

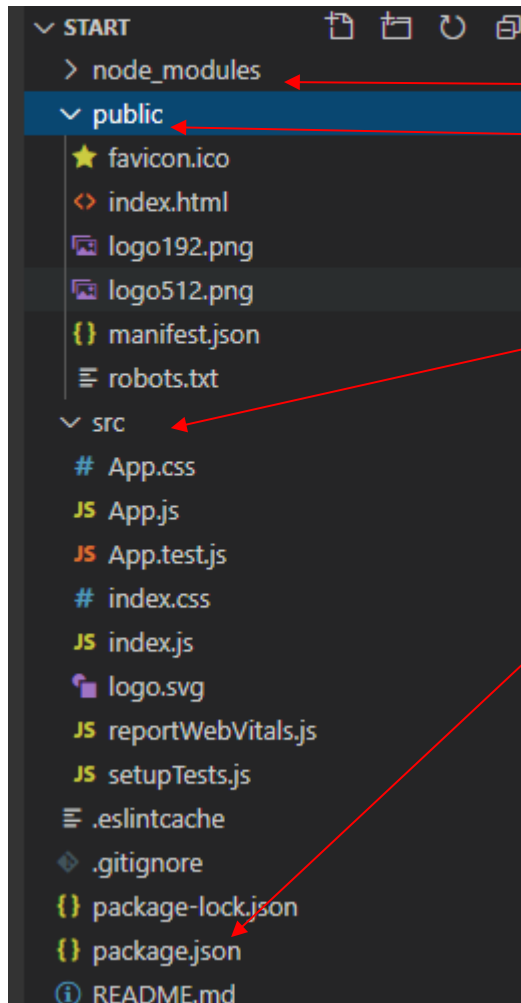
Web Programming JavaScript – node.js - react

**Lecture 5-2:
React Intro**
23.11.

**Stefan Noll
Manuel Fehrenbach
Winter Semester 22/23**

- How to create Node.js React-App:
 - Open the directory where the other Node.js projects are located
 - Open the Terminal of your operating system, it should point to the directory with all the Node.js projects.
- Steps how to create a React Project:
 - 1) In the terminal execute: „npx create-react-app [project name]“
 - 2) After the project was created open the project in visual studio code
 - 3) Within visual studio open in the upper menu „terminal“
 - 4) In the terminal execute: „npm start“

- First look into a React-App:



- All installed packages are located here

- Root directory, which is send to the client from the webserver

- Within there exists the *only .html* file (index.html) which is needed

- Within the *src directory* are all our *JavaScript files and directory located*, which we need for our react app.

- Within package.json are all installed packages listed

- First look into a React-App:
- Title of the website, the *only* thing you need to change within the *index.html* file
- Root div-Element in which react-code is importet

```

> public > index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6     <meta name="viewport" content="width=device-width, initial-scale=1" />
7     <meta name="theme-color" content="#000000" />
8     <meta
9       name="description"
10      content="Web site created using create-react-app"
11    />
12    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13    <!--
14      manifest.json provides metadata used when your web app is installed on a
15      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
16    -->
17    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
18    <!--
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="root"></div>
32    <!--
33      This HTML file is a template.
34      If you open it directly in the browser, you will see an empty page.
35
36      You can add webfonts, meta tags, or analytics to this file.
37      The build step will place the bundled scripts into the <body> tag.
38
39      To begin the development, run `npm start` or `yarn start`.
40      To create a production bundle, use `npm run build` or `yarn build`.
41    -->
42  </body>
43 </html>

```

Index.js

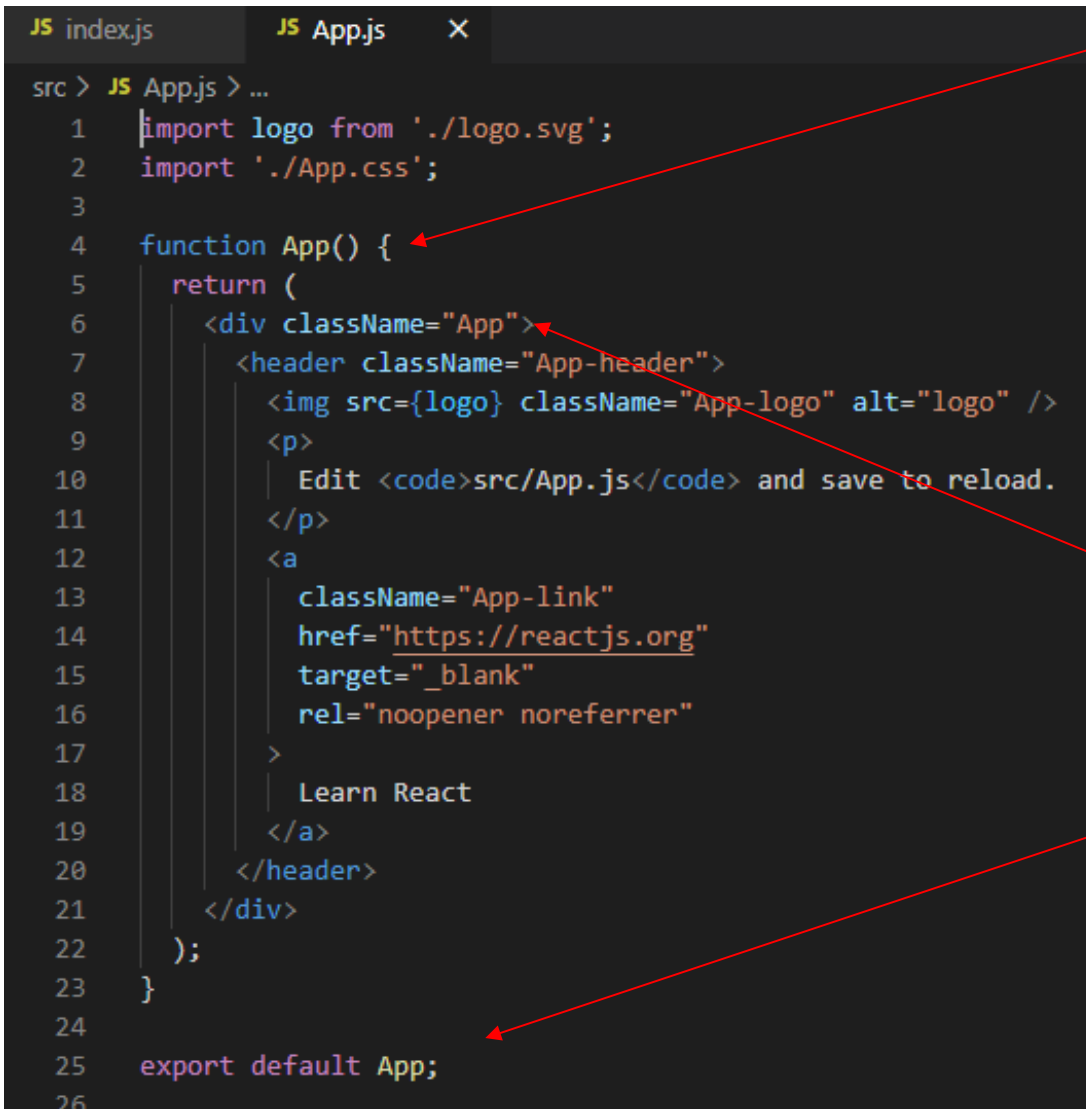
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
```

```
const root = ReactDOM.createRoot(
  document.getElementById('root')
);

root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

- First look into a React-App:
- Different packages or Object/Functions which are used within this file
- To be able to see our website we need the **ReactDOM** object and call the **createRoot()** function, which returns us a **root** object. The function **createRoot()** *expects one parameter* which tells our react app, *where* it should place our code (default is div-tag with the id root)
- From the **root object** we call the function **render()**, which expect *exactly one outer jsx element*
- In the beginning there is nothing much you need to change in the index.js file

- First look into a React-App:



```

JS index.js  JS App.js  X
src > JS App.js > ...
1  import logo from './logo.svg';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className="App">
7        <header className="App-header">
8          <img src={logo} className="App-logo" alt="logo" />
9          <p>
10             Edit <code>src/App.js</code> and save to reload.
11          </p>
12          <a
13            className="App-link"
14            href="https://reactjs.org"
15            target="_blank"
16            rel="noopener noreferrer"
17          >
18            Learn React
19          </a>
20        </header>
21      </div>
22    );
23  }
24
25  export default App;
26

```

- Within App.js we see our first component. All react apps consist of *different components*. What components are in detail will be later explained.
- Here we have *Functional Component* App, which we export at the end of the file (Components are in general only functions)
- Here we see a JSX-Tag (<div>) which will be automatically translated to HTML
- Through the export we are able to use the Component in index.js or other components.

JSX

- JSX in detail:
 - Without JSX we would need to write React Elemente as shown in the picture:
 - We would need to import **React** object from the **react package** and call **createElement()**
 - First parameter: Is the *HTML-Element* which should be created
 - Second parameter: *HTML-Attributes*
 - Third parameter: Are the *Children* the element should have

```
function Menu(props){
  return React.createElement("h1",{id:"header-1"},"Titel");
}

function Liste(props){
  return React.createElement("ul",{id:"header-1"},
    React.createElement("li",null,"Pizza"),
    React.createElement("li",null,"Burger")
  );
}
```



```
<h1 id="header-1">Titel</h1>

<ul>
  <li>Pizza</li>
  <li>Burger</li>
</ul>
```


- JSX in detail:
 - JSX is a combination of JavaScript and XML
 - Usage as HTML-Tags e.g: `<h1>`, `<div>`,...
 - JSX is ***not*** HTML => JSX is just another and easier way to create React elements.
 - React Element: `React.createElement(Menu, {list:[...]})`;
 - JSX: `<Menu list={[...]}>`

- JSX in detail:
 - In JavaScript *class* is a keyword, because of this css class-attribute is called *className* in JSX.
 - As *everything* between *opening and closing JSX-Element* is seen as a *string*, it is not possible to use JavaScript without using `{}`.
 - e.g.:


```
let title = „Title“;
<h1>{title}</h1>
<input type=„checkbox“ defaultChecked={false}/>
```
 - Between `{}` it is also possible to calculations or execute other JavaScript code:


```
<h1>{„Large“ + title}</h1>
<h1>{5+5}</h1>
```

React Components

- What are Components?
 - Code snippets, which can be used independently
 - Components are *JavaScript functions*, just that components work *isolated* from each other and return *JSX (HTML)*
 - There are two types of components **class components** and **functional components**, but only the later should be used.
 - Functional components are *defined* and *structured* as normal JavaScript functions.
 - A functional component has a parameter *props*, with which it is possible to access *properties* as well as *child-elements*

- Functional Components?
 - Components can have **attributes** which can be **accessed** through the **props** parameter.
The **value** of the attributes can **any** type which exists in JavaScript (JSON, function, objects,...)
 - Between the **opening** and **closing** tag there can also be other **component** or **JSX-elements** to which we get access to through **props.children**

```
src > Person > JS Person.js > [?] person
1  import React from "react";
2
3  const person = (props) => {
4    return (
5      <div>
6        <p>Name: {props.name}</p>
7        <p>Age: {props.age}</p>
8        <p>Beruf: {props.children}</p>
9      </div>
10    );
11  }
12
13  export default person;
```

```

v import logo from './logo.svg';
import './App.css';
import React, {Component} from "react";

import Person from "../Person/Person";

v class App extends Component{
v   render(){
v     return (
v       <div className="App">
v         <Person name="Hans" age="84" >Rentner</Person>
v         <Person name="Peter" age="60" />
v       </div>
v     );
v   }
}
export default App;
```

Name: Hans
Age: 84
Beruf: Rentner
Name: Peter
Age: 60
Beruf:

- Functional Components?
 - Within the **return** statement there is **always** only **one** root element, in which the other components are integrated or other JSX-elements are used as shown in the left picture (**div element = root element**).
 - The App component is later **imported** and **used** in the index.js file as shown in the right picture.
 - One important aspect is, that functional components need to start with an **upper letter**.

```
JS App.js  X  # Task.css  JS Task.js
src > JS App.js > ...
 1 > import { ...
14 function App() {
15   return (
16     <div className="App">
17       <Header />
18       <ShowToDo />
19     </div>
20   );
21 }
22
23 export default App;
```

App.js from To-Do Exercise

```
const root = ReactDOM.createRoot(
  document.getElementById('root'));
root.render(
  <BrowserRouter>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </BrowserRouter>
);
```

Index.js from To-Do Exercise

- Stateless vs Stateful Component
 - There is another categorization between components: **stateless** and **stateful** components
 - Every component which uses the **useState()** Hook is a **stateful** component.
In this component a specific state is saved. These components are sometimes called **smart components**
 - Every Component which does **not** use that **useState()** Hook is a **stateless** component or also known as **dump components** or **presentation component**
 - Usually it's better to have **more dump components**, as these components can be used in **other project** and only need data via children or attributes

- Steps creating functional components

- 1) Within the **src** directory create a new directory **components** if not existing already and create **within** the components directory a new directory with the **name** of the new component
- 2) In the new component directory create two files (JavaScript and CSS) with the name of the component (First character is an uppercase)
- 3) At the start of the JavaScript file, the **React** object from the **react** package is imported
- 4) After the 3. step the functional component is defined (usually an arrow-function). The first character is also an uppercase and the camel-case format is used.
- 5) At the end of the file, the function is exported: **export default ComponentName;**
- 6) The functional component can be imported in other components, but the first character of the component must be an uppercase character. Lowercase elements are always JSX (HTML) elements.

e.g. import **Task** from “./Task/Task”;

To-Do's

To-Do's

New To-Do

Open

Completed

Test



To-Do's

To-Do's

New To-Do

New Task Title:

Add

React Hooks

- What are React Hooks?
 - Before hooks were introduced, there were class components with which you could save a state of a component. After hooks were introduced, you don't need class components anymore and you should not use them anymore.
 - React Hooks are only useable in functional components.
 - Hooks look like the following: **use[NameOfHook]() e.g.: useState()**
 - First is **use** and after is the **name** of the hook
 - With hooks it is possible to have a state in a functional component with which you are able to save multiple states: **useState()**
 - It is also possible to execute code **before** a component is **rendered** or when a state in the component is **changed**. Before hooks you would have to use different **lifecycle function** in **class components** but now you only need the **useEffect()** hook.
 - You can use the same kind of Hooks multiple times in a component.
- Why were react hooks introduced?
 - It is possible to **reuse „stateful logic“** without **changing** the component **hierarchy**.
 - A Big class component can be broken down into multiple better readable functional components.
 - **Overhead** to learn React is **reduced** with the introduction of **hook**.

- React Hook rules
 - React Hooks are **not** allowed to be executed in **loops**, **conditions** or **nested** functions
 - React Hooks are **only** allowed to be **called** in **functional** components
- How does React know which state corresponds to which useState?
 - As mentioned before you can use the Hook multiple times in a component.
 - React relies on the order in which Hooks are called.
 - The order of the Hook calls is the same on every render. Like this React can associate some local state with each of them.

```
// 🚫 We're breaking the first rule by using a Hook in a condition
if (name !== '') {
  useEffect(function persistForm() {
    localStorage.setItem('formData', name);
  });
}
```

```
useEffect(function persistForm() {
  // 👍 We're not breaking the first rule anymore
  if (name !== '') {
    localStorage.setItem('formData', name);
  }
});
```

- What React Hooks are there?
 - Basic Hooks
 - useState
 - useEffect
 - useContext
 - Other Hooks
 - useReducer
 - useCallback
 - useMemo
 - useRef
 - useImperativeHandle
 - useLayoutEffect
 - useDebugValue

- useState:
 - Adds a local state object to a functional component
 - The state object still exists after a re-render of the component
 - useState function returns a pair of the current state of the state object and a function with which the state object can be changed.
 - useState has only one parameter, which is the initial value of the state object. The initial value is only set in the first render
 - It is possible to use multiple useState hooks in one functional component.

```
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 }
```

Example on how to use the useState hook

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;
```

NewTask.js from the ToDo-App

New Task Title:

Add

- useEffect:
 - In react there are all kinds of side-effects e.g. requesting data from a API, changing the DOM. All these kind of actions are called side-effects, because these actions can not be executed while the component is rendering.
 - With the useEffect hook it is possible to use these kind of side-effects.
 - If useEffect is used it tells react, that side-effect should be executed after changes to the DOM where made.
 - UseEffect combines some of the old lifecycle functions (componentDidMount, componentDidUpdate *and* componentWillUnmount)
 - The same as with useState is true for useEffect as there can be multiple useEffect hooks in component.
 - UseEffect hook is called after every render, even after the first render.
 - It is also possible to tell the useEffect hook when it should be called e.g. if it should only be called once or when a specific state object is change. To do this you use [] as the second parameter of the useEffect hook.
 - Within [] put the state object you want to look out for if these change or if the [] are empty the useEffect is called only once when the component is created

```

5  import './ShowToDo.css'
6  import { useState,useEffect } from "react";
7
8  const ShowToDo = ()=>{
9      const [tasks, setTasks] = useState([])
10
11      useEffect(()=>{
12          if(tasks.length === 0){
13              setTasks([{id:1,title:"Task 1",completed:false},
14                  {id:2,title:"Task 2",completed:true},
15                  {id:3,title:"Task 3",completed:false}]);
16          }
17      } /* ...
25      },[])
26
27      const moveTask = (taskId)=>{ ...
44      }
45
46      const deleteTask = (taskId) =>{ ...
59      }
60
61      const completedTasks = [];
62      const openTasks = [];
63      for(const task of tasks){ ...
71      }
72
73      return(
74          <Content>
75              <div className="ShowToDo">
76                  <div className="ShowToDo__Wrapper">
77                      <h2>
78                          Open
79                          <hr />
80                      </h2>
81                      <div>
82                          {openTasks}
83                      </div>
84                  </div>
85                  <div className="ShowToDo__Wrapper">
86                      <h2>
87                          Completed
88                          <hr />
89                      </h2>
90                      <div>
91                          {completedTasks}
92                      </div>
93                  </div>
94              </div>
95          </Content>
96      )
97  }
98
99  export default ShowToDo;

```

Using empty [] to let the useEffect only executed once or if you want to let the useEffect only execute if a state-variable, like in this example **tasks**, is changed. If you remove the [] the useEffect is executed everytime something is changing within the component

```

11      useEffect(()=>{
12          if(tasks.length === 0){
13              setTasks([{id:1,title:"Task 1",completed:false},
14                  {id:2,title:"Task 2",completed:true},
15                  {id:3,title:"Task 3",completed:false}]);
16          }
17      } /* ...
25      },[])
26

```

To-Do's

To-Do's

New To-Do

Open

Completed

- Custom Hooks:
 - Beside the current existing Hooks it is also possible to write custom Hooks.
 - You are able to extract component logic into reusable functions
 - We can also use Hooks to share functionality between two or more components without copy&paste the same code.
 - A custom Hook is a JavaScript function whose name starts with **use** and that may call other Hooks.
 - Unlike a React component, a custom Hook doesn't need to have a specific signature. We can decide what it takes as arguments and how many as well as what our custom Hook, if anything, should return.

```
import React, { useState, useEffect } from 'react';

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

```
import React, { useState, useEffect } from 'react';

function FriendListItem(props) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });

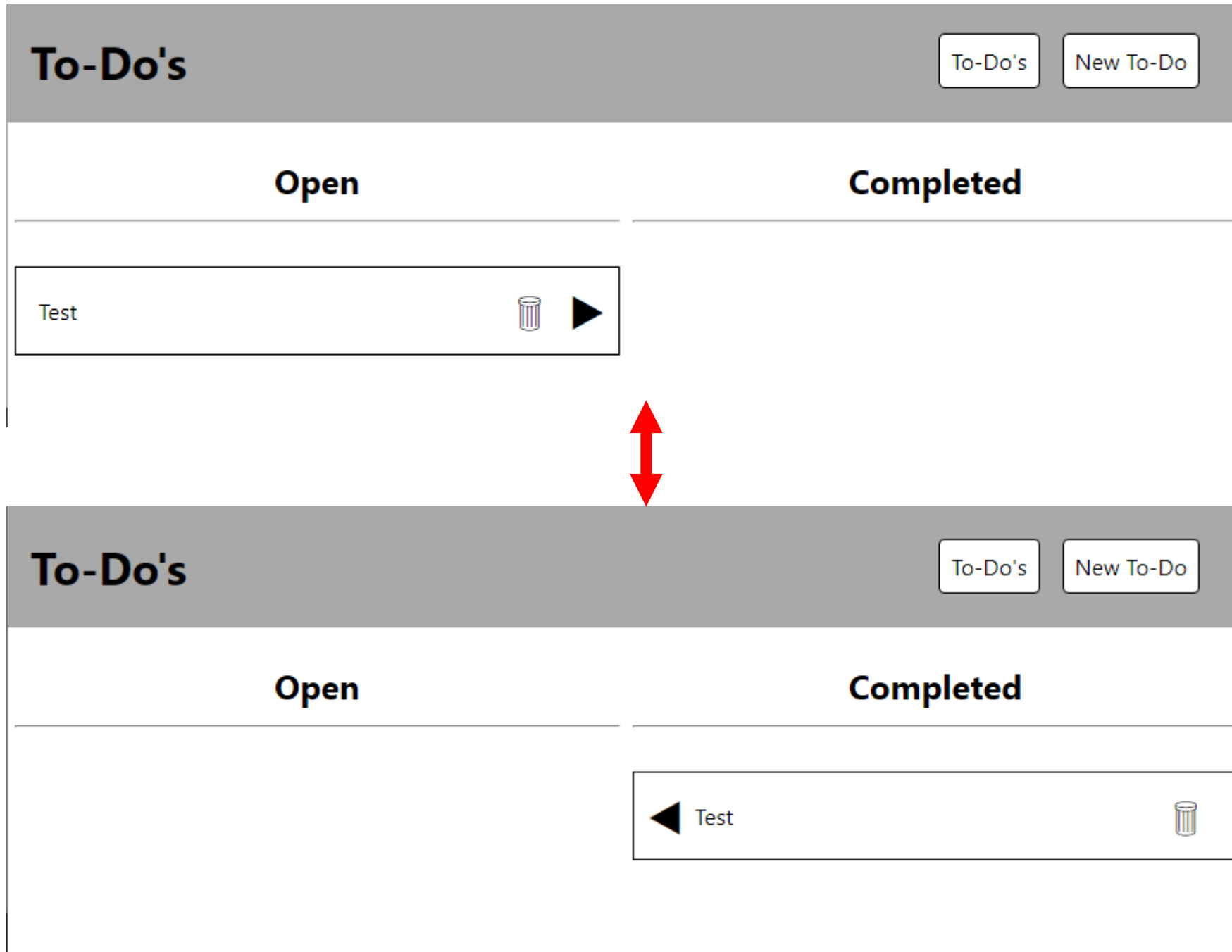
  return isOnline;
}
```

```
function FriendStatus(props) {  
  const isOnline = useFriendStatus(props.friend.id);  
  
  if (isOnline === null) {  
    return 'Loading...';  
  }  
  return isOnline ? 'Online' : 'Offline';  
}
```

```
function FriendListItem(props) {  
  const isOnline = useFriendStatus(props.friend.id);  
  
  return (  
    <li style={{ color: isOnline ? 'green' : 'black' }}>  
      {props.friend.name}  
    </li>  
  );  
}
```


React Event-Handling

- The same events that exist in the normal JavaScript also exist in React.
The only difference is, that the events in React are written in Camel-Case.
- The event you want to handle needs to be added as an attribute/property to the JSX-Element
 - The event is called for example like this: **event={someFunction}**
 - **Important:** It is important that you do not use () and only pass the function reference.
If not the function will be executed instantly and not when the event occurs.
- Through events it is also possible to change the state object (execute the set-function from the useState() pair)
- The function which should handle an event can be called anything, but you will often see that these functions are called something like: **someFunctionHandler()**, **onSomethingChange()**
- If the function which should handle the event needs some kind of parameter you should enclose this function with an arrow-function



ShowToDo component (moveTask, deleteTask and for loop)

```

JS ShowToDo.js X # ShowToDo.css # Task.css JS Task.js # Content.css 1 JS Content.js # Header.css JS Header.js
src > components > ShowToDo > JS ShowToDo.js > ShowToDo > moveTask
27 const moveTask = (taskId)=>{
28     const taskIndex = tasks.findIndex((v)=>{return v.id === taskId});
29
30     const task = {...tasks[taskIndex]};
31     task.completed = !task.completed;
32
33     const tasksCopy = [...tasks];
34     tasksCopy[taskIndex] = task;
35     setTasks(tasksCopy)
36
37 > /* ...
44 }
45
46 const deleteTask = (taskId) =>{
47     const taskIndex = tasks.findIndex((v)=>{return v.id === taskId});
48     const task = {...tasks[taskIndex]}
49     const tasksCopy = [...tasks];
50     tasksCopy.splice(taskIndex,1);
51     setTasks(tasksCopy)
52 > /* ...
59 }
60
61 const completedTasks = [];
62 const openTasks = [];
63 for(const task of tasks){
64     if(task.completed){
65         completedTasks.push(<Task task={task} key={task.id}
66             moveTask={()=>moveTask(task.id)} deleteTask={()=>deleteTask(task.id)}/>)
67     }else{
68         openTasks.push(<Task task={task} key={task.id}
69             moveTask={()=>moveTask(task.id)} deleteTask={()=>deleteTask(task.id)}/>)
70     }
71 }
72

```

Here we set props-attributes **moveTask** and **deleteTask** which are arrow-Functions which call **moveTask** or **deleteTask**

We do this because we want that the Task component is able to call these function if a click-events occurs

```
60
61   const completedTasks = [];
62   const openTasks = [];
63   for(const task of tasks){
64     if(task.completed){
65       completedTasks.push(<Task task={task} key={task.id}
66         moveTask={()=>moveTask(task.id)} deleteTask={()=>deleteTask(task.id)}/>)
67     }else{
68       openTasks.push(<Task task={task} key={task.id}
69         moveTask={()=>moveTask(task.id)} deleteTask={()=>deleteTask(task.id)}/>)
70     }
71   }
72
```

Task Component

Test



Within the Task component we use the props-attributes **moveTask** and **deleteTask** and call these function if a click-event occurs

```
6   if(props.task.completed){
7     arrowLeft = [
8       <div className="Task_ArrowLeft" onClick={props.moveTask}>
9         &#9664;
10      </div>
11    ];
12   }else{
13     arrowRight = (
14       <div className="Task_ArrowRight" onClick={props.moveTask}>
15         &#9654;
16       </div>
17     )
18   }
```

Test  

 Test 

```
19   return (
20     <div className="Task">
21       {arrowLeft}
22       <div className="Task_Content">
23         <div className="Task_Content_Title">
24           {props.task.title}
25         </div>
26         <div className="Task_Content_Delete" onClick={props.deleteTask}>
27           <div>
28             &#128465;
29           </div>
30         </div>
31       </div>
32       {arrowRight}
33     </div>
34   )
35 }
```

```
# Content.css 1 # index.css JS Content.js JS App.js JS ShowToDo.js # ShowToDo.css JS Task.js X #
src > components > Task > JS Task.js > [Task]
1
2 import './Task.css'
3 const Task = (props) => {
4   let arrowLeft = null;
5   let arrowRight = null;
6   if (props.task.completed) {
7     arrowLeft = (
8       <div className="Task_ArrowLeft" onClick={props.moveTask}>
9         &#9664;
10      </div>
11    );
12   } else {
13     arrowRight = (
14       <div className="Task_ArrowRight" onClick={props.moveTask}>
15         &#9654;
16      </div>
17     );
18   }
19   return (
20     <div className="Task">
21       {arrowLeft}
22       <div className="Task_Content">
23         <div className="Task_Content_Title">
24           {props.task.title}
25         </div>
26         <div className="Task_Content_Delete" onClick={props.deleteTask}>
27           <div>
28             &#128465;
29           </div>
30         </div>
31       </div>
32       {arrowRight}
33     </div>
34   );
35 }
36
37
38 export default Task;
```

- Function reference components:
 - As we could see in the last few slides it is possible to pass functions as a reference to components.
 - Function references can be accessed through the props parameter of the component and the attributes you created
 - As we could see if you need to pass arguments to a function, which should be executed in a component/event you should enclose this function in an arrow-function:
 - `<div attribute = {()=>{myFunction(Parameter)}} />`
 - In this way you are able to use `()` and the function is not instantly executed because of the enclosing arrow-function. Only if the arrow-function is executed, is the function within executed

```

61  const completedTasks = [];
62  const openTasks = [];
63  for(const task of tasks){
64    if(task.completed){
65      completedTasks.push(<Task task={task} key={task.id}
66        moveTask={()=>moveTask(task.id)} deleteTask={()=>deleteTask(task.id)}/>)
67    }else{
68      openTasks.push(<Task task={task} key={task.id}
69        moveTask={()=>moveTask(task.id)} deleteTask={()=>deleteTask(task.id)}/>)
70    }
71  }

```

Task Component

Test




Deleting/Change data from a state object

- Deleting/Change data from a state object
 - If you want to delete or change data from a state object, you first have to create a copy of the state object you want to delete/change.
 - If the data you want to change is within an array or is an array you have to follow the following steps:
 - 1) Find index of the data you want to change in the array
 - 2) Create a copy of the object to want to change through the spread operator `{...obj}`
 - 3) Change the copied object
 - 4) Copy the complete array through spread operator `[...array]`
 - 5) In the copied array overwrite the object with the new object at the index
 - 6) Update state-object with the copied and changed array using set-function from pair from `useState()`
 - If you want to delete data from an array follow the following steps:
 - 1) Find index of the data you want to change in the array
 - 2) Copy the complete array through spread operator `[...array]`
 - 3) In the copied array call the `splice` function from JavaScript and the first parameter is the index of the data you want to delete from the array and the second parameter is 1 (as you only want one element to be deleted from the array)
 - 4) Update state-object with the copy and changed array using set-function from pair from `useState()`

Changing data in an array (Function from ShowToDo.js component)

```
7   const moveTask = (taskId)=>{  
8       const taskIndex = tasks.findIndex((v)=>{return v.id === taskId});  
9  
10      const task = {...tasks[taskIndex]};  
11      task.completed = !task.completed;  
12  
13      const tasksCopy = [...tasks];  
14      tasksCopy[taskIndex] = task;  
15      setTasks(tasksCopy)  
16  
17  > /* ...  
14  }  
15
```

Delete data from an array (Function from ShowToDo.js component)

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

Using an array to create multiple components

- Using an array to create multiple components
 - If you have an array filled with data which you want to display it would be tedious work not dynamic to write for each data entry the component from hand.
 - For this it is possible to iterate over an array and create an array with react components which are later displayed
 - The important thing which you need to take note of is, that the components need a **key** attribute which is unique (In most cases data from an api have an id which you can use as a key)

In this component we want to show open ToDo tasks and completed ToDo tasks

- 1) In the useEffect hook we set the tasks (later we use our api server to retrieve the tasks)
- 2) After there are data in the **tasks** object, we iterate over it and create multiple Task components, which are saved in two different Arrays (**completedTasks**, **openTasks**).
- 3) We set the **key** attribute to the task **id**. Furthermore we pass two functions as reference to the component (**moveTask**, **deleteTask**)

```
useEffect(()=>{
  if(tasks.length === 0){
    setTasks([
      {id:1,title:"Task 1",completed:false},
      {id:2,title:"Task 2",completed:true},      1.
      {id:3,title:"Task 3",completed:false}]);
  }
  /* ...
},[])
```

```
const completedTasks = [];
const openTasks = [];
for(const task of tasks){ 2.
  if(task.completed){      3.
    completedTasks.push(
      <Task task={task} key={task.id}
        moveTask={()=>moveTask(task.id)} deleteTask={()=>deleteTask(task.id)}/>
    )
  }else{
    openTasks.push(
      <Task task={task} key={task.id}
        moveTask={()=>moveTask(task.id)} deleteTask={()=>deleteTask(task.id)}/>
    )
  }
}
```

```

8  const ShowToDo = ()=>{
9      const [tasks, setTasks] = useState([])

10
11      useEffect(()=>{
12          if(tasks.length === 0){
13              setTasks([
14                  {id:1,title:"Task 1",completed:false},
15                  {id:2,title:"Task 2",completed:true},
16                  {id:3,title:"Task 3",completed:false}
17              ]);
18          }
19      }
20      /* ...
21      },[])
22
23      const moveTask = (taskId)=>{...
24      }
25
26      const deleteTask = (taskId) =>{ ...
27      }
28
29      const completedTasks = [];
30      const openTasks = [];
31      for(const task of tasks){
32          if(task.completed){
33              completedTasks.push(
34                  <Task task={task} key={task.id}
35                  moveTask={()=>moveTask(task.id)}
36                  deleteTask={()=>deleteTask(task.id)}/>
37              )
38          }else{
39              openTasks.push(
40                  <Task task={task} key={task.id}
41                  moveTask={()=>moveTask(task.id)}
42                  deleteTask={()=>deleteTask(task.id)}/>
43              )
44          }
45      }
46  }

```


Show data through conditions

- Show data through conditions:
 - It is often the case that you want to show some data to user **only** if specific **conditions** are met. If these conditions are **not** met **nothing** is shown to the user
 - Through conditions and showing data only if these conditions are met it is possible to **create** and **control the flow of a app**.
 - In most cases the **state** object is used and checked if it is **filled** with data or if a state object ist set to **true**.

In the return statement of the ShowToDo component, we put the arrays within the {}. (1) If there are objects within these arrays, it will be displayed to client, else the array is empty and nothing is displayed.

```

73     return(
74         <Content>
75             <div className="ShowToDo">
76                 <div className="ShowToDo__Wrapper">
77                     <h2>
78                         Open
79                         <hr />
80                     </h2>
81                     <div>
82                         {openTasks}1.
83                     </div>
84                 </div>
85                 <div className="ShowToDo__Wrapper">
86                     <h2>
87                         Completed
88                         <hr />
89                     </h2>
90                     <div>
91                         {completedTasks}1.
92                     </div>
93                 </div>
94             </div>
95         </Content>
96     )
97 }

```



In the Task component we check, if the task is (1) completed by using the **props** parameter and access the **task** object and within this object the **completed** property.

If the task is completed we change our (2) **arrowLeft** variable from **null** to a JSX-Div component if not we change **arrowRight**. This way either of the two arrows will be shown and if a variable is **null** it is ignored.

We also use our (3) function-reference in **props.moveTask** or **props.deleteTask** and set the **onClick** event

```

if(props.task.completed){ 1.
  2. arrowLeft = (
    3. <div className="Task_ArrowLeft" onClick={props.moveTask}>
      &#9664;
    </div>
  );
}else{
  arrowRight = (
    <div className="Task_ArrowRight" onClick={props.deleteTask}>
      &#128465;
    </div>
  );
}

return (
  <div className="Task">
    {arrowLeft}
    <div className="Task_Content">
      <div className="Task_Content_Title">
        {props.task.title}
      </div>
      <div className="Task_Content_Delete" onClick={props.deleteTask} 3.
        <div>
          &#128465;
        </div>
      </div>
    </div>
    {arrowRight}
  </div>
)

export default Task;

```

```
1
2 import './Task.css'
3 const Task = (props)=>{
4   let arrowLeft = null;
5   let arrowRight = null;
6
7   if(props.task.completed){
8     arrowLeft = (
9       <div className="Task_ArrowLeft" onClick={props.moveTask}>
10         &#9664;
11       </div>
12     );
13   }else{
14     arrowRight = (
15       <div className="Task_ArrowRight" onClick={props.moveTask}>
16         &#9654;
17       </div>
18     )
19   }
20   return (
21     <div className="Task">
22       {arrowLeft}
23       <div className="Task_Content">
24         <div className="Task_Content_Title">
25           {props.task.title}
26         </div>
27         <div className="Task_Content_Delete" onClick={props.deleteTask}>
28           <div>
29             &#128465;
30           </div>
31         </div>
32       </div>
33       {arrowRight}
34     </div>
35   )
36 }
37 export default Task;
```

Component Styling with Stylesheets

- Component Styling with Stylesheets
 - To design your component you can use **CSS**
 - As mentioned in the beginning you can do this through a **stylesheet file**
 - Styling a component is done in the most cases through **CSS-classes**
 - But as **class** is a keyword in JavaScript. To add a Styling-Class to a JSX-tag you need to name it **className=““**
 - The CSS file is **loaded** when the component is rendered the **first time** and the CSS-classes are **global**
 - To be able to use the CSS file you need to **import** the file like: **import „./style.css“;**

```

JS ShowToDo.js x # ShowToDo.css # Task.css JS Task.js # Content.css 1 JS
src > components > ShowToDo > JS ShowToDo.js > ShowToDo
5 import './ShowToDo.css'
6 import { useState,useEffect } from "react";
7
8 const ShowToDo = ()=>{
9   const [tasks, setTasks] = useState([])
10
11   useEffect(()=>{
12     if(tasks.length === 0){
13       setTasks([{id:1,title:"Task 1",completed:false},
14         {id:2,title:"Task 2",completed:true},
15         {id:3,title:"Task 3",completed:false}]);
16     }
17   } /* ...
25 },[])
26
27 const moveTask = (taskId)=>{ ...
44 }
45
46 const deleteTask = (taskId) =>{ ...
59 }
60
61 const completedTasks = [];
62 const openTasks = [];
63 for(const task of tasks){ ...
71 }
72
73 return(
74   <Content>
75     <div className="ShowToDo">
76       <div className="ShowToDo__Wrapper"> ...
84     </div>
85     <div className="ShowToDo__Wrapper"> ...
93   </div>
94   </Content>
95 )
96 }
97
98
99 export default ShowToDo;

```

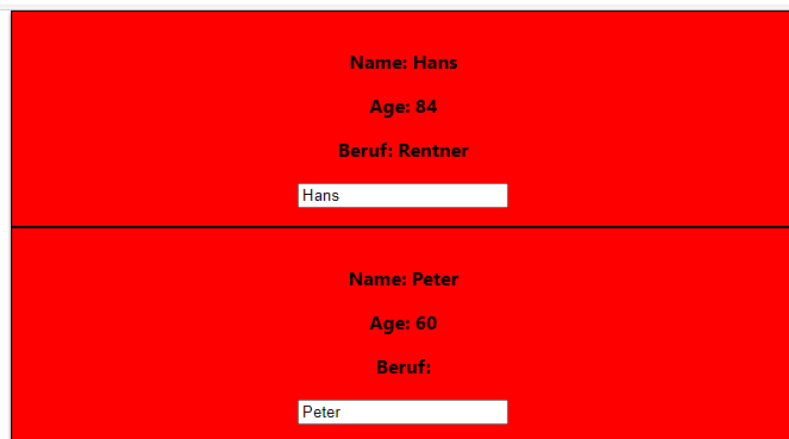
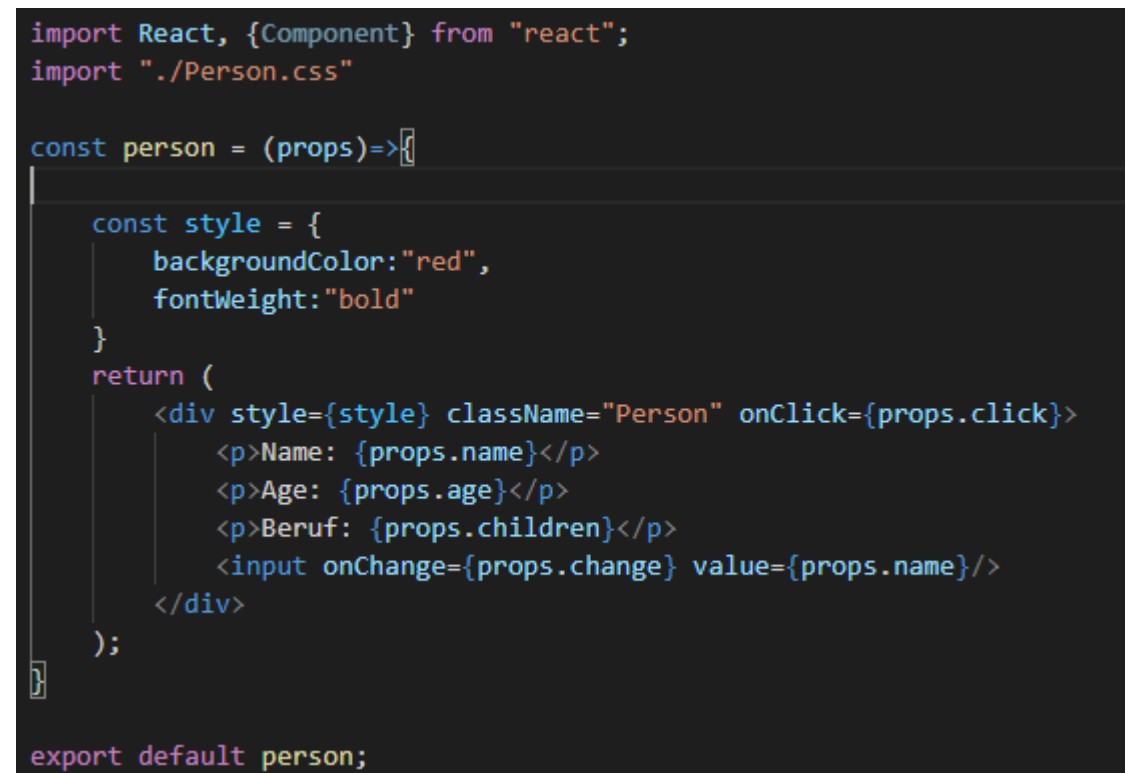
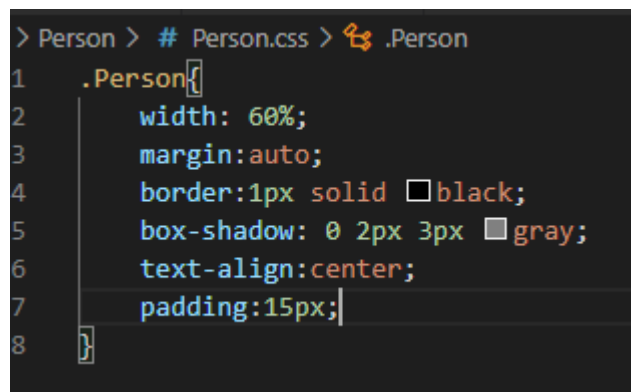
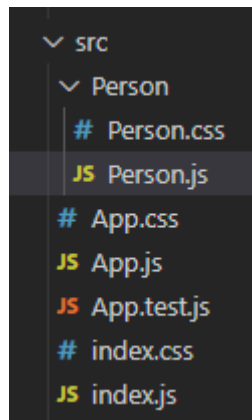
```

JS ShowToDo.js # ShowToDo.css x # Task.css JS Task
src > components > ShowToDo > # ShowToDo.css > .ShowToDo__Wra
1 .ShowToDo{
2   display: flex;
3   justify-content: space-between;
4   flex: 0 0 100%;
5 }
6
7 .ShowToDo__Wrapper{
8   display: flex;
9   flex: 1;
10  flex-direction: column;
11  padding: 0.3em;
12 }

```


Component Styling with Inline-Styles

- Component Styling with Inline-Styles
 - Sometimes it is necessary to use inline-styling at a component.
 - To use inline-styling you need a constant object in which we write the css statements in Camel-Case
 - e.g.: **background-color** => **backgroundColor**
 - The styling-object will be passed to style attribute of a JSX-tag
 - e.g..: **<div style={stylingObjekt}>...</div>**
 - Hoover effects are not possible with inline-styles



- Exercise 5
 - 1) Create a new React-App
 - 2) Create components from an array of data
 - 3) This App should **show** to-do tasks
 - 4) you should be able to **change** task to completed/uncompleted and show this visually in the app.
 - 5) You should be able to delete a task
 - 6) Style the different Components

Hint: After creating the initial react project, all the code you need for this exercise is already in the slides and needs to be assembled correctly

CSS and JavaScript for Exercise 3

```

Anzeigen  Generiere  Ausführen  Terminal  Hilfe
JS ShowToDo.js  # ShowToDo.css X  # Task.css
src > components > ShowToDo > # ShowToDo.css > .ShowT
1  .ShowToDo{
2      display: flex;
3      justify-content: space-between;
4      flex: 0 0 100%;
5  }
6
7  .ShowToDo__Wrapper{
8      display: flex;
9      flex: 1;
10     flex-direction: column;
11     padding: 0.3em;
12 }

```

```

JS ShowToDo.js # ShowToDo.css # Task.css X JS Task.js
src > components > Task > # Task.css > 🔗 .Task:not(:first-child)

1  .Task{
2      display: flex;
3      border: 1px solid black;
4      padding: 0.5em;
5      align-items: center;
6  }
7
8  .Task:not(:first-child){
9      margin: 0.25em 0;
10 }
11
12 .Task_Content{
13     display: inherit;
14     justify-content: space-between;
15     width: 100%;
16 }
17
18 .Task_Content_Title, .Task_Content_Delete{
19     display: flex;
20     align-items: center;
21     padding: 0 0.5em;
22 }
23
24 .Task_Content_Delete{
25     font-size:xx-large;
26     justify-content: right;
27     cursor: pointer;
28 }
29
30 .Task_ArrowLeft, .Task_ArrowRight{
31     font-size:xx-large;
32     cursor: pointer;
33 }

```

Header.css

JS Header.js X

JS ShowToDo.js

ShowToDo.css

Task.css

JS Task.js

Content

src > components > Header > JS Header.js > [X] 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          <div to="/" className="Header__Button">To-Do's</div>
15          <div to="/add" className="Header__Button">New To-Do</div>
16        </div>
17      </div>
18    )
19  }
20
21  export default Header;

```



```
# Header.css X JS Header.js JS ShowToDo.js # ShowToDo.js
src > components > Header > # Header.css > .Header_Button
1  .Header{
2      padding: 0em 1em;
3      background-color: darkgray;
4      display: flex;
5      justify-content: space-between;
6      align-items: center;
7  }
8
9
10 .Header__Buttons{
11     display: flex;
12     margin: 0.5em;
13 }
14
15 .Header__Button{
16     padding: 0.5em;
17     margin: 0 0.5em;
18     border: 1px solid black;
19     background-color: white;
20     border-radius: 0.25em;
21     text-decoration: none;
22     color: black;
23 }
```

```

Anzeigen  Generiere zu  Ausführen  Terminal  Hilfe

JS Content.js x # Content.css 1 # Header.css 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;

```

```

Anzeigen  Generiere zu  Ausführen  Terminal

JS Content.js # Content.css 1 x #

src > components > Content > # Content.css
1  .Content{
2  }

```