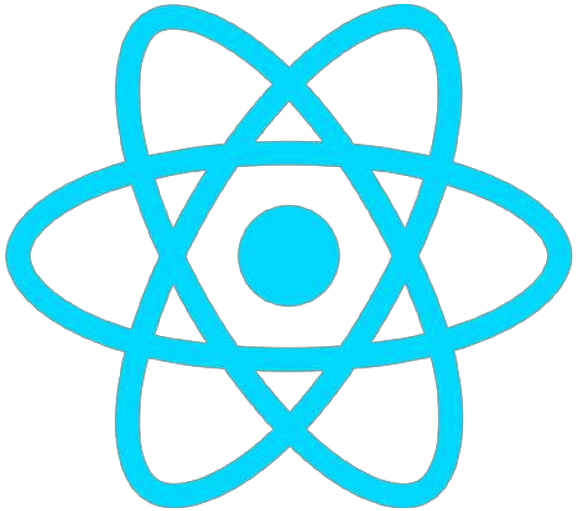
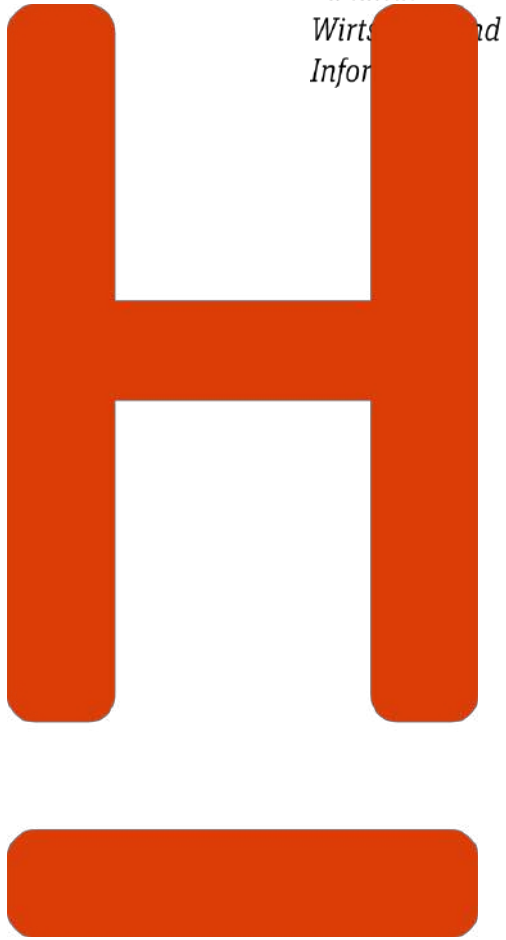
HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirts[illegible]nd
Infor[illegible]

Web applications with



React



Introduction round

Kelvin Homann

- Born 1998 in Neustadt a. Rbge.
- 2017 - 2021 Bachelor's degree program in Media Design Informatics
- 2021 Start of professional life (IT Consultant Software Engineering)

How can you reach me? kelvin.homann@gmail.com

Who are you?

- Name
- Age
- Previous experience in software development/app development?
- Which development environment?
- Expectations and wishes for the course

Web development basics?

HTML, CSS and Javascript

HTML

What is HTML?

HTML stands for Hypertext Markup Language.
It is the structural language (not a programming language!) for websites and defines the meaning of content.

Basic HTML tags:
<html>: Root element of the document.
<head>: Contains meta information such as titles and links to external files.
<body>: Contains the actual content of the page.
<h1>, <p>, <ul>, <li>: Tags for headings, paragraphs, unordered lists and list elements.

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Meine Webseite</title>
</head>
<body>
  <h1>Willkommen!</h1>
  <p>Dies ist meine erste Webseite.</p>
</body>
</html>
```

The Document Object Model (DOM)

What is the DOM?

The Document Object Model (DOM) is a programming interface for HTML documents.

It represents the structure of a document as a tree, in which each element is represented as a node in the tree.

Why is the DOM important?
Dynamic interaction: Allows the content and structure of a website to be changed after loading.

Standardized interface: Provides a standardized way to access HTML documents.

DOM: Document Object Model

Hierarchical structure

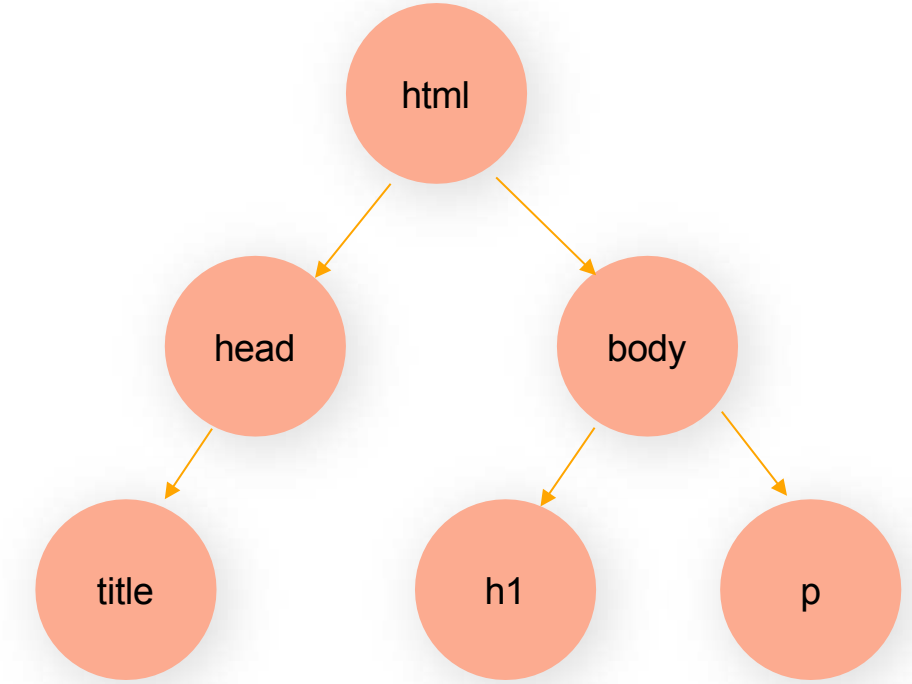
- Each HTML element is a node in the tree.
- Elements can be nested and have parents, children and siblings.

DOM elements and attributes

- **Elements:** HTML tags such as <p>, , <h1>, etc.
- **Attributes:** Properties of elements, e.g., id, class, src.

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Meine Webseite</title>
</head>
<body>
  <h1>Willkommen!</h1>
  <p>Dies ist meine erste Webseite.</p>
</body>
</html>
```



CSS

What is CSS?

CSS stands for Cascading Style Sheets.
CSS designs the appearance and layout of HTML documents.

Styling:
Colors, fonts, spacing, etc.

Layout:
Arrangement of elements on the page.

Animations and transitions:
Movements and transitions for an interactive experience.

Basic CSS rules:
Selectors select HTML elements.

Properties set the appearance of the selected elements.

Combining HTML and CSS

Inline

index.html

```
<html>
  <body>
    <div style="color: red; font-size: 16px;">Text</div>
  </body>
</html>
```

Internal stylesheet

index.html

```
<html>
  <head>
  <style>
    body { background-color: #f0f0f0; }
    h1 { color: blue; }
  </style>
</head>
  <body>
    ...
  </body>
</html>
```

External stylesheet

Best practice: Separation of HTML and CSS for better maintainability.

index.html

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="styles.css">
  </head>
  <body>
    ...
  </body>
</html>
```

CSS frameworks

What are CSS frameworks?

CSS frameworks are ready-made CSS libraries that speed up development. They offer predefined classes for layout, typography, forms and more.

Example: Bootstrap

CDN integration: Integration via Content Delivery Network (CDN) for easy use.

index.html

```
<html>
  <head>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
  </head>
  <body>
    ...
  </body>
</html>
```

Use of bootstrap classes

Container: - Limits the content to a centered column.
Grid system: and - Simple column layouts.
Navigation: <nav class="navbar"> - Creation of responsive navigation bars.
Components: <button class="btn">, <input class="form-control"> - Predefined UI elements.

Javascript

What is JavaScript?

JavaScript is (among other things) a programming language for websites.

Javascript enables interaction with the user and the modification of HTML and CSS.

index.html

```
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Hello World</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
  <script>
    // Variablen
    let x = 5;

    // Funktionen
    function sayHello() {
      console.log("Hello!");
    }

    // Klassen
    class Student {
      constructor(name) {
        this.name = name;
      }

      hello() {
        alert("Hello, " + this.name + "!");
      }
    }

    // Event-Handling
    element.addEventListener("click", function () {
      /* Code hier */
    });
  </script>
</html>
```

Javascript and the DOM

JavaScript can manipulate the DOM to dynamically change content.

Examples:

- **getElementById:**
document.getElementById('demo')

- **getElementsByClassName:**
document.getElementsByClassName('highlight')

- **getElementsByTagName:**
document.getElementsByTagName('p')

JS beispiel.js

```
// Ändern des Textes eines HTML-Elements mit ID 'demo'
let element = document.getElementById('demo');
element.innerHTML = 'Neuer Text';
```

API documentation: https://developer.mozilla.org/en-US/docs/Web/API

DOM manipulation and events

DOM manipulation

innerHTML: Change the HTML content of an element.

style: Change CSS styles directly.

setAttribute: Set attributes.

Events and event handling

addEventListener: Add event listeners.

Event object: Information about the triggered event

JS beispiel.js

```
// Hinzufügen eines Klick-Event-Listeners
// zu einem Element mit ID 'myButton'
let button = document.getElementById('myButton');
button.addEventListener('click', function(event) {
  // Code, der bei einem Klick ausgeführt wird
  console.log('Button wurde geklickt!');
});
```

API documentation: https://developer.mozilla.org/en-US/docs/Web/API

DOM manipulation and creation of elements

Dynamic element creation

createElement:
Create new DOM elements.

appendChild and removeChild:
Adding and removing elements.

JS beispiel.js

```
// Erstellen und Hinzufügen eines neuen Listenelements
let newListElement = document.createElement('li');
newListElement.textContent = 'Neues Element';
document.getElementById('myList').appendChild(newListElement);
```

API documentation: https://developer.mozilla.org/en-US/docs/Web/API

How is Javascript integrated?

Inline Javascript

index.html

```
<html>
  ...
  <body>...</body>
  <script>
    // JavaScript-Code hier
  </script>
</html>
```

Per Javascript file

Best Practice

Separation of HTML and JavaScript for better readability and maintainability.

index.html

```
<html>
  <script src="my-script.js"></script>
  <body>...</body>
</html>
```

Questions: Web development basics

What role does CSS play in web development?

CSS designs the appearance and layout of HTML documents.

Briefly explain the Document Object Model (DOM) and how JavaScript interacts with it.

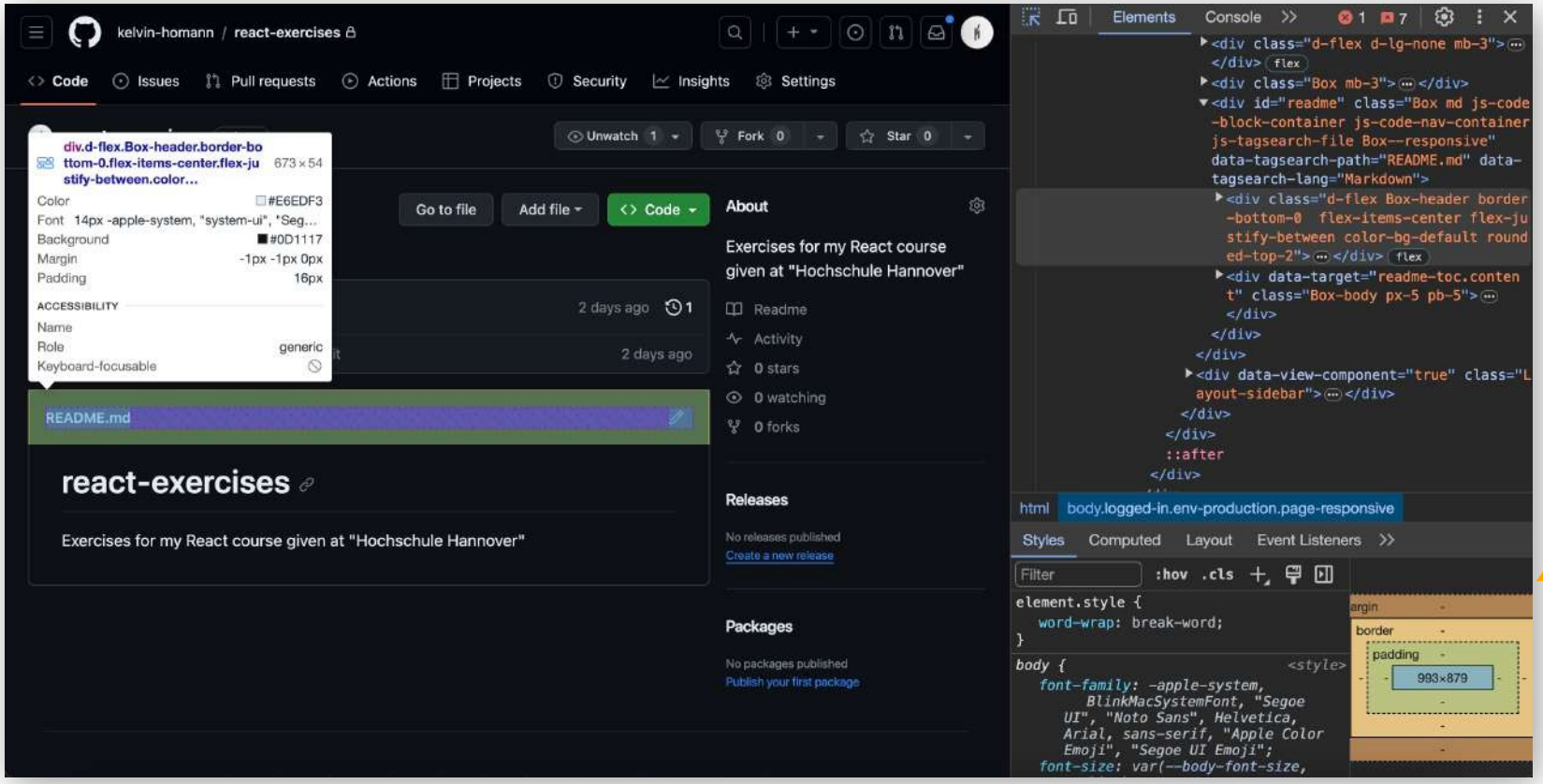
The Document Object Model (DOM) is a tree structure that represents the hierarchy of an HTML document; JavaScript interacts with it by selecting elements, manipulating content, changing the structure and reacting to events.

What are the advantages of separating HTML, CSS and JavaScript in web development?

The separation of HTML, CSS and JavaScript enables a clear structure, better maintainability and the reusability of code.

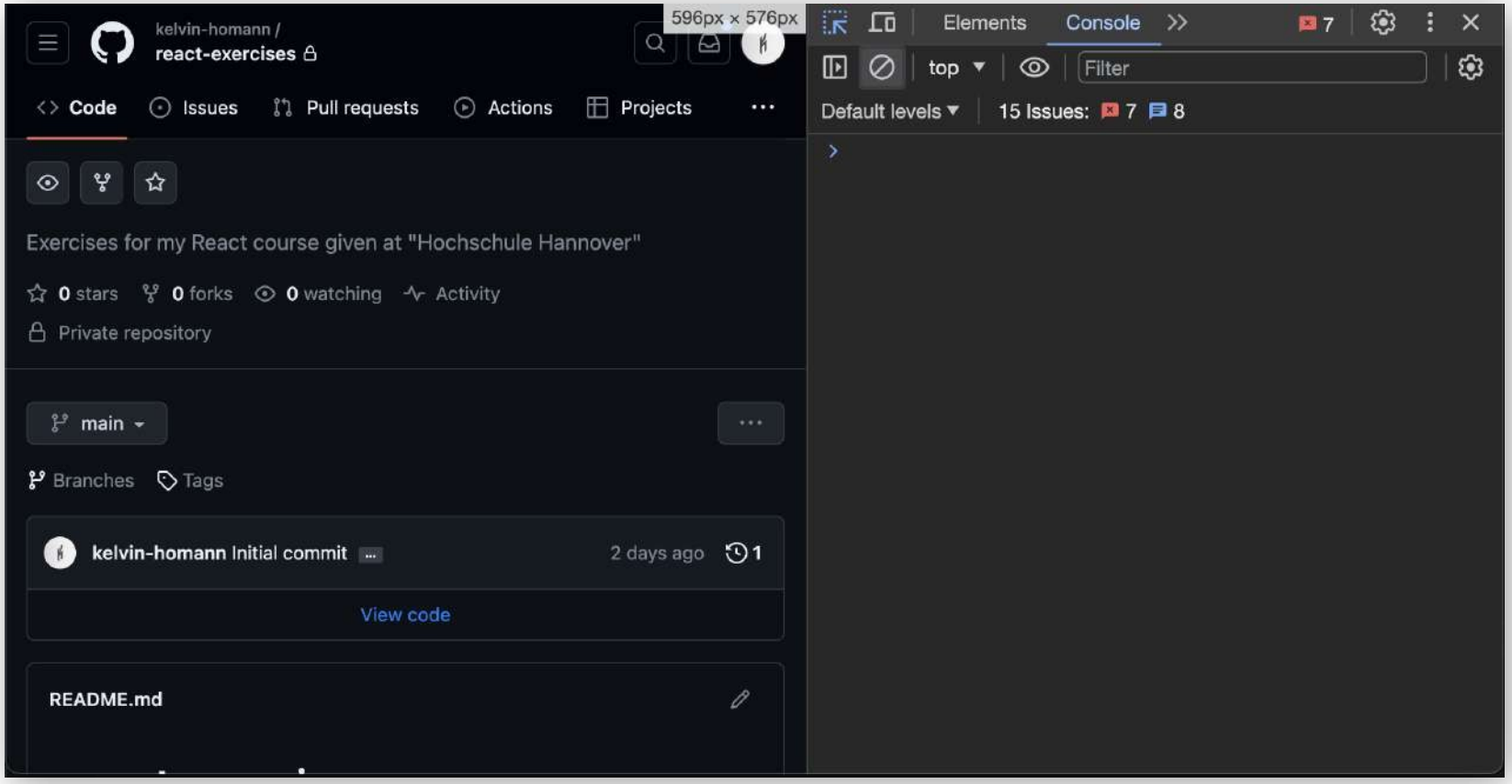
Chrome Developer Tools "Elements"

Open with F12 or right-click "examine"



Chrome Developer Tools "Console"

Open with F12 or right-click "examine"



Exercise: Basics

Create a new folder and open it in VSCode.
Then create the following 3 files in this folder:

- index.html
- index.css
- index.js

Writes the basic structure of an HTML document in index.html. Then integrate the index.css and index.js into the HTML file.

Finally, rebuild the website on the right:

- Blue button
- A counter is incremented when the button is clicked

React basics

What is React, components, hooks, etc.?

What is React?

Open source Javascript library developed by Meta

⚠ *Attention: React is **not** a framework*

Built for component-based development of single-page applications (SPAs) and mobile applications Uses a concept

called "Virtual DOM"

- Enables efficient processing of the effects of state changes on the DOM

https://github.com/facebook/react

Installation - New React project

Preconditions

Node: https://nodejs.org/en/

In the terminal:

```
npm create vite@latest
```

Then follow the selection options

Alternatively, finished project here:

```
git clone https://github.com/kelvin-homann/react-exercises
```

Exercise: New React project

Try the following command yourself

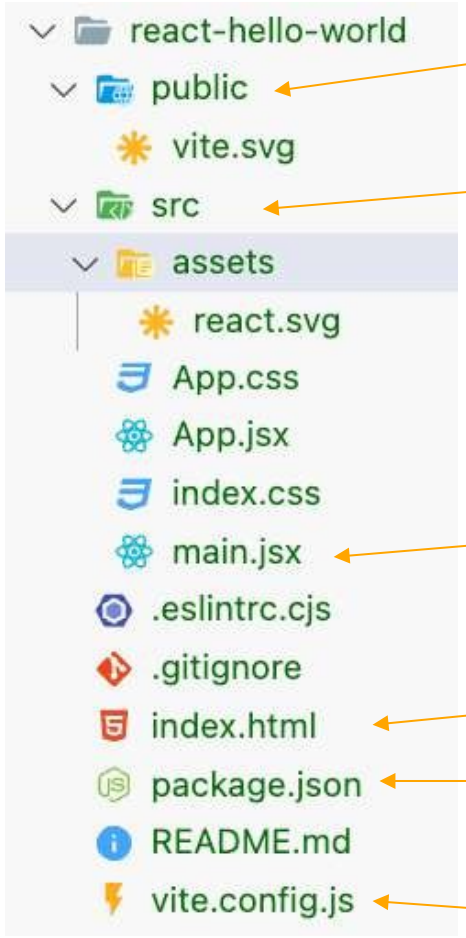
```
npm create vite@latest
```

```
cd my-project
```

```
npm install
```

```
npm run dev
```

React project structure



public folder
Static files (images, fonts etc.) are located here.

src folder
Here is the source code

main.jsx
Entrypoint of the React application

index.html
This is the "root" node of the React application

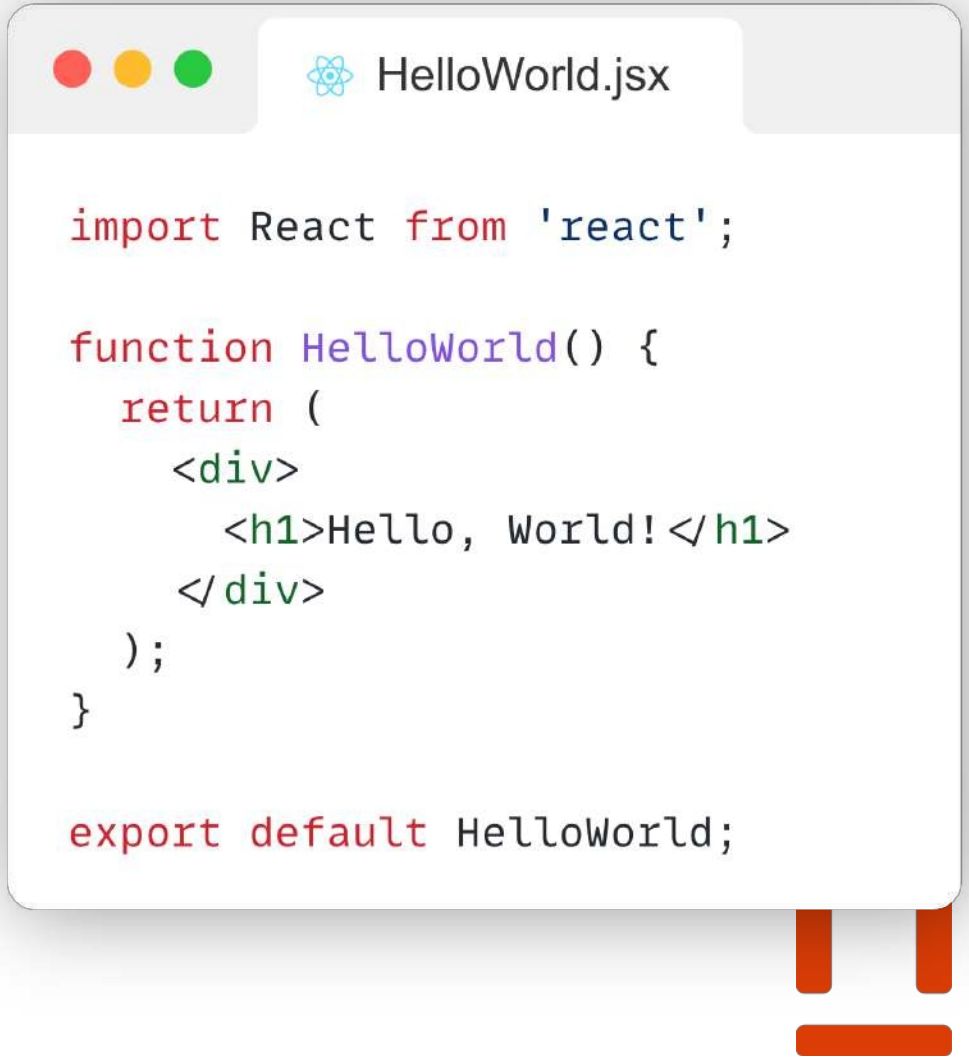
package.json
This is the heart of a Javascript application This is where scripts, dependencies etc. are configured

vite.config.js
The build is configured here

Components

Components

- Small reusable units
- Represent part of a user interface
- Created with the help of Javascript function or class
  - **Attention**: Class components are no longer state of the art
  - The only exception: ErrorBoundaries
  - Always return JSX



Virtual DOM (Virtual Document Object Model)

- Classic updating of the DOM is expensive
  - especially with many updates
- Virtual DOM is an in-memory representation of the DOM
- If the state of an application changes, the Virtual DOM is updated instead of modifying the DOM directly.
- React compares updated Virtual DOM with the previous Virtual DOM and identifies the minimum changes that need to be made to the real DOM to keep the application up to date

HelloWorld.jsx

```jsx
import React from 'react';

function HelloWorld() {
  return (
    <div>
      <h1>Hello, World!</h1>
    </div>
  );
}

export default HelloWorld;
```

JSX (Javascript XML)

- JSX is a syntax extension for JavaScript
- enables HTML-like syntax to be used in JavaScript code
- JSX is used in React components to display the content of the component
- If a component was written with JSX, React converts it into an object that corresponds to the Virtual DOM
- When the component is displayed in the browser, the content of the component is rendered into the real DOM.

HelloWorld.jsx

```jsx
import React from 'react';

function HelloWorld() {
  return (
    <div>
      <h1>Hello, World!</h1>
    </div>
  );
}

export default HelloWorld;
```

Props

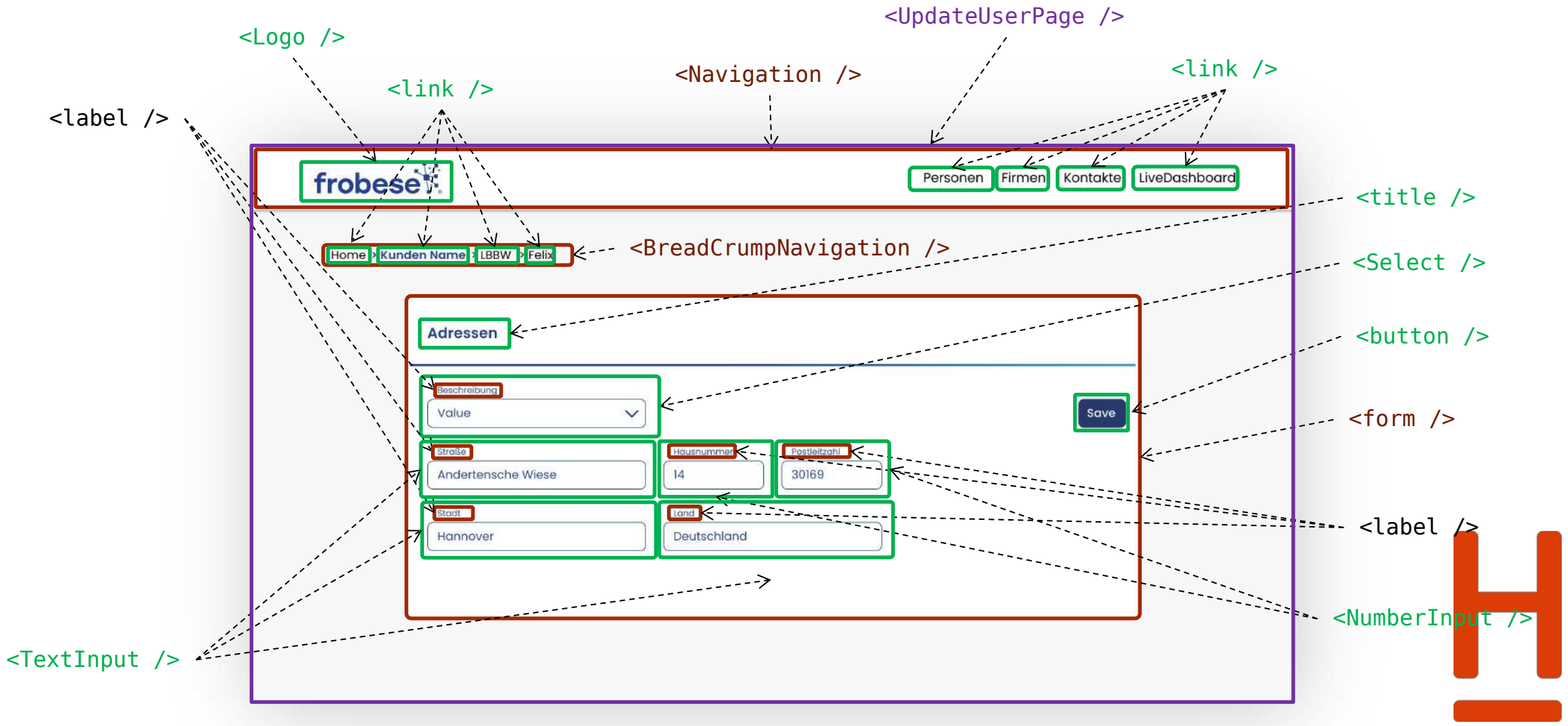
"properties"

- Possibility to transfer values from a component to its child components
- Props are transferred to components as attributes and can be used in the component to control its behavior and appearance.
- React re-renders all child components when a prop is passed through.

main.jsx

```jsx
function MyButton({ color, children }) {
  return (
    <button style={{ backgroundColor: color }}>
      {children}
    </button>
  )
}

function MyApp() {
  return (
    <>
      <MyButton color="blue">Click me</MyButton>
      <MyButton color="red">Click me too</MyButton>
    </>
  )
}
```

Think in components



Exercise components

Develops a `UserProfile` component

- Accepts 2 props: `firstName`, `lastName` and `age`
- The two props should be rendered in the component

App.jsx

```jsx
import UserProfile from "./UserProfile";

function App() {
  return <UserProfile firstName="Max" lastName="Mustermann" age={25} />;
}
```

Questions: Components

What do components return in React?

Components in React return JSX (Javascript XML)

What happens when a component passes an updated "prop" to a child component?

The child component is re-rendered to process the updated prop.

What is the virtual DOM in React basically like?

The virtual DOM is an in-memory representation of the DOM in the browser and can be updated internally when changes are made to React in order to subsequently update the "correct" DOM.

Hooks

Hooks in React

Hooks are a feature of React that allows functionality to be shared and reused. They are an exclusive feature of functional components.

- Hooks consist of functions/hooks provided by React
  - useState
  - useEffect
  - useContext
  - useMemo
  - etc.

Start with the prefix ***use***

useState

This hook makes it possible to manage the status of a component in a function component.

- Takes initial value as state
- Returns array with current state and setter function

UseStateExample.jsx

```jsx
function CountTracker() {
  // Erstelle einen stateful Wert und eine Funktion zum Aktualisieren dieses Werts
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>The count is {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}
```

useEffect

This hook makes it possible to execute page effects after rendering a component.

- First argument as callback method
  - Executed when the component is re-rendered
- Second argument List of dependencies
  - Determines when callback function is executed
    - With update from State

UseEffectExample.jsx

```jsx
function CountTracker() {
  const [count, setCount] = useState(0)

  useEffect(() => {
    console.log(`The count is now ${count}`)
  }, [count])

  return (
    <div>
      <p>The count is {count}</p>
      <button onClick={() =>
        setCount(count + 1)
      }>Increment</button>
      <button onClick={() =>
        setCount(count - 1)
      }>Decrement</button>
    </div>
  )
}
```

useEffect

- Executed after rendering
- Side effects can be adjusted in the `return`

In this example, the useEffect hook is only executed after the initial rendering

UseEffectExample.jsx

```jsx
function IntervalTracker() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setCount(count + 1);
    }, 1000);
    return () => {
      clearInterval(interval);
    };
  }, []);

  return <p>The count is {count}</p>;
}
```

useMemo

The useMemo hook in React makes it possible to save a value that is only recalculated when certain dependencies change.

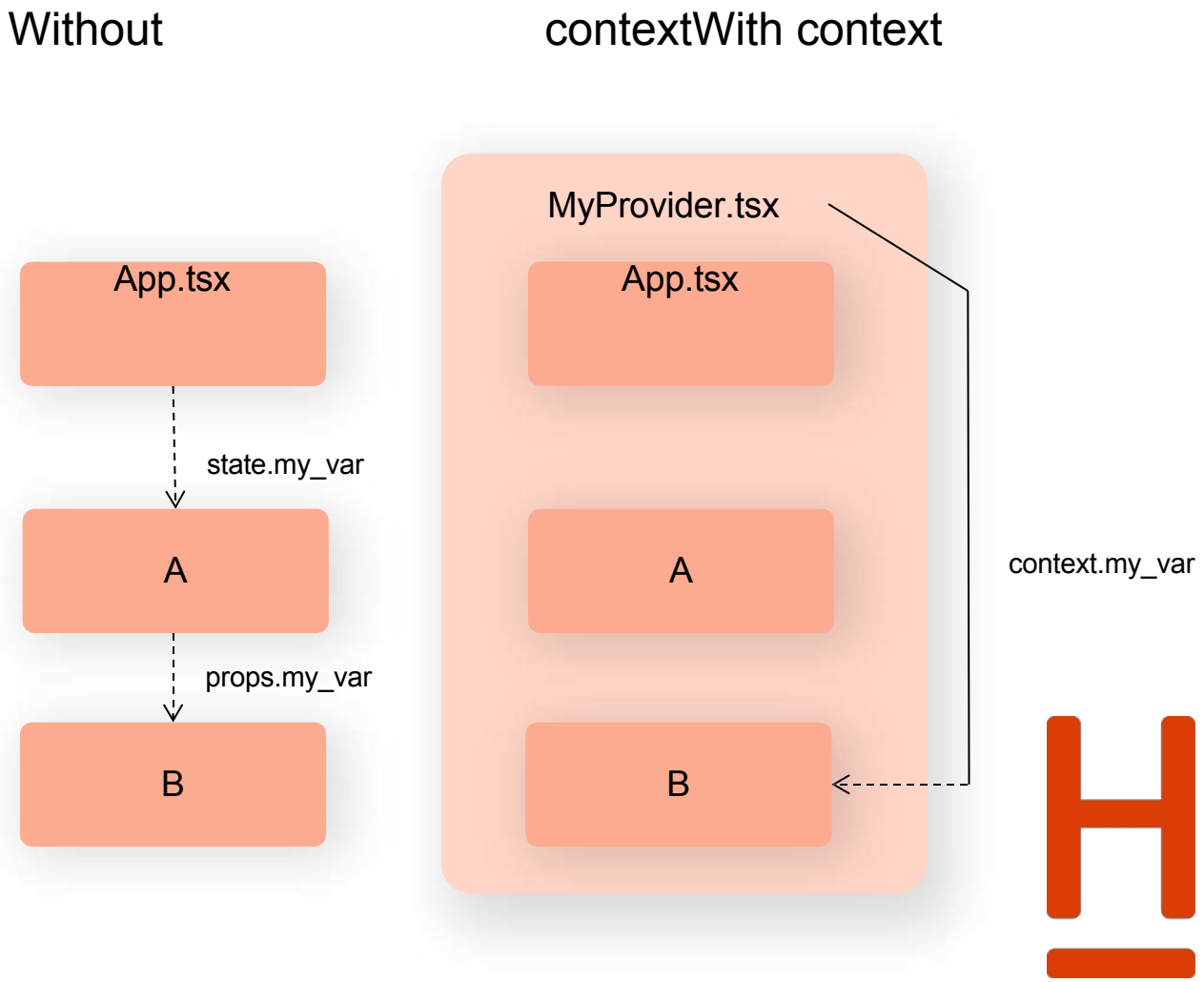
- Useful for expensive calculations
- Similar to useEffect: callback function and dependency array as parameters

UseMemoExample.jsx

```jsx
function MyComponent(props) {
  const expensiveValue = useMemo(() => {
    // Do some expensive calculation here
    return result;
  }, [props.x, props.y]);

  return <p>The expensive value is {expensiveValue}</p>;
}
```

useContext

- Mechanism to store and share values at component level
- values do not have to be passed through the entire component tree.
  becom
  e



useContext: When to use

- Sharing "global" data
  - e.g. current language, current user, themes etc.
- Can also be used on subtrees
- Should not be used excessively
  - Makes it difficult to reuse the components

UseContextExample.jsx

```jsx
const ThemeContext = React.createContext('light');

const App = () => {
  return (
    <ThemeContext.Provider value="dark">
      <Toolbar />
    </ThemeContext.Provider>
  );
}

// theme muss nicht über props weitergereicht werden
function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

const ThemedButton = () => {
  const theme = useContext(ThemeContext);
  return <Button theme={theme} />;
}
```

Joint exercise: useContext

App.jsx

```
import { UserContext } from './UserContext'
import UserProfile from './UserProfile'

const App = () => {
  return (
    <UserContext>
      <div>
        <h1>User Profile App</div>
        <UserProfile />
      </div>
    </UserContext>
  );
};

export default App;
```

Implement your own hooks

Function with "use" prefix using the React hooks

useMousePosition.jsx

```jsx
function useMousePosition() {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  useEffect(() => {
    function updateMousePosition(event) {
      setPosition({ x: event.clientX, y: event.clientY });
    }

    window.addEventListener('mousemove', updateMousePosition);

    return () => {
      window.removeEventListener('mousemove', updateMousePosition);
    };
  }, []);

  return position;
}
```

MyComponent.jsx

```jsx
function MyComponent() {
  const position = useMousePosition();
  return (
    <p>
      The mouse position is
      {position.x}, {position.y}
    </p>
  );
}
```

Questions: Hooks

How are hooks easy to recognize?

Hooks are written with the prefix "**use**" (**useState**, useEffect etc.)

When does it make sense to use the useMemo hook?

The useMemo hook is only re-executed when props from the dependency array change. The hook is therefore ideal for optimizing computationally intensive operations.

What happens if an empty dependency array is passed to the useEffect hook?

The hook is then only executed once after the initial rendering.

What is the fundamental advantage of useContext?

useContext allows you to share state across components. This means that props do not have to be passed through an entire subtree.

Exercise: Own hook "useCounter"

Write your own hook with the name:
useCounter
It returns 3 functions and the counter reading:

- count: the current meter reading
- increment: Increments the counter by +1
- decrement: Counts down the counter by -1
- reset: resets the counter to the initial value

In addition, the useCounter hook should be able to return an initial value.

App.jsx

```jsx
import useCounter from './useCounter';

const App = () => {
  const { count, increment, decrement, reset } = useCounter(0);

  return (
    <div>
      <h1>Counter App</h1>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
};

export default App;
```

HTTP and the Fetch API

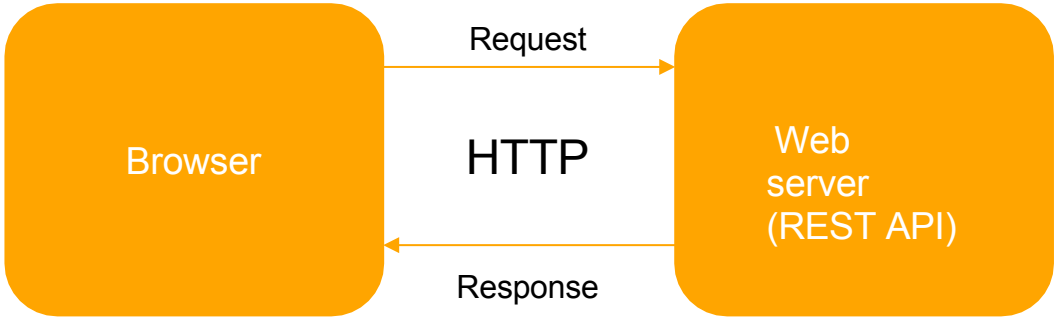
What is HTTP?

HTTP stands for **Hypertext Transfer Protocol**.
It is a protocol for transmitting data via the Internet.

HTTP defines how requests and responses are structured between web browsers and servers.

Functions of HTTP requests:
Sent from the client to the server to request resources.

Responses:
Sent from the server to the client to transfer requested resources.



HTTP methods

HTTP methods define the type of interaction between clients and servers.
Each method has a specific purpose and contributes to the structuring and organization of data transmission on the Internet.

The most important methods:
GET ("Please read this resource and give it to me, Server!"):
Requesting data from a specific resource.

POST ("Please create a new resource, Server!"):
Create a resource on the server.

PUT ("Please update this resource, Server!"):
Updating a resource on the server.

DELETE ("Please delete this resource, Server!"):
Delete a resource.

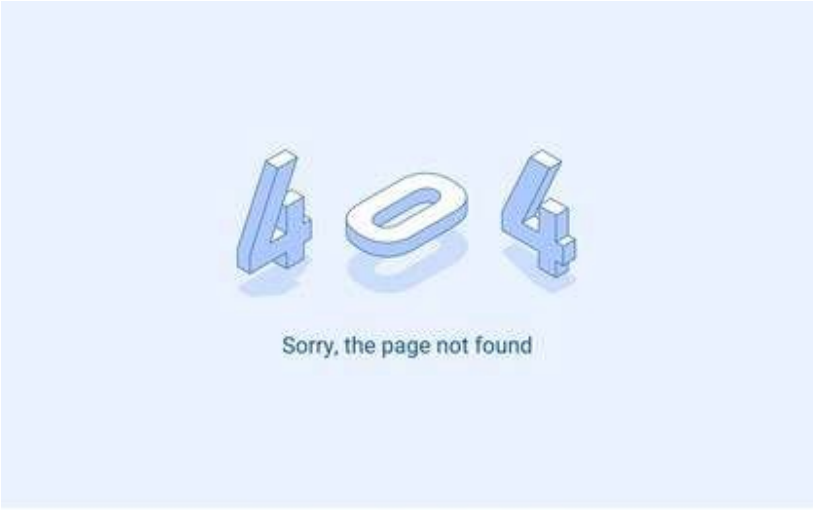
This is also referred to as the CRUD scheme: Create (POST), Read (GET), Update (PUT), Delete (DELETE)

HTTP status codes

HTTP status codes are communication signals from the server to the client that indicate the success or failure of an HTTP request.

1xx: Information - The request is being processed.
2xx: Successful request - The request was successfully understood, accepted and processed.
3xx: Redirection - Further actions are required to complete the request.
4xx: Client error - The request contains an error or cannot be processed.
5xx: Server error - The server could not fulfill the request.

Examples
200 OK: Successful request.
404 Not Found: The requested resource was not found.
500 Internal Server Error: A server error has occurred.



Fetch API

The Fetch API is a modern API for sending HTTP requests and receiving responses in JavaScript.

JS fetch.js

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => {
    // Verarbeitung der erhaltenen Daten
  })
  .catch(error => {
    // Behandlung von Fehlern
  });
```

Calling the Fetch API returns a **Promise**

Promises are a pattern for processing asynchronous operations. They represent the success or failure of an asynchronous task and enable it to be processed.

Exercise: Fetch API and hooks

Implements a `useFetch` hook that allows data to be retrieved from a URL using `fetch` and displayed in a component.

Should accept a URL as an argument and retrieve data using `fetch`

Should return an object with three values:
`data`: the retrieved data
`loading`: the current charge level
`error`: default `null`, error object if errors occur

When the loading process is running, `loading` should be `true` and `data` and `error` `null`

If the loading process has been successfully completed, `loading` should be `false` and `data` should contain the retrieved data

If an error occurs during the loading process, `loading` should be `false` and `error should` contain the error
Optional: Typing with Typescript

UserList.jsx

```jsx
function UserList() {
 const baseUrl = "https://dummyjson.com/";
 const { data, loading, error } = useFetch(`${baseUrl}users`);

 if (loading) {
   return <p>Ladevorgang läuft...</p>;
 }

 if (error) {
   return <p>Es ist ein Fehler aufgetreten: {error.message}</p>;
 }

 if (!data) {
   return null;
 }

 return (
   <ul>
     {data.users.map((user) => (
       <li key={user.id}>{user.firstName}</li>
     ))}
   </ul>
 );
}
```

Frontend patterns

MVC etc.

Patterns in the FE

In complex applications, organizing the code can be a challenge. Challenges:
Data flow: Efficient and clearly structured management of the data flow is crucial.

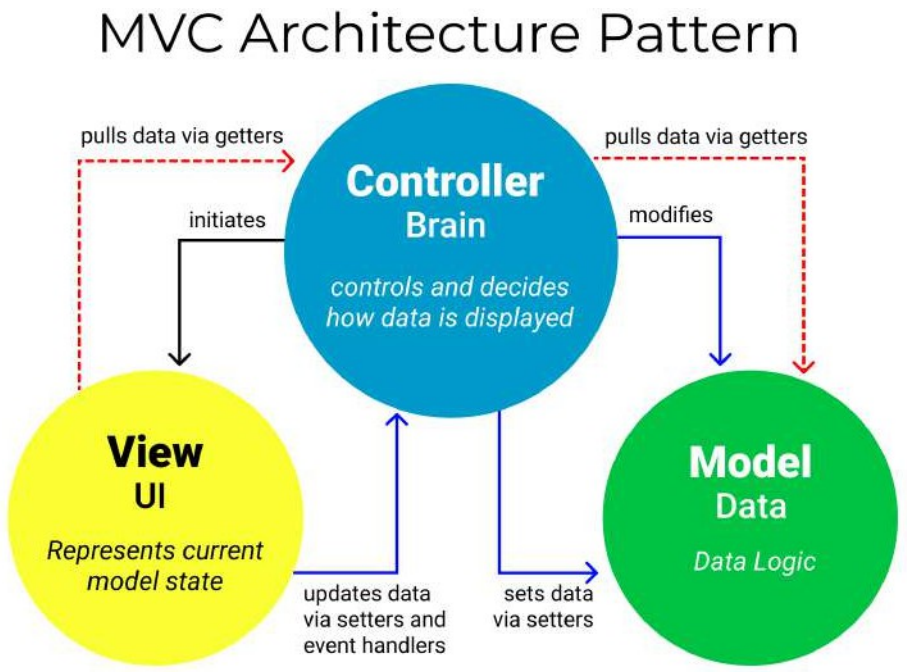
Maintainability: Clear structures enable the maintainability and scalability of applications.

MVC - Model View Controller

Model: Data and business logic.

View: Display of data for users.

Controller: Accepts user interaction and updates the model and view accordingly.



https://www.freecodecamp.org/news/content/images/2021/04/MVC3.png

State management

State management

What happens when the application gets bigger? Is it then sufficient to use useState and props?

No, above a certain application size the application becomes unmaintainable and is too complicated.

Question: What is the problem with the following code example?

App.jsx

```jsx
import { useState } from 'react';
import Cart from './Cart';
import ProductList from './ProductList';

const App = () => {
  const [cart, setCart] = useState([]);
  const [products, setProducts] = useState([...]); // Produktliste aus API-Aufruf

  const addToCart = (product) => {
    setCart([...cart, product]);
  };

  return (
    <div>
      <h1>E-Commerce App</h1>
      <ProductList products={products} addToCart={addToCart} />
      <Cart cart={cart} />
    </div>
  );
};
```

State management

As the size of the application grows, it makes sense to consider advanced state management.

- Refers to the use of libraries such as Redux or Mobx
- Managing the status in central stores
- Decoupling of UI and data
- Is not always necessary, small applications can also manage without libraries
  - useState in conjunction with your own hooks

mobx

The MyStore `class` represents a Mobx Store that can hold and change observable data.

`@observable` decorator marks observable variables `increment` and `decrement` are annotated with `@action` so that changes are communicated to the component

JS MyStore.js

```
import { observable, action } from 'mobx';

class MyStore {
  @observable count = 0;

  @action
  increment() {
    this.count++;
  }
  @action
  decrement() {
    this.count--;
  }
}

const store = new MyStore();
```

mobx

Use of the `useObserver hook` to monitor the component and automatically re-render it if the value of the count property in the store changes.

The component displays the current value of the count property and provides buttons to increase or decrease the value by calling the increment and decrement methods of the store.

MyComponent.jsx

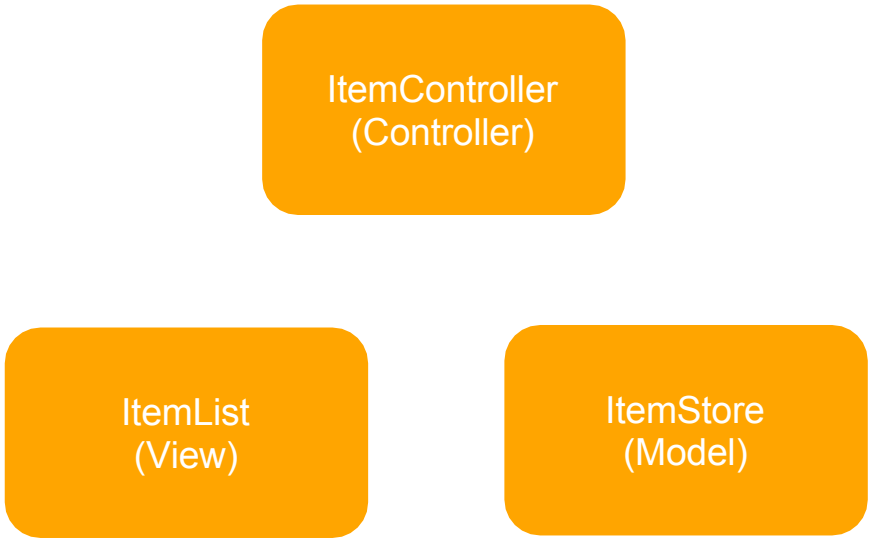
```jsx
import { useObserver } from 'mobx-react-lite';

const MyComponent = () => {
  return useObserver(() => (
    <>
      <p>The count is {store.count}</p>
      <button onClick={store.increment}>
        Increment
      </button>
      <button onClick={store.decrement}>
        Decrement
      </button>
    </>
  ));
};
```

Questions: State management

TODO

TODO

Joint exercise: MVC with React and mobx



SPA vs SSR

Architectures for web applications

What is a software architecture

The organization of software elements and their interactions in order to provide certain functions.

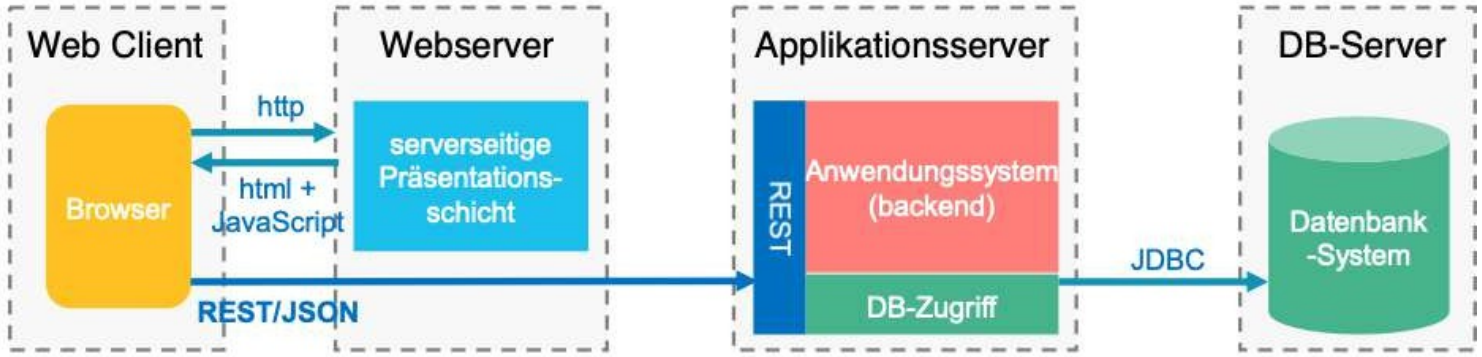
- **Structured development:** Facilitates the structured development of large and complex applications.
- **Maintainability:** Enables easier maintenance and updating of the code.
- **Scalability:** Allows the application to grow without massive restructuring.

Architectures for web applications

Single Page Applications

Single page applications are virtually an HTML page that is dynamically updated with Javascript.

Main components in a SPA
View: UI components that are presented to the user.
Router: Manages the navigation within the application.
State Management: Saves and updates the application status.
API calls: Communication with the server or other external resources.

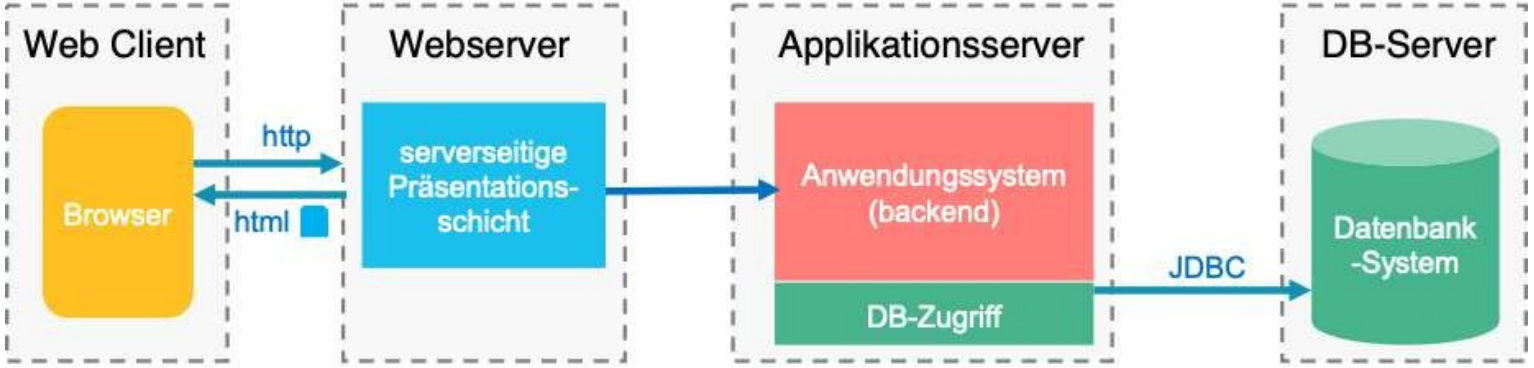


Architectures for web applications

Server-side rendering

The server renders the HTML page and sends it to the client.

Main components in an SSR system
Server: Renders HTML on request and sends it to the client.
Client: Takes control after initial rendering for dynamic updates.
Router: Manages the navigation, both on the server side and on the client side.
State management: Can exist on the server or the client, depending on the requirements.



Architectures for web applications

Comparison SPA vs. SSR

Advantages SPA:
- Fast, seamless user experience
- Low server overhead after initialization

Advantages of SSR:
- Fast initial page load time
- Also works with lightweight clients
- Better SEO

Disadvantages SPA:
- Slower initial page load time
- SEO can be challenging

Disadvantages of SSR:
- More complex implementation
- Higher server costs
- Not responsive

Questions: Architectures for web applications

What is the difference between server-side rendering (SSR) and single-page applications (SPA)?

SSR applications generate finished HTML which is sent to the client and SPAs "render" the application with Javascript in the client.

Routing

Routing with react-router

Routing takes care of navigation between different views in web applications. Allows users to navigate

through the application and display different content.

React Router is a library for managing URLs
- Renders UIs based on current URL
- Enables navigation between URLs

Example shows a router component
- Link component Works like <a/> tag
- Route component controls which component is rendered on which URL

router.jsx

```jsx
function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Route exact path="/" component={Home} />
      <Route path="/about" component={About} />
    </BrowserRouter>
  );
}
```

```
$ npm install react-router-dom
```