

Web Programming

JavaScript – node.js - react

Lecture 6-1:
React Intro
30.11.

Stefan Noll
Manuel Fehrenbach
Winter Semester 22/23

Two Way Binding

- Two Way Binding:
 - Two way binding is needed if you want to handle **input** through `<input />` tags.
 - You need to set the **value** attribute to the value of **state-object** and you also need to have a **function** to handle **inputs** (**onChange** events) from the user.
Only if you have these two things in place have you implemented the `<input />` tag the right way in react and used **two way binding**
 - The **onChange** event passes an **event-object** to the function which handles the event.
Through this event-object you can **access** the value of the `<input />` on other properties of the `<input />` tag

```

7  const NewTask = (props) => {
8    const [title, setTitle] = useState("")
9    const navigate = useNavigate()

10
11    const onTitleChange = (e) => {
12      setTitle(e.target.value);
13    }
14

```

```

23  return(
24    <Content>
25      <div className='NewTask'>
26        <div className='NewTask__Content'>
27          <label>New Task Title:<input type="text" value={title} onChange={onTitleChange}/></label>
28          <button onClick={addNewTask}>Add</button>
29        </div>
30      </div>
31    </Content>
32  )
33 }

```

NewTask Component

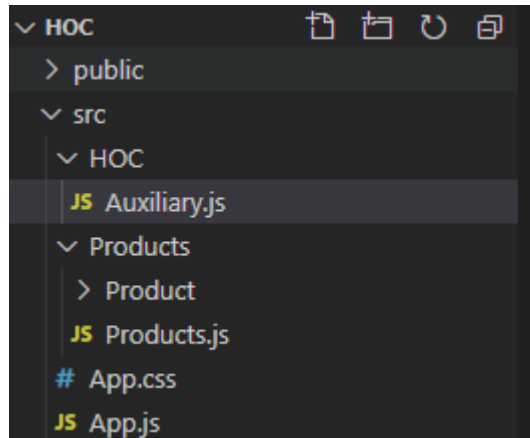
```
# Content.css 1 # index.css JS Content.js JS App.js JS ShowToDo.js # ShowToDo.css JS Task.js # Task.css JS NewTask.js X JS inc
src > components > NewTask > JS NewTask.js > ...
1 import { useState } from 'react'
2 import { useNavigate } from 'react-router-dom';
3 import Content from '../Content/Content'
4 import axios from "../../axios";
5 import './NewTask.css'
6
7 const NewTask = (props)=>{
8   const [title, setTitle] = useState("")
9   const navigate = useNavigate()
10
11   const onTitleChange = (e)=>{
12     setTitle(e.target.value);
13   }
14
15   const addNewTask = ()=>{
16     axios.post('/task',{title}).then((res)=>{
17       navigate("/");
18     }).catch((err)=>{
19       console.log(err)
20     })
21   }
22
23   return(
24     <Content>
25       <div className='NewTask'>
26         <div className='NewTask__Content'>
27           <label>New Task Title:<input type="text" value={title} onChange={onTitleChange}/></label>
28           <button onClick={addNewTask}>Add</button>
29         </div>
30       </div>
31     </Content>
32   )
33 }
34
35 export default NewTask;
```

New Task Title:

Add

HOC

- What are HOCs?
 - HOCs in React and JavaScript are the same: **Higher Order Functions**
 - We are only able to return one JSX-element in a react component, but sometimes even one JSX-element might throw off our CSS-design or the flow of our app.
For this problem we can use HOCs
 - As a component can return a JSX-element or an element which is a component, we can create pseudo tags with HOCs. These pseudo tags are accepted from react like normal JSX-elements but in truth do nothing
 - In most cases we create a seperated directory within the src directory only for HOCs.
The directory is also named **HOC**



```

JS Auxiliary.js X JS App.js
src > HOC > JS Auxiliary.js > [⌘] default
1  import React from "react";
2
3  const Aux = (props) =>{
4    |   return (props.children);
5  }
6
7  export default Aux;
  
```

```

JS Auxiliary.js JS App.js X
src > JS App.js > App > render
1  import React, { useState } from 'react';
2  import Products from '../Products/Products';
3  import Aux from '../HOC/Auxiliary';
4
5  class App extends Component{
6
7    state={ ...
8  }
9
10 toggleProductsHandler = ()=>{ ...
11 }
12
13 changeNameHandler = (event, id)=>{ ...
14 }
15
16 render(){
17   let products = null
18
19   if(this.state.showProducts){
20     products = (<Products products={this.state.product} change={this.changeNameHandler} />)
21   }
22
23   return (
24     <Aux className="App">
25       <h1>Ich bin eine React App</h1>
26       <button onClick={this.toggleProductsHandler}>Anzeigen/Verstecken von Produkten</button>
27       {products}
28       <Aux>Hallo</Aux>
29     </Aux>
30   );
31 }
32
33 export default App;
  
```


- Besides the normal HOC (Aux.js) there is also different kind of type for HOCs
- This kind of HOC starts always with a **with[Component Name]**
- This kind of HOC in most cases adds something to a component.
 - This method is usually used from third-party react packages

```
JS Auxiliary.js  JS WithClass.js X  JS App.js
src > HOC > JS WithClass.js > [e] withClass
1  import React from 'react.js'
2
3
4  const withClass = props =>{
5    <div className={props.classes}>{props.children}</div>
6  }
7
8
9  export default withClass;
```

- A third option for a HOC in react is to write function which expects a component as a parameter and returns JSX-element.
- As this kind of a HOC is not a component it must be treated as a „normal“ function and be used like one.
- This kind of HOC is used when you export a component.
The Component you want to export is the parameter of the HOC.
 - e.g.: `export default withClass(App,“ClassnameStyling“);`
- This kind of HOC often adds logic to a Component and is used from third party react packages
- With `{...props}` attributes are passed to the WrappedComponent as shown in the picture

```
import React from 'react';

const withClass = (WrappedComponent, className) =>{
  return props =>{
    <div className={className}><WrappedComponent {...props}/></div>
  }
}

export default withClass;
```

From To-Do exercise the Content.js

```
# Content.css 1    # index.css    JS Content.js X    JS
src > components > Content > JS Content.js > [🔗] default
1  import './Content.css'
2  const Content = (props) => {
3      return(
4          <div className='Content'>
5              {props.children}
6          </div>
7      )
8  }
9
10
11  export default Content;
```

React Fragment

- What are react fragments:
 - Besides HOCs there is a React build in functionality, which lets you return multiple React elements without adding unnecessary elements in the parent component
 - With fragments you are able to group a list of children and return these without adding nodes to the DOM
 - Fragments with the explicit **<React.Fragment>** Syntax may have keys, which we need, if we iterate through an array or map a collection to an array of fragments

```

src > Tables > JS Tables.js > ...
1 import Tableitems from "../TableItems/TableItems"
2 const tables = ()=>{
3
4   return (
5     <div>
6       <tables>
7         <thead>
8           <tr>
9             <td>A</td>
10            <td>B</td>
11            <td>C</td>
12          </tr>
13        </thead>
14        <tbody>
15          <Tableitems />
16        </tbody>
17      </tables>
18    </div>
19  )
20 }
21
22
23 export default tables;

```

```

src > Tables > TableItems > JS TableItems.js > ...
1 import React from 'react';
2 const tableitems = ()=>{
3   return(
4     <React.Fragment>
5       <tr>
6         <td>1</td>
7         <td>Test</td>
8         <td>Data</td>
9       </tr>
10 > <tr> ...
14 </tr>
15 > <tr> ...
19 </tr>
20 > <tr> ...
24 </tr>
25 </React.Fragment>
26 )
27 }
28 export default tableitems;

```

Keyed Fragments

```
function Glossary(props) {  
  return (  
    <dl>  
      {props.items.map(item => (  
        // Without the `key`, React will fire a key warning  
        <React.Fragment key={item.id}>  
          <dt>{item.term}</dt>  
          <dd>{item.description}</dd>  
        </React.Fragment>  
      ))}  
    </dl>  
  );  
}
```

React Routing

- Why routing in React?
 - User should have the feeling that the react app is like any other website, with different urls etc.
 - Different content should be visible through different url's
 - Some content should only be shown after a login
 - Content should be shown through url parameter
- We need to install an extra package for this, as this is not delivered with the default react installation

npm i react-router-dom

- Most important/used Components/Hooks from the **react-router-dom** package:
- **BrowserRouter**
 - The best way to use this component is if it encloses the `<App />` component within the `index.js` file.
 - Through this method all other components within the app are able to use the functions the package has to offer.
 - Stores the current location in the browser's addressbar using clean URLs and navigates using the browser's built-in history stack
- **Link / NavLink**
 - To be able to link to an other url within a react app you can't use the HTML a-tag you have to use the Link or NavLink component.
 - In most cases the Link/NavLink components are used in a Navigation component, but you can use these components in any other component as you like.
 - Special about NavLink: It knows wether or not it is „active“.

Useful when building menu like Tabs, Burger-Menu etc.

- **Navigate**
 - Changes the current location when the component is rendered
 - Intern it uses the `useNavigate()` hook
- **Routes**
 - Need to enclose the `Route` component as it looks through all children if the url matches with the url in the browser. If there was a match all other `Route` components are ignored
- **Route**
 - With this component the different urls are defined the app should have
 - You are able to define which component should be shown through which url
- **useNavigate Hook**
 - With this hook you are able to change the location of the user programmatically
- To use this package you need to install it like this: **`npm install react-router-dom`**

BrowserRouter

```
1  import * as React from "react";
2  import * as ReactDOM from "react-dom";
3  import { BrowserRouter } from "react-router-dom";
4
5  ReactDOM.render(
6    <BrowserRouter>
7      {/* The rest of your app goes here */}
8    </BrowserRouter>,
9    root
10  );
```

Link

```
1  import * as React from "react";
2  import { Link } from "react-router-dom";
3
4  function UsersIndexPage({ users }) {
5    return (
6      <div>
7        <h1>Users</h1>
8        <ul>
9          {users.map((user) => (
10            <li key={user.id}>
11              <Link to={user.id}>{user.name}</Link>
12            </li>
13          ))}
14        </ul>
15      </div>
16    );
17  }
```

Route/Routes

```
1  <Routes>
2    <Route path="/" element={<Dashboard />}>
3      <Route
4        path="messages"
5        element={<DashboardMessages />}
6      />
7    <Route path="tasks" element={<DashboardTasks />} />
8  </Route>
9  <Route path="about" element={<AboutPage />} />
10 </Routes>
```

Navigate

```
1  import * as React from "react";
2  import { Navigate } from "react-router-dom";
3
4  class LoginForm extends React.Component {
5    state = { user: null, error: null };
6
7    async handleSubmit(event) {
8      event.preventDefault();
9      try {
10        let user = await login(event.target);
11        this.setState({ user });
12      } catch (error) {
13        this.setState({ error });
14      }
15    }
16
17    render() {
18      let { user, error } = this.state;
19      return (
20        <div>
21          {error && <p>{error.message}</p>}
22          {user && (
23            <Navigate to="/dashboard" replace={true} />
24          )}
25          <form
26            onSubmit={(event) => this.handleSubmit(event)}
27          >
28            <input type="text" name="username" />
29            <input type="password" name="password" />
30          </form>
31        </div>
32      );
33    }
34  }
```

useNavigate

```
1  import { useNavigate } from "react-router-dom";
2
3  function SignupForm() {
4    let navigate = useNavigate();
5
6    async function handleSubmit(event) {
7      event.preventDefault();
8      await submitForm(event.target);
9      navigate("../success", { replace: true });
10   }
11
12   return <form onSubmit={handleSubmit}>{/* ... */}</form>;
13 }
```


index.js from the To-Do app

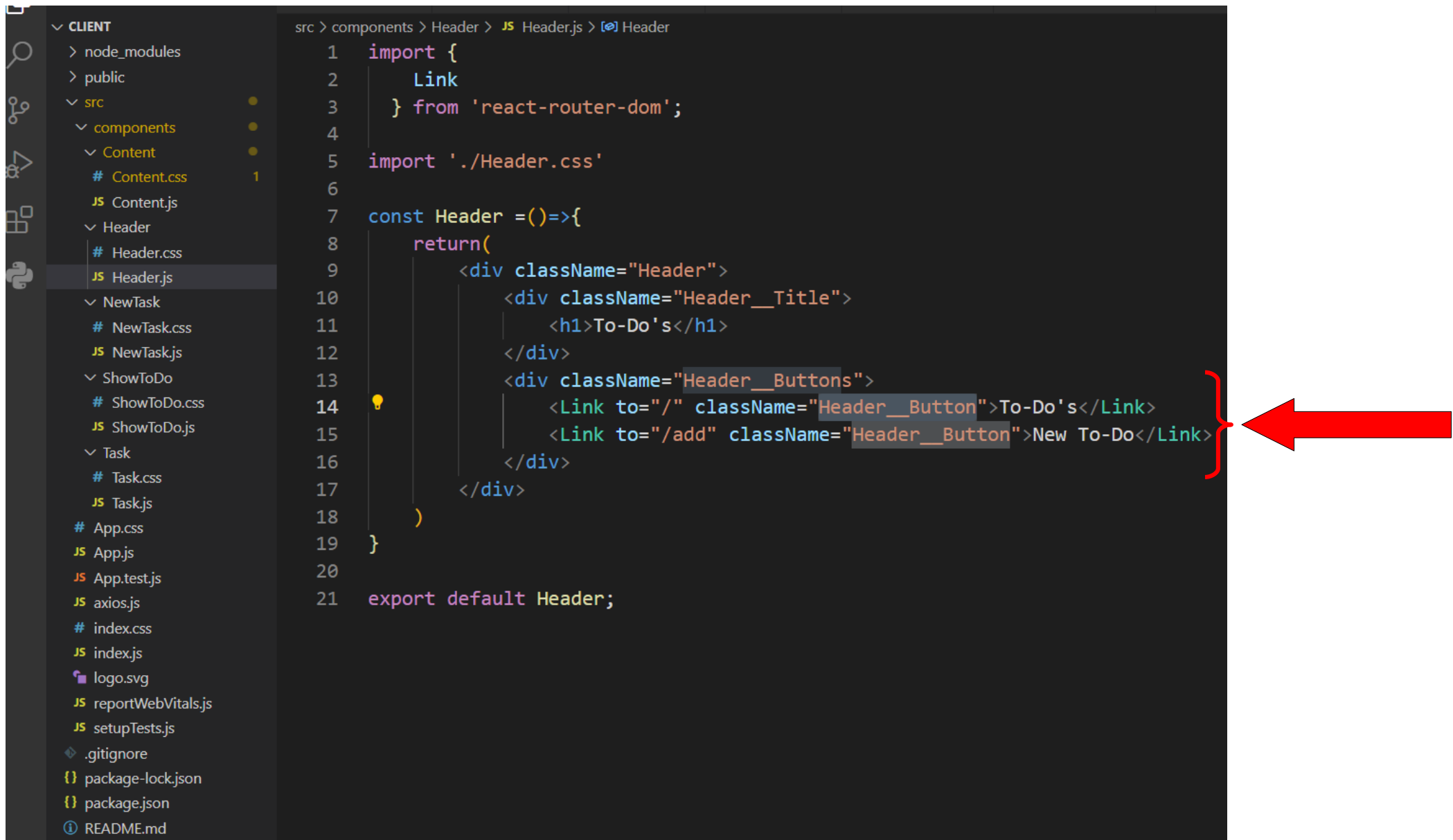
```
src > JS index.js > ...
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import {
4    BrowserRouter
5  } from 'react-router-dom';
6  import './index.css';
7  import App from './App';
8  import reportWebVitals from './reportWebVitals';
9
10 const root = ReactDOM.createRoot(
11   | | | | | | | document.getElementById('root'));
12 root.render(
13   <BrowserRouter>
14     <React.StrictMode>
15       <App />
16     </React.StrictMode>
17   </BrowserRouter>
18 );
19
20
21 // If you want to start measuring performance in your app, pass a function
22 // to log results (for example: reportWebVitals(console.log))
23 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
24 reportWebVitals();
25
```

App.js from the To-Do app

```
JS Content.js  # Content.css 1  JS index.js  JS App.js  # Header.css  JS H

src > JS App.js > ...
 1  import {Routes,Route} from 'react-router-dom';
 2  import './App.css';
 3  import Header from './components/Header/Header';
 4  import ShowToDo from './components/ShowToDo/ShowToDo';
 5  import NewTask from './components/NewTask/NewTask';
 6  function App() {
 7    return (
 8      <div className="App">
 9        <Header />
10        <Routes>
11          <Route exact path="/" element={<ShowToDo />} />
12          <Route exact path="/add" element={<NewTask />} />
13        </Routes>
14      </div>
15    );
16  }
17
18  export default App;
19
```

Header.js from the To-Do app



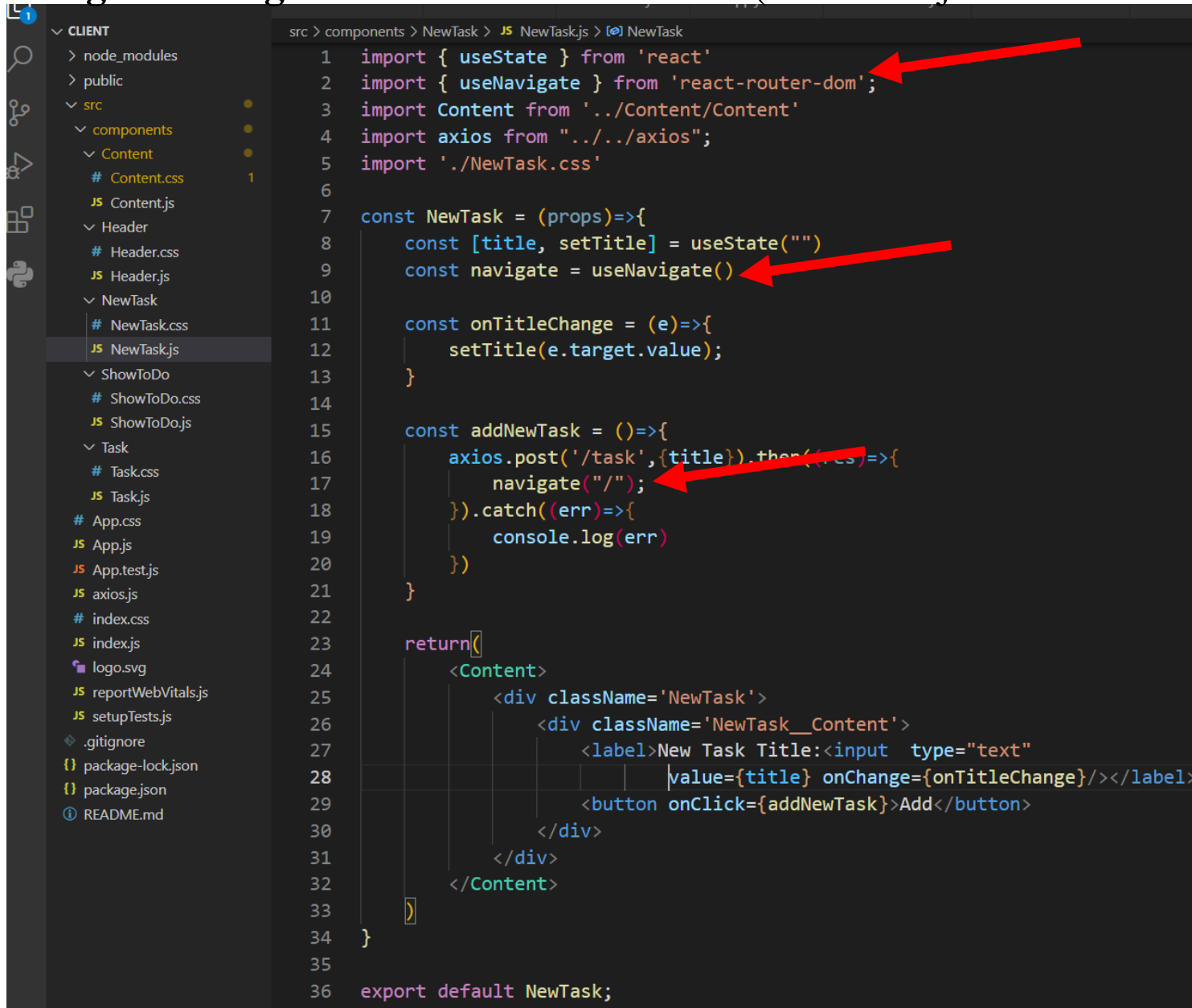
```
src > components > Header > JS Header.js > Header
1  import {
2    Link
3  } from 'react-router-dom';
4
5  import './Header.css'
6
7  const Header = () => {
8    return (
9      <div className="Header">
10        <div className="Header__Title">
11          <h1>To-Do's</h1>
12        </div>
13        <div className="Header__Buttons">
14          <Link to="/" className="Header__Button">To-Do's</Link>
15          <Link to="/add" className="Header__Button">New To-Do</Link>
16        </div>
17      </div>
18    )
19  }
20
21  export default Header;
```

To-Do's

To-Do's

New To-Do

Using useNavigate Hook to redirect user (NewTask.js from the To-Do App)



```

1  import { useState } from 'react'
2  import { useNavigate } from 'react-router-dom';
3  import Content from '../Content/Content'
4  import axios from "../../axios";
5  import './NewTask.css'
6
7  const NewTask = (props)=>{
8      const [title, setTitle] = useState("")
9      const navigate = useNavigate()
10
11      const onTitleChange = (e)=>{
12          setTitle(e.target.value);
13      }
14
15      const addNewTask = ()=>{
16          axios.post('/task', {title}).then((res)=>{
17              navigate("/");
18          }).catch((err)=>{
19              console.log(err)
20          })
21      }
22
23      return(
24          <Content>
25              <div className='NewTask'>
26                  <div className='NewTask__Content'>
27                      <label>New Task Title:<input type="text"
28                          value={title} onChange={onTitleChange}/></label>
29                      <button onClick={addNewTask}>Add</button>
30                  </div>
31              </div>
32          </Content>
33      )
34  }
35
36  export default NewTask;

```

Ajax/HTTP Request React

- If you are creating a react app it is a **SPA (Single Page Application)**, because of this the data do not get **updated** through a **reload** of the page like in php.
- The data a SPA consumes are often from an api-server.
To get the data you need to request them from the server.
- These requests are executed through **ajax /http-requests**
- There are multiple ways you could make these requests, most common is to use the node.js package axios.
- **npm install axios**

- REST request in React:
 - There are multiple ways to make http-request in react:
 - 1) Make a http-request in the useEffect Hook
 - Good if you want to load data when the component is rendered
 - 2) Make a http-request through an event (e.g. user clicks on a button)
 - If you want to update, create new data on server or something should be executed on the server through user interaction

- How are axios request made?
 - First you have to import the axios package
 - Use axios object to get access to GET, POST, PUT and DELETE request

```
const axios = require('axios').default;  
  
// Make a request for a user with a given ID  
axios.get('/user?ID=12345')  
  .then(function (response) {  
    // handle success  
    console.log(response);  
  })  
  .catch(function (error) {  
    // handle error  
    console.log(error);  
  })  
  .then(function () {  
    // always executed  
  });
```

First parameter of a axios-request is always the url. It is also possible to add query parameters in the url as shown in the picture

If the request was a success then the response from the server is in the **response** object from the response and the **.then** part will be executed

If the request was a failure then there also data in the **response** object, but the **.catch** code is executed

```
axios.post('/user', {  
  firstName: 'Fred',  
  lastName: 'Flintstone'  
})  
.then(function (response) {  
  console.log(response);  
})  
.catch(function (error) {  
  console.log(error);  
});
```

For POST, PUT and DELETE the second parameter is a **object** or **data** (in general that you want to **send** to the server) in the **body** of the http-request like shown in the picture

- If the http-request is made to same host where the web-app is hosted you don't need to specify the url. As we have seen in the example before.

```
axios.get('/task').then((res)=>{  
  //some code  
}).catch((err)=>{  
  console.log(err)  
})
```

- If you want to make a request to a different host you have to specify the whole url.

```
axios.get('http://www.localhost:3005/task').then((res)=>{  
  //some code  
}).catch((err)=>{  
  console.log(err)  
})
```

- In axios we also have the option to specify a baseUrl. With this you don't have to specify the whole url if the request is made to different url. This prevents copy&past mistakes while creating http-requests and prevents the overhead in changing all http-request, should the url change. (example follows later)

Axios Request with options

```
export const getTodos = () => {  
  return dispatch => {  
    axios.get('/todos', {withCredential:true}).then((response) => {  
      dispatch(storeTodos(response.data));  
    }).catch((error) => {  
      console.log(error);  
    })  
  }  
}
```

If you want to set some options for your http-request these can be made through an object, which is the second parameter in the GET-Request and third parameter in POST-, PUT- or DELETE-request

In this example we want to send a cookie with our http-request, for this we set the option

{withCredential:true}. In this way the cookie is sent with the http-request

NewTask.js

```
4
5  const addNewTask = ()=>{
6    axios.post('/task',{title}).then((res)=>{
7      navigate("/");
8    }).catch((err)=>{
9      console.log(err)
10    })
11  }
```

Here a POST request to the server is made here. We send a JSON object to server which only has the title.

e.g.:

```
{
  „title“:“Task1“
}
```

ShowToDo.js

```

1  import React from "react";
2  import Content from "../Content/Content";
3  import Task from "../Task/Task";
4  import axios from "../..//axios";
5  import './ShowToDo.css'
6  import { useState,useEffect } from "react";
7  const ShowToDo = ()=>{
8      const [tasks, setTasks] = useState([])
9
10     useEffect(()=>{
11         if(tasks.length === 0){
12             axios.get('/tasks').then((res)=>{
13                 setTasks(res.data)
14             }).catch((err)=>{
15                 console.log(err)
16             })
17         }
18     },[])
19 }

```

GET request to the api server in the useEffect() hook and the hook is only executed once, which can be seen on the square brackets at the end of the hook.

If we put a useState variable like **tasks** within the brackets, the **useEffect** Hook will only be executed if we **change** the task variable through **setTasks**

To-Do's

To-Do's
New To-Do

Open	Completed
	<div> <div>◀</div> <div>Test</div> <div>🗑</div> </div>

ShowToDo Component

ShowToDo.js

```
const moveTask = (taskId) => {
  const taskIndex = tasks.findIndex((v) => {return v.id === taskId});

  const task = {...tasks[taskIndex]};
  task.completed = !task.completed;

  const tasksCopy = [...tasks];
  tasksCopy[taskIndex] = task;

  axios.put('/task', {...task}).then((res) => {
    setTasks(tasksCopy)
  }).catch((err) => {
    console.log(err)
  })
}

const deleteTask = (taskId) => {
  const taskIndex = tasks.findIndex((v) => {return v.id === taskId});
  const task = {...tasks[taskIndex]}
  const tasksCopy = [...tasks];
  tasksCopy.splice(taskIndex, 1);

  axios.delete(`/task/${taskId}`).then((res) => {
    setTasks(tasksCopy)
  }).catch((err) => {
    console.log(err)
  })
}
```

In this picture you can see how delete and put request to the api-server are made and what changes we were made to these functions

Important changes to server for the delete route

```
app.delete('/task/:id', (req, res) => {  
  const id = req.params.id  
  try{ ...  
  }catch(error){ ...  
  }  
  /*let searchedtaskIndex = data.tasks.findIndex((v) => v.id == id) ...  
})
```

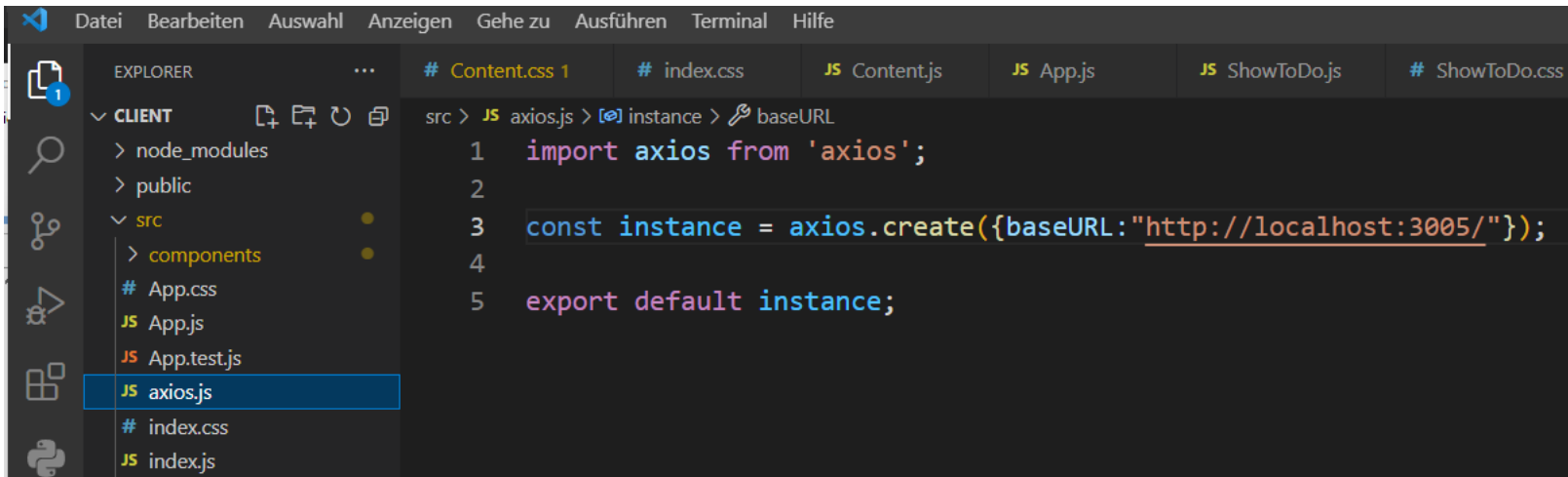

How a GET request looks like after the server send a response:

[App.js:14](#)

```
▼ {data: Array(100), status: 200, statusText: "OK", headers: {...}, config: {...}, ...} ⓘ  
  ▶ config: {url: "http://jsonplaceholder.typicode.com/posts", method: "get", headers: {...}, t...  
  ▶ data: (100) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}  
  ▶ headers: {cache-control: "max-age=43200", content-type: "application/json; charset=utf-8"...  
  ▶ request: XMLHttpRequest {readyState: 4, timeout: 0, withCredentials: false, upload: XMLHt...  
    status: 200  
    statusText: "OK"  
  ▶ __proto__: Object
```

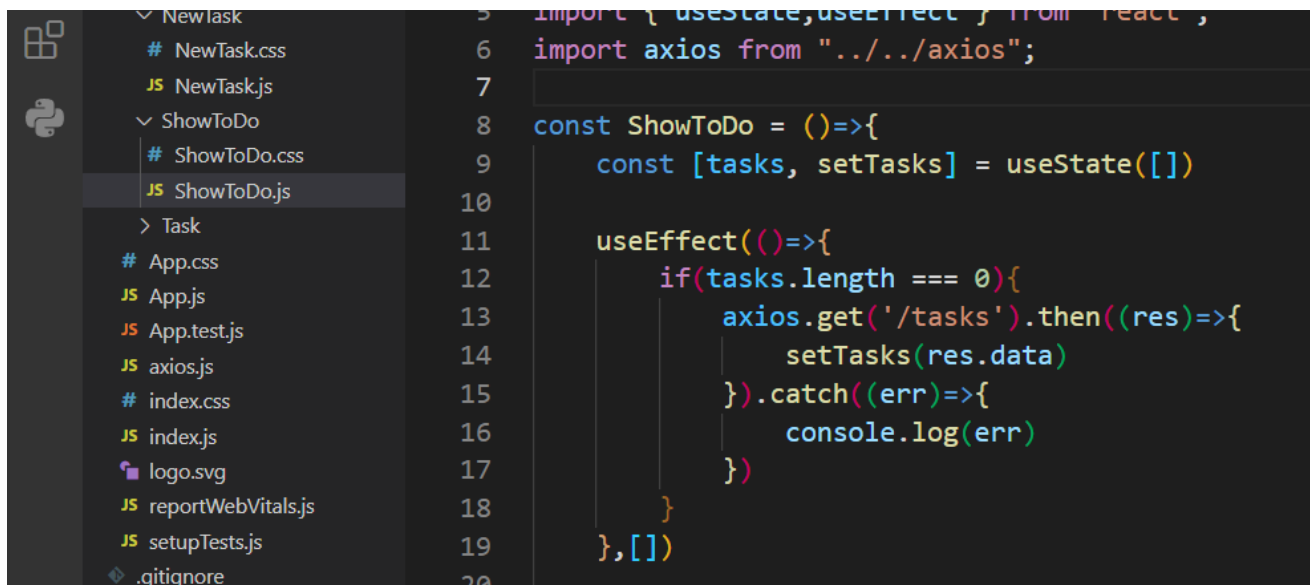
- Optimize http-request through a changed axios object:
 - As an app grows you will have multiple http-requests in your app. Because of this it is easy to make mistakes while copying the url.
 - To reduce this risk it is possible in axios to set a base-url which will be used for all http-requesta. Like this you only need to set the specific routes. It is also easy to change the base-url should this change in the future
 - How you can do this is shown in the picture in the next slide.
 - 1) You create a new file in the **src** directory named e.g. **axios or axios-url.js**
 - 2) Import axios package
 - 3) Create an axios instance and set the baseURL
 - 4) Export the created instance
 - 5) Everytime you want to make a http-request in a component you import this axios instance instead the axios object from the axios package

Creating an **axios.js** file within the **src** directory and create an **axios instance** and we set the **baseUrl**, which will be used for ajax-requests.



This screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows the project structure with the 'src' directory expanded, and 'axios.js' is selected. The main editor area shows the content of 'axios.js' with the following code:

```
src > JS axios.js > [?] instance > baseUrl
1  import axios from 'axios';
2
3  const instance = axios.create({baseUrl:"http://localhost:3005/"});
4
5  export default instance;
```

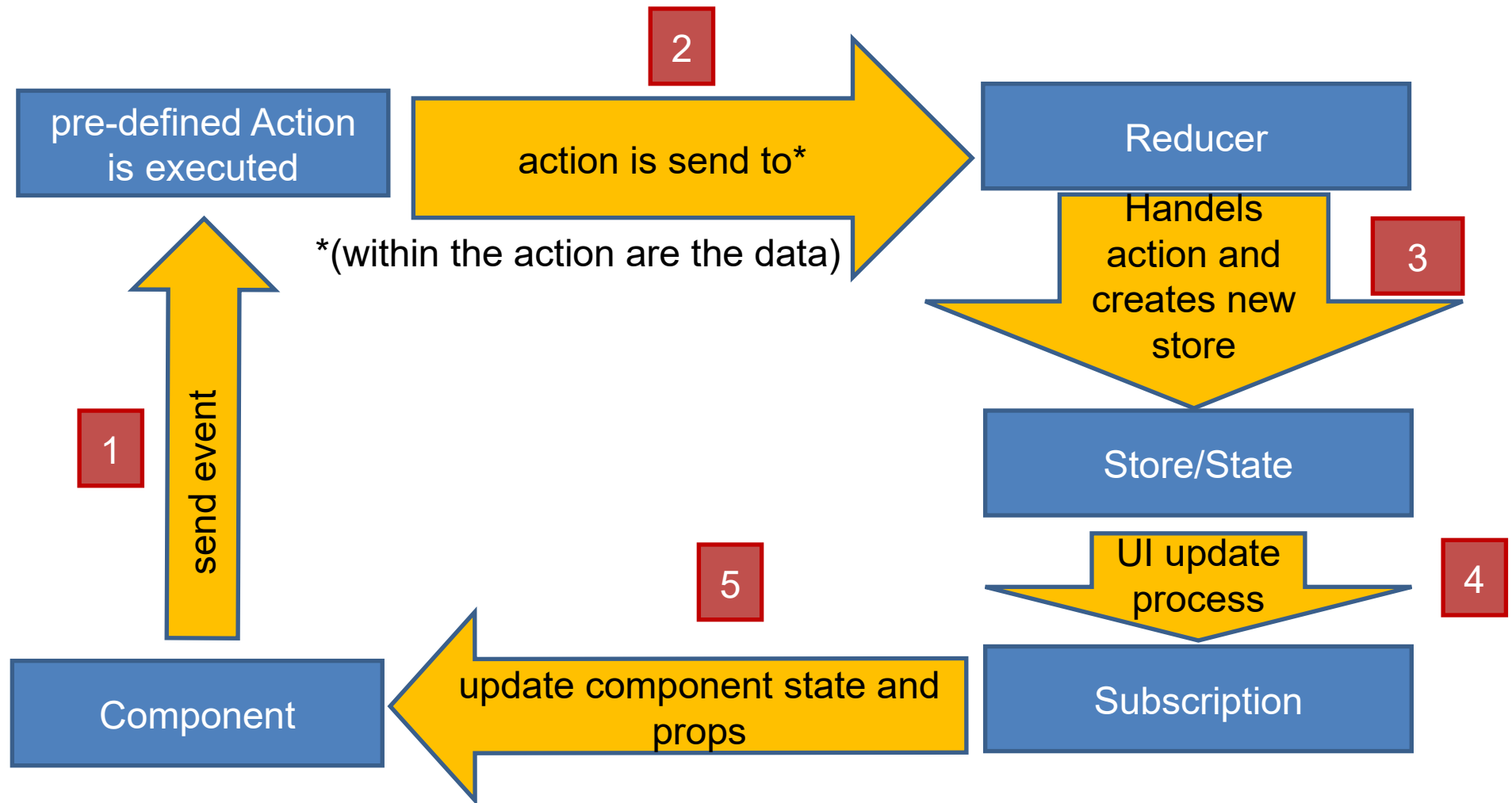


This screenshot shows the Visual Studio Code interface with 'ShowToDo.js' selected in the Explorer. The main editor area shows the code for 'ShowToDo.js' which uses the axios instance:

```
5  import { useState, useEffect } from 'react';
6  import axios from "../../axios";
7
8  const ShowToDo = ()=>{
9      const [tasks, setTasks] = useState([])
10
11      useEffect(()=>{
12          if(tasks.length === 0){
13              axios.get('/tasks').then((res)=>{
14                  setTasks(res.data)
15              }).catch((err)=>{
16                  console.log(err)
17              })
18          }
19      },[])
20  }
```

React Redux

- Redux is a state management library built in and for JavaScript
- With Redux it is possible to save and access data through whole app in a central area
- It is possible to define actions (functions) which your able to access in the whole app
- You are able to use this library in react
- In react there usually exists component which use useState to create local state object and pass data to children. You also could use the useContext hook but to save data at a central area. But useContext hook does not provide the functionality and possibilities in data management as redux.
- With redux your are able to control all the data and actions in a central area and you are also able to use mutliple data storages for a better management of data
- Redux is easy to add data or action (functions) once setup.
- Redux is also one of the most used state manegement library for react.
- Redux install:
 - **npm install redux**
 - **npm install @reduxjs/toolkit react-redux**



```

1. import { createAction, createReducer, createAsyncThunk } from "@reduxjs/toolkit";
   import axios from '../axios'

2. export const changeTaskState = createAction('todos/changeStateToDo');
   export const deleteTaskId = createAction('todo/deleteToDoId');

3. export const addTask = createAsyncThunk('todos/addToDo', async (task) => {
   |   return await axios.post('/task', task)
   | });
   export const loadTodos = createAsyncThunk('todo/loadTodos', async () => {
   |   const response = await axios.get('/tasks');
   |   return response.data;
   | });

4. const initialState = {todos: []}

5. const todoReducer = createReducer(initialState, (builder) => { ...
   | });
   export default todoReducer;
  
```

1) You need to define some actions which your app should have

1. Create **reducer** directory in the **src** directory and create a new JavaScript file **reducer.js** within this directory and import the **createAction**, **createReducer**, **CreateAsyncThunk** from the **@reduxjs/toolkit** library

2. With the function **createAction()** we define synchronous actions (functions), which you are able to access in the components. **Do not forget to export these actions**

3. With **createAsyncThunk** you are able to create asynchronous action which is important, because **redux** is synchronous and if asynchronous code is executed bad things will happen to your central state.

4. **initialState** is the initial instance of your reducer and the reducer is initialized with this object. Here you can predefine some attributes etc.

5. With **createReducer** you create a reducer, which is the hearth of **redux** and manages the data for you. It is the central state of your app⁴⁷

```
const todoReducer = createReducer(initialState, (builder)=>{
  builder.addCase(changeTaskState, (state, action)=>{
    const taskIndex = state.todos.findIndex((v)=>{return v.id === action.payload.taskId});

    const task = {...state.todos[taskIndex]};
    task.completed = !task.completed;

    const todosCopy = [...state.todos];
    todosCopy[taskIndex] = task;
    return {
      ...state,
      todosCopy
    }
  })
  .addCase(deleteTask, (state, action)=>{
    const taskIndex = state.todos.findIndex((v)=>{return v.id === action.payload.taskId});
    const todosCopy = [...state.todos];
    todosCopy.splice(taskIndex, 1);

    return {
      ...state,
      todosCopy
    }
  })
  .addCase(loadTodos.fulfilled, (state, action)=>{
    return {
      ...state,
      todos: action.payload
    }
  })
  .addCase(addTask, (state, action)=>{
    const todosCopy = [...state.todos];
    todosCopy.push(action.payload);
    return {
      ...state,
      todosCopy
    }
  })
});

export default todoReducer;
```

2) After defining your actions you call **createReducer** and the first parameter is your **initialState** object you defined before. The second parameter is a function in which all the **data management** will happen. This function will have parameter named **builder**. Through this builder object you are able to access **redux management functions**

3) With **builder.addCase** you can add **actions** to the **reducer**, which you defined before. If these actions are **called** within a **component**, the actions **within the reducer** are **executed**. E.g. the **completed** action will be executed if the action „**todo/completed**“ (last slide) is called somewhere in your component. The first parameter is the **action object** you created and the second parameter is a **function** which will be **executed** when the **action was called**. This function has two parameters: The first is the **current reducer state** and as the second parameter the **action** which was called. Within the **action object** you are able to access the **data** which the current reducer should be **updated** with via **action.payload**

4) If you want to add **asynchronous** actions to the reducer you need to add **fulfilled** to the action declaration. With this the reducer **waits** till the action is **completed** before the data are **updated**.

5) At the end you need to export the reducer


```

1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import {BrowserRouter} from 'react-router-dom';
4  import './index.css';
5  import App from './App';
6  import reportWebVitals from './reportWebVitals';
7
8  import { configureStore } from '@reduxjs/toolkit';
9  import { Provider } from 'react-redux';
10 import rootReducer from './reducer/reducer';
11
12 const store = configureStore({reducer:rootReducer});
13
14 const root = ReactDOM.createRoot(
15   document.getElementById('root'));
16 root.render(
17   <Provider store={store}>
18     <BrowserRouter>
19       <React.StrictMode>
20         <App />
21       </React.StrictMode>
22     </BrowserRouter>
23   </Provider>
24 );
25
26 // If you want to start measuring performance in your app, pass
27 // to log results (for example: reportWebVitals(console.log))
28 // or send to an analytics endpoint. Learn more: https://bit
29 reportWebVitals();
30

```

Within index.js:

1. Import **configureStore** and **Provider**

2. Import your **Reducer** and pass it to the **configureStore** function

3. **configureStore** is responsible that the data-management, which we created in the last slide, is accessible from everywhere

4. The **Provider** component encloses the **App** component

5. In the last step you need to pass the store constant you created, in the second step, to the Provider component

```

JS NewTask.js • JS index.js JS Header.js JS reducer.js # Header.css JS Content.js # Conter
src > components > NewTask > JS NewTask.js > [🔍] NewTask > [🔍] addNewTask
1  import { useState } from 'react'
2  import { useNavigate } from 'react-router-dom';
3  import Content from '../Content/Content'
4  import axios from "../../axios";
5  import './NewTask.css'
6
7  import { useDispatch } from 'react-redux';
8  import {addTask} from '../../reducer/reducer'
9
10 const NewTask = (props)=>{
11   const [title, setTitle] = useState("");
12   const navigate = useNavigate();
13   const dispatch = useDispatch();
14
15   const onTitleChange = (e)=>{
16     setTitle(e.target.value);
17   }
18
19   const addNewTask = async ()=>{
20     await dispatch(addTask({title})).unwrap();
21     navigate("/");
22   }
23   /* ...
24
25   return(
26     <Content>
27       <div className='NewTask'>
28         <div className='NewTask__Content'>
29           <label>New Task Title:<input type="text"
30             value={title} onChange={onTitleChange}/></label>
31           <button onClick={addNewTask}>Add</button>
32         </div>
33       </div>
34     </Content>
35   )
36 }
37
38 export default NewTask;

```

- Import **useDispatch** hook
- Import the **actions** you defined before

1. With **useDispatch** you can call a action which you imported and was defined in the reducer file

2. The parameter you pass to a **normal** action are later accessible with the payload object. If you want to pass multiple data you need to use an object or array. In our example the action is a **async action**, so the parameter is a normal parameter, as we could see in **addTask** action

```

JS ShowToDo.js x JS NewTask.js JS index.js JS Header.js JS reducer.js # Header.css JS Content.js
src > components > ShowToDo > JS ShowToDo.js > ShowToDo > useEffect() callback
1 import React from "react";
2 import Content from "../Content/Content";
3 import axios from "../../axios";
4 import Task from "../Task/Task";
5 import './ShowToDo.css'
6 import { useState,useEffect } from "react";
7 import { useDispatch,useSelector } from 'react-redux';
8 import {loadTodos,deleteTaskId,changeTaskState} from '../reducer/reducer'
9
10 const ShowToDo = ()=>{
11   //const [tasks, setTasks] = useState([])
12   const tasks = useSelector((state)=>{return state.todos});
13   const dispatch = useDispatch();
14
15   useEffect(()=>{
16     dispatch(loadTodos());
17   },[])
18
19   const moveTask = (taskId)=>{
20     const taskIndex = tasks.findIndex((v)=>{return v.id === taskId});
21     const task = {...tasks[taskIndex]};
22
23     axios.put('/task',{...task}).then((res)=>{
24       //setTasks({taskId})
25       dispatch(changeTaskState({taskId}))
26     }).catch((err)=>{
27       console.log(err)
28     })
29   }
30
31   const deleteTask = (taskId) =>{
32     axios.delete('/task',{data:{id:taskId}}).then((res)=>{
33       //setTasks({taskId})
34       dispatch(deleteTaskId({taskId}))
35     }).catch((err)=>{
36       console.log(err)
37     })
38   }
39
40   /* ...
41
42   */
43 }
44
45
46
47
48
49
50
51
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

```

- Import **useDispatch** and **useSelector** hook
- Import the **actions** you defined before

1. With **useSelector** hook the data from the reducer is accessible within this component. As a parameter its expects a function which returns a specific **state-object (reducer)**

2. With **useDispatch** you can call a action which you imported and was defined in the reducer file

3. The parameter you pass to a **normal** action are later accessible with the payload object. If you want to pass multiple data you need to use an object or array. In our example the action is a **async action**, so the parameter is a normal parameter, as we could see in addTask action

- Exercise 6 (Change the react-App from exercise 5)
 - 1) You should be able to add new tasks
 - 2) Use routing to only display the component, which shows all tasks or the component with which you are able to create a new task
 - 3) As shown in the slides, create a instance of axios and set the baseURL and use this instance for all api-requests
 - 4) The React-App uses the API-Server from exercise 4 for requests
- Special Exercise 7 (Change the react-App from exercise 6)
 - 1) Change the App so it uses redux
 - 2) API-Request should be done over react-redux
 - 3) Data should be saved in the redux store
 - 4) Changes in the task should be done in the redux store

Hint: All the code you need for this exercise is already in the slides and needs to be assembled correctly