  isLoggedIn: false,  onLogout: () => {},  onLogin: (email, password) => {}, });  export const AuthContextProvider = (props) => {  const [isLoggedIn, setIsLoggedIn] = useState(false);  useEffect(() => {  const storedUserLoggedInInformation = localStorage.getItem("isLoggedIn");   if (storedUserLoggedInInformation === "1") {  setIsLoggedIn(true);  }  }, []);   const logoutHandler = () => {  localStorage.removeItem("isLoggedIn");  setIsLoggedIn(false);  };   const loginHandler = () => {  localStorage.setItem("isLoggedIn", "1");  setIsLoggedIn(true);  };   return (  <AuthContext.Provider  value={{  isLoggedIn: isLoggedIn,  onLogout: logoutHandler,  onLogin: loginHandler,  }}  >  {props.children}  </AuthContext.Provider>  ); };  export default AuthContext; |
| --- |

| import React, { useContext } from "react"; import Login from "./components/Login/Login"; import Home from "./components/Home/Home"; import MainHeader from "./components/MainHeader/MainHeader"; import AuthContext from "./store/auth-context";  function App() {  const ctx = useContext(AuthContext);  return (  <React.Fragment>  <MainHeader />  <main>  {!ctx.isLoggedIn && <Login />}  {ctx.isLoggedIn && <Home />}  </main>  </React.Fragment>  ); }  export default App; |
| --- |

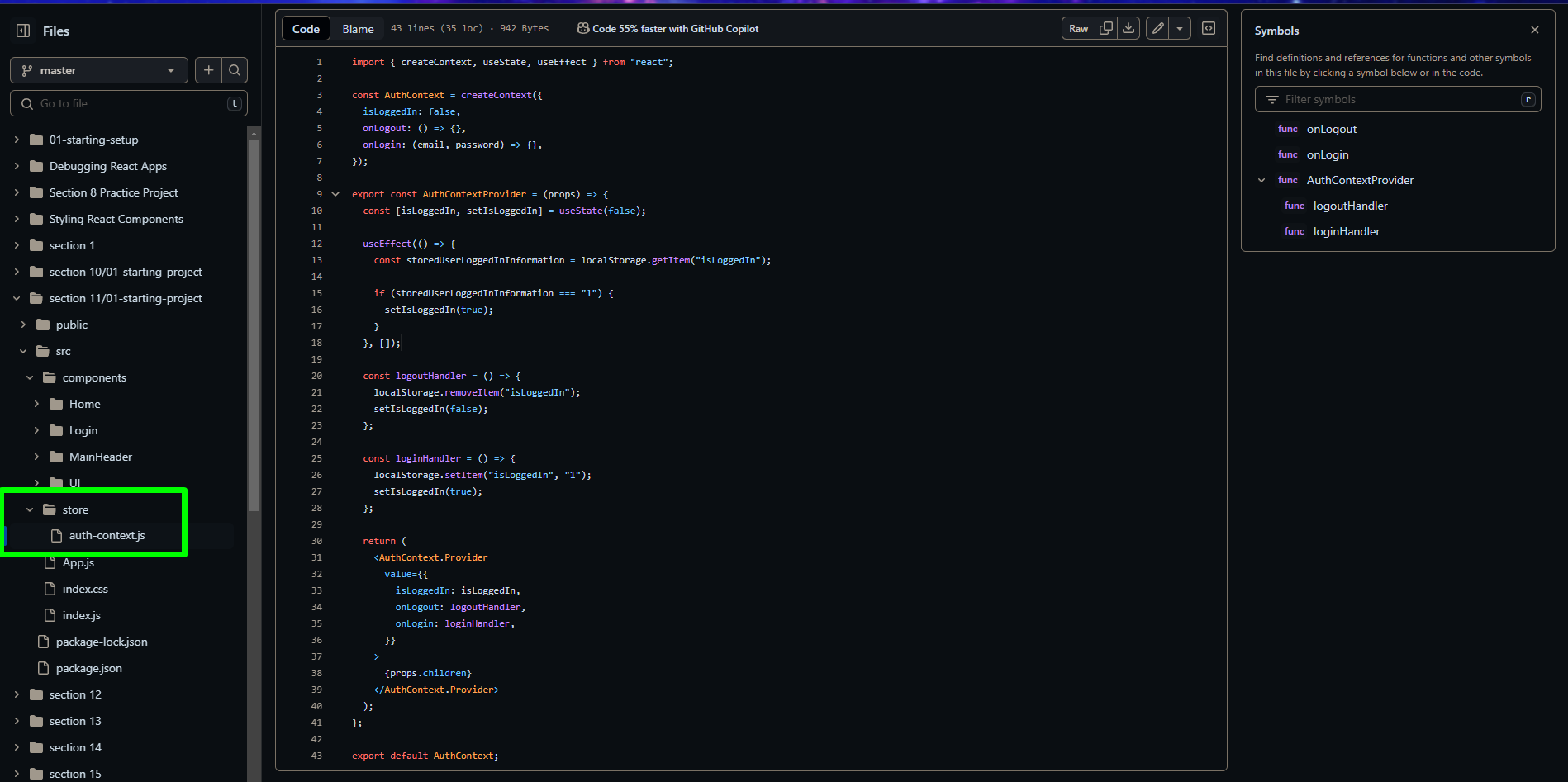
| import React, { useContext } from "react";  import AuthContext from "../../store/auth-context"; import classes from "./Navigation.module.css";  const Navigation = () => {  const ctx = useContext(AuthContext);  return (  <nav className={classes.nav}>  <ul>  {ctx.isLoggedIn && (  <li>  <a href="/">Users</a>  </li>  )}  {ctx.isLoggedIn && (  <li>  <a href="/">Admin</a>  </li>  )}  {ctx.isLoggedIn && (  <li>  <button onClick={ctx.onLogout}>Logout</button>  </li>  )}  </ul>  </nav>  ); };  export default Navigation; |
| --- |

| import React, { useContext } from "react";  import Card from "../UI/Card/Card"; import classes from "./Home.module.css"; import AuthContext from "../../store/auth-context"; import Button from "../UI/Button/Button";  const Home = () => {  const authCtx = useContext(AuthContext);  return (  <Card className={classes.home}>  <h1>Welcome back!</h1>  <Button onClick={authCtx.onLogout}>Logout</Button>  </Card>  ); };  export default Home; |
| --- |

#### 

#### 

#### TODO: PUT NEW IMAGE HERE FROM CARBON LATER



### useReducer

In React, useReducer is a hook that provides an alternative to useState for managing complex state logic. It is especially useful when the state depends on previous state values or when the state logic is too complex to be handled by multiple useState calls. useReducer works by taking a reducer function and an initial state, then returning the current state and a dispatch function to update the state.

Here's how useReducer works:

1. Defining the reducer function:  
   The reducer function determines how the state should be updated based on the action received. It takes the current state and an action as arguments and returns a new state.

| const reducer = (state, action) => {  switch (action.type) {  case "increment":  return { count: state.count + 1 };  case "decrement":  return { count: state.count - 1 };  default:  throw new Error("Unknown action type");  } }; |
| --- |

2. Using useReducer in a component:

You call useReducer with the reducer function and an initial state. It returns the current state and a dispatch function that you can use to send actions to the reducer.

| import React, { useReducer } from "react";  const initialState = { count: 0 };  function Counter() {  const [state, dispatch] = useReducer(reducer, initialState);   return (  <div>  <p>Count: {state.count}</p>  <button onClick={() => dispatch({ type: "increment" })}>Increment</button>  <button onClick={() => dispatch({ type: "decrement" })}>Decrement</button>  </div>  ); }  export default Counter; |
| --- |

#### Examples

Example of a complete implementation:

| import React, { useReducer } from "react";  const initialState = { count: 0 };  const reducer = (state, action) => {  switch (action.type) {  case "increment":  return { count: state.count + 1 };  case "decrement":  return { count: state.count - 1 };  default:  throw new Error("Unknown action type");  } };  function Counter() {  const [state, dispatch] = useReducer(reducer, initialState);   return (  <div>  <p>Count: {state.count}</p>  <button onClick={() => dispatch({ type: "increment" })}>Increment</button>  <button onClick={() => dispatch({ type: "decrement" })}>Decrement</button>  </div>  ); }  export default Counter; |
| --- |

In this example:

1. We define an initial state with count: 0.
2. We define a reducer function that handles increment and decrement actions.
3. We use useReducer in the Counter component, passing the reducer function and initial state. The state and dispatch function are returned by useReducer.
4. We use the dispatch function to send actions to the reducer, which updates the state accordingly.

Summary: useReducer is a React hook that manages complex state logic by using a reducer function. It is useful for state management when the logic involves multiple sub-values or depends on previous state values, providing a more structured and scalable approach compared to useState.

One more example

| import React, {  useState,  useEffect,  useReducer,  useContext,  useRef, } from "react";  import Card from "../UI/Card/Card"; import Button from "../UI/Button/Button"; import AuthContext from "../../store/auth-context"; import Input from "../UI/Input/Input"; import classes from "./Login.module.css";  const emailReducer = (state, action) => {  if (action.type === "USER\_INPUT") {  return { value: action.val, isValid: action.val.includes("@") };  }  if (action.type === "INPUT\_BLUR") {  return { value: state.value, isValid: state.value.includes("@") };  }  return { value: "", isValid: false }; };  const passwordReduser = (state, action) => {  if (action.type === "USER\_PASSWORD\_INPUT") {  return { value: action.val, isValid: action.val.trim().length > 6 };  }  if (action.type === "PASSWORD\_INPUT\_BLUR") {  return { value: state.value, isValid: state.value.trim().length > 6 };  }  return { value: "", isValid: false }; };  const Login = (props) => {  // const [enteredEmail, setEnteredEmail] = useState("");  // const [emailIsValid, setEmailIsValid] = useState();  // const [enteredPassword, setEnteredPassword] = useState("");  // const [passwordIsValid, setPasswordIsValid] = useState();  const [formIsValid, setFormIsValid] = useState(false);   const [emailState, dispatchEmail] = useReducer(emailReducer, {  value: "",  isValid: null,  });   const [passwordState, dispatchPassword] = useReducer(passwordReduser, {  value: "",  isValid: null,  });   const authCtx = useContext(AuthContext);  const emailInputRef = useRef();  const passwordInputRef = useRef();   const { isValid: emailIsValid } = emailState;  const { isValid: passwordIsValid } = passwordState;   useEffect(() => {  const identifier = setTimeout(() => {  console.log("Checking form validity");  setFormIsValid(emailIsValid && passwordIsValid);  }, 500);   //Clean up function  return () => {  console.log("CLEANUP");  clearTimeout(identifier);  };  }, [emailIsValid, passwordIsValid]);   const emailChangeHandler = (event) => {  // setEnteredEmail(event.target.value);  dispatchEmail({ type: "USER\_INPUT", val: event.target.value });   // setFormIsValid(event.target.value.includes("@") && passwordState.isValid);  };   const passwordChangeHandler = (event) => {  // setEnteredPassword(event.target.value);  dispatchPassword({ type: "USER\_PASSWORD\_INPUT", val: event.target.value });   // setFormIsValid(  // event.target.value.trim().length > 6 && emailState.value.includes("@")  // );  };   const validateEmailHandler = () => {  // setEmailIsValid(emailState.isValid);  dispatchEmail({ type: "INPUT\_BLUR" });  };   const validatePasswordHandler = () => {  // setPasswordIsValid(enteredPassword.trim().length > 6);  dispatchPassword({ type: "PASSWORD\_INPUT\_BLUR" });  };   const submitHandler = (event) => {  event.preventDefault();  if (formIsValid) {  authCtx.onLogin(emailState.value, passwordState.value);  } else if (!emailIsValid) {  emailInputRef.current.focus();  } else {  passwordInputRef.current.focus();  }  };   return (  <Card className={classes.login}>  <form onSubmit={submitHandler}>  {/\* Lecturer version \*/}  <Input  ref={emailInputRef}  id="email"  label="E-Mail"  type="email"  isValid={emailIsValid}  value={emailState.value}  onChange={emailChangeHandler}  onBlur={validateEmailHandler}  />   {/\* My version \*/}  {/\* <Input  className={`${classes.control} ${  emailState.isValid === false ? classes.invalid : ""  }`}  htmlFor={"email"}  labelText={"E-Mail"}  type={"email"}  id={"email"}  value={emailState.value}  onChangeHandler={emailChangeHandler}  onBlurHandler={validateEmailHandler}  /> \*/}  {/\* <div  className={`${classes.control} ${  emailState.isValid === false ? classes.invalid : ""  }`}  >  <label htmlFor="email">E-Mail</label>  <input  type="email"  id="email"  value={emailState.value}  onChange={emailChangeHandler}  onBlur={validateEmailHandler}  />  </div> \*/}  <Input  ref={passwordInputRef}  id="password"  label="Password"  type="password"  isValid={passwordIsValid}  value={passwordState.value}  onChange={passwordChangeHandler}  onBlur={validatePasswordHandler}  />  {/\* <div  className={`${classes.control} ${  passwordState.isValid === false ? classes.invalid : ""  }`}  >  <label htmlFor="password">Password</label>  <input  type="password"  id="password"  value={passwordState.value}  onChange={passwordChangeHandler}  onBlur={validatePasswordHandler}  />  </div> \*/}  <div className={classes.actions}>  <Button type="submit" className={classes.btn}>  Login  </Button>  </div>  </form>  </Card>  ); };  export default Login; |
| --- |