

Web Programmierung

Public Version 2024-25 – Version 1

Welcome

Dozent



Jens Reiner

Solution Expert PlanB. GmbH

DPF Software Engineering & Architecture

Bachelor - Software Engineering Universität Ulm

Die letzten 3 Jahre Web Programmierung unterrichtet
Mehrere Jahre Erfahrung in der modernen Web Entwicklung
<https://www.linkedin.com/in/jens-reiner/>

Welcome

Disclaimer

- Anbei eine öffentliche Version der Folien meiner Vorlesung an der DHBW im Fach Web Programmierung
- Die Vorlesung wurde im Jahr 2024 / 2025 im 3. Semester Wirtschaftsinformatik Bachelor gehalten
- Die Public Version beinhaltet alle Unterlagen aber ohne DHBW Bezug oder Prüfungsleistung Informationen etc.
- Bitte die License beachten (Siehe Footer oder nächste Slide)

Vorlesung

Verwendung der Unterlagen

- Alle Unterlagen der Vorlesung sind unter Creative Commons BY-NC-SA 4.0 lizenziert
- Siehe:
<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.de>

Sie dürfen:

Teilen — das Material in jedwedem Format oder Medium vervielfältigen und weiterverbreiten

Bearbeiten — das Material remixen, verändern und darauf aufbauen

Der Lizenzgeber kann diese Freiheiten nicht widerrufen solange Sie sich an die Lizenzbedingungen halten.

Unter folgenden Bedingungen:



Namensnennung — Sie müssen angemessene Urheber- und Rechteangaben machen, einen Link zur Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden. Diese Angaben dürfen in jeder angemessenen Art und Weise gemacht werden, allerdings nicht so, dass der Eindruck entsteht, der Lizenzgeber unterstützte gerade Sie oder Ihre Nutzung besonders.



Nicht kommerziell — Sie dürfen das Material nicht für kommerzielle Zwecke nutzen.



Weitergabe unter gleichen Bedingungen — Wenn Sie das Material remixen, verändern oder anderweitig direkt darauf aufbauen, dürfen Sie Ihre Beiträge nur unter derselben Lizenz wie das Original verbreiten.

Keine weiteren Einschränkungen — Sie dürfen keine zusätzlichen Klauseln oder technische Verfahren einsetzen, die anderen rechtlich irgendetwas untersagen, was die Lizenz erlaubt.

Vorlesung

Agenda

- Web Programmierung im Alltag
- Programmieren mit JavaScript
- HTTP-Protokoll, REST und Client-Server Architekturen

Vorlesung

Agenda

- Entwicklung von einer Server Schnittstelle (Backend)
- Entwicklung von einer Client Applikation (Frontend)
- Erweiterte Konzepte sowie Tools & Werkzeuge

Vorlesung

Ziel der Veranstaltung

Am Ende können Sie eine eigene Schnittstelle programmieren und Daten über JSON austauschen. Darüber hinaus können Sie eine Webseite mit einem JavaScript Framework implementieren und Daten zwischen Client und Server senden.

Sie lernen Werkzeuge für die Entwicklung kennen und können Software Anforderungen mit einer Web Technologie umsetzen. Außerdem lernen Sie Code zu lesen und zu verstehen.

Vorlesung

Hinweis zu Quellcode

- Alle Quellcode Beispiele wurden mit Node Version 20.11.1 entwickelt und getestet
 - Eine genaue Installation Anleitung erfolgt im jeweiligen Kapitel
- Alle Quellcode Beispiele sollten OS unabhängig funktionieren und auch ältere / neuere Node Versionen unterstützen
 - Ausnahme sind selbst erstelle Distros
 - Node >= 18 ist für ein paar Features erforderlich (e.g. Fetch API)
- Alle Quellcode Beispiele finden Sie auch auf Github
 - Auf den Folien sind diese als Screenshot hinterlegt
 - Auf Quellcode Folien finden Sie oben rechts ein Link zu der Datei auf Github

Vorlesung

Software zum installieren

- NodeJS (Version 20.n LTS)
 - Empfehlung: Installieren Sie NodeJS über einen Package Manager
<https://nodejs.org/en/download/package-manager>
 - Alternativ: <https://nodejs.org/en/download/prebuilt-installer>
- API Client
 - In der Vorlesung verwenden wir Bruno <https://www.usebruno.com/downloads>
 - Alternativ: Postman <https://www.postman.com>

Vorlesung

Literatur & Unterlagen

- Exploring JavaScript
 - Sie können das Gesamte Buch Online Kostenlos lesen
 - <https://exploringjs.com/js/index.html>
 - ISBN: 1091210098 (Print ist noch die 2019 Version, Digital gibt es die 2024 Version)
- Mastering API Architecture: Design, Operate, and Evolve API-Based Systems
 - ISBN: 1492090638
- Restful Web API Patterns and Practices Cookbook: Connecting and Orchestrating Microservices and Distributed Data
 - ISBN: 1098106741

Vorlesung

Literatur & Unterlagen

- Github Repository ergänzend zu den Folien
 - <https://github.com/wasdJens/dhbw-webprogrammierung-wwi>
- Github Repository für die Entwicklung
 - <https://github.com/wasdJens/dhbw-webprogrammierung-wwi-vorlesung>
- In Eigener Sache: Dont Panic Web Edition
 - <https://github.com/wasdJens/dont-panic-web-edition>
 - Mein persönliches Handbuch um die Welt der Web Entwicklung näher zu bringen
 - Wird ständig mit neuen Themen erweitert (Aktuell in Arbeit: In Depth Angular) und ist für immer kostenlos

Web Programmierung

Warum eigentlich?

Web Programmierung

Historie



[9.10.98] Immer mehr Bücher werden über das Internet gekauft, 1998 schon für etwa 70 Mio DM. Der Onlinehandel habe, laut Börsenverein des deutschen Buchhandels, aber eine andere Kundenstruktur und werde dem traditionellen Handel nur beschränkt schaden.

.Translate Tweet



9:53 PM - 9 Oct 2018

Web Programmierung

Frage

- Was für Anwendungen nutzen Sie täglich?
 - Bspw. Smartphone App, Webseiten etc.

Web Programmierung

Im Alltag

- Fast alles, was sie täglich an Digitalen Produkten nutzen, hat einen Bezug zur Web Programmierung
- Online Shopping, Social Media, Suchmaschinen, Online Banking, Nachrichten, Streaming Dienste, Mobile Apps, IoT Geräte, Lernplattformen etc.

Web Programmierung

Im Alltag

- Web Anwendungen sind nicht mehr nur beschränkt auf Desktop Computern mit Webbrowser
- Auch Smartphones, Tablets, Smart TVs und co nutzen Web Anwendungen für die Bereitstellung von Digitalen Produkten
- Netflix, Instagram etc.

Web Programmierung

Im Alltag

- Immer mehr Applikationen werden als Web Anwendungen entwickelt
- Gerade Cloud Migrationen von Legacy Software in die Cloud ist häufig damit verbunden, alte Desktop Applikationen als Web Anwendungen neu zu entwickeln
- Figma, Messenger, Teams, Office 365, VSCode, SAP, Salesforce

Web Programmierung

Entstehung des WWW

ca. 1960	1989	1991-1993	1993	1995
ARPANET Paketvermittlungsnetz des US Verteidigungsministeriums. Vorläufer des Internets	WWW Tim-Berners Lee entwickelt am CERN das Konzept des World-Wide-Web als System von Hypertext-Dokumenten	HTTP & HTML Protokolle des WWW werden formalisiert und standardisiert	Mosaic Browser Erster Grafischer Webbrowser, breitere Öffentlichkeit kann nun das WWW nutzen	JavaScript Start der interaktiven Scriptsprache

Web Programmierung

Entstehung des WWW

1990er	ca. 2005	ca. 2006	ca. 2007	2020er
E-Commerce Start des Onlinehandels und Kommerzialisierung des WWW bspw. durch Amazon und Ebay	AJAX Interaktive Webanwendungen ohne Neuladen (Web 2.0) wie bspw. Gmail, Maps oder Videostreaming	Cloud Computing Rechenleistung wird genauer abgerechnet und nur nach Nutzung bspw. Amazon Web Services	Mobile Endgeräte Smartphones wie bspw. das iPhone gewinnen an Beliebtheit und öffnen Zugang um WWW	Web 3.0 Dezentralisierung, Datensouveränität und bspw. Blockchain ermöglichen neue Interaktionsformen

Web Programmierung

Anwendungen in der Industrie

- Technologien aus der Web Programmierung finden vor allem in der Entwicklung von modernen Schnittstellen statt
- Webtechnologien sind sehr flexibel und können auf fast jedem Endgerät ausgeführt werden
 - Bspw. In Car Infotainment
- Webtechnologien sind privat und vergleichsweise günstig in der Implementierung
 - Bspw. gibt es mehr Experten/innen die sich mit Webtechnologien auskennen als mit anderen Technologien

Web Programmierung

Webtechnologien im Betrieb

- Wie nutzt ihr Betrieb heute Webtechnologien?
 - Bspw. Schnittstellen, Anwendungen, Cloud Technologien, etc.

Web Programmierung

Bedeutung

- Nahezu jede Anwendung, die Sie täglich nutzen, ist mit Webtechnologien umgesetzt
- Fast jedes Unternehmen arbeitet heutzutage mit Schnittstellen und tauscht Daten über HTTP miteinander aus
- Das WWW als solches ist im heutigen Alltag nicht mehr Wegzudenken

Web Programmierung

Bedeutung

- Und aus diesem Grund möchte ich Ihnen auch die Welt der Web Programmierung näher bringen
- Technologien, Programmiersprachen, Software Konzepte / Architekturen etc. All das schauen wir uns im Rahmen der Vorlesung einmal genauer an
- Und am Ende können Sie selbst einschätzen, wenn Sie ein neues Software Produkt umsetzen müssen, welche Technologien Sie verwenden

JavaScript

Eine Einführung in die Programmiersprache

JavaScript

Lernziele

- Was ist JavaScript
- Variablen, Objekte und Datentypen in JavaScript
- Funktionen und Arrays in JavaScript
- Einfache Programme mit JavaScript entwerfen und implementieren

JavaScript

Warum JavaScript?

- 98,8% aller Webseiten verwenden JavaScript
 - <https://w3techs.com/technologies/details/cp-javascript>
- Plattformunabhängig – Mit JavaScript können Sie für Desktop, Browser und Mobil Geräte Software entwickeln
 - Bspw. PWAs, Electron, React Native, Ionic, NodeJS
- Client und Server Applikationen können dieselbe Programmiersprache verwenden

JavaScript

Warum JavaScript?

- Die Entwickler Community ist riesig und fast alles an Informationen können Sie frei im Web abrufen
- Sie können JavaScript komplett frei verwenden (Open Source License)
 - JavaScript gehört keiner Organisation stattdessen gibt es das TC39 Komitee die den Standard der Sprache immer weiterentwickeln
 - <https://tc39.es/de/>
 - Der Name JavaScript ist aber leider weiterhin eine Trademark von Oracle
- *Persönliche Meinung: Es ist eine sehr einfache Programmiersprache zum lernen*



JavaScript Developers



JavaScript

Woher kommt JavaScript?

- JavaScript ist eine sehr junge Programmiersprache und wurde 1995 entworfen, um Programme für Webseiten zu schreiben
- JavaScript ist geprägt von Konzepten aus anderen Programmiersprachen die zu dieser Zeit beliebt waren
 - Bspw. Die Syntax kommt primär von Java
- Heute ist JavaScript Teil von jedem modernen Browser und Sie haben vermutlich täglich mehrfach Berührungspunkte mit JavaScript

JavaScript

Eigenschaften von JavaScript

- JavaScript gibt es in unterschiedlichen Versionen und nicht jeder Browser kann jedes Feature
- Die Definition von JavaScript wird in ECMAScript Versionen festgehalten
 - ECMAScript beschreibt wie JavaScript funktioniert
- Seit ECMAScript 6 wurden fast alle quirks die JavaScript anfangs unbeliebt gemacht hat ausgebessert

JavaScript

Was ist JavaScript?

- Eine Just-In-Time Programmiersprache mit first-class functions support
- JavaScript ist dynamisch typisiert, Prototypen basierend und kann sowohl Objekt Orientierte, imperative als auch deklarative Programmierstile abbilden
- JavaScript wird als JavaScript ausgeliefert. JavaScript muss *nicht kompiliert** werden und es gibt auch kein Binäres Source Code Format
 - Allerdings gibt es ein Proposal das zu ändern <https://github.com/tc39/proposal-binary-ast> und auch Anbieter wie bspw. Cloudflare haben mit solchen Formaten bereits Experimente gemacht

JavaScript

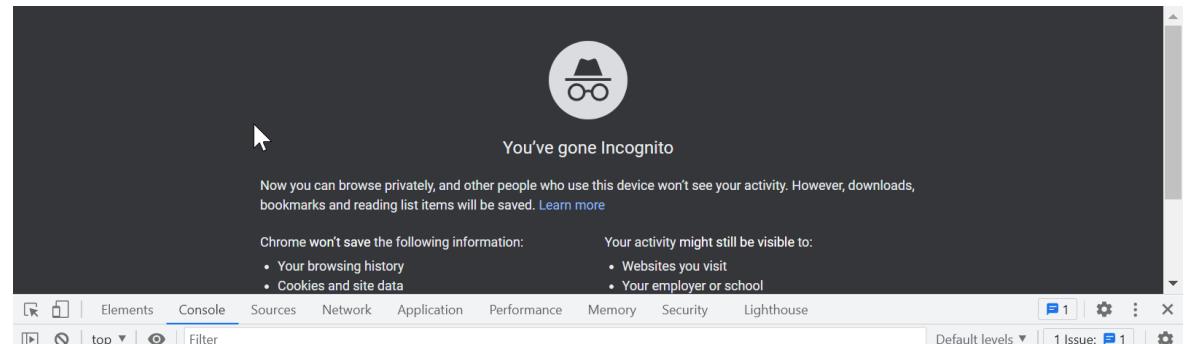
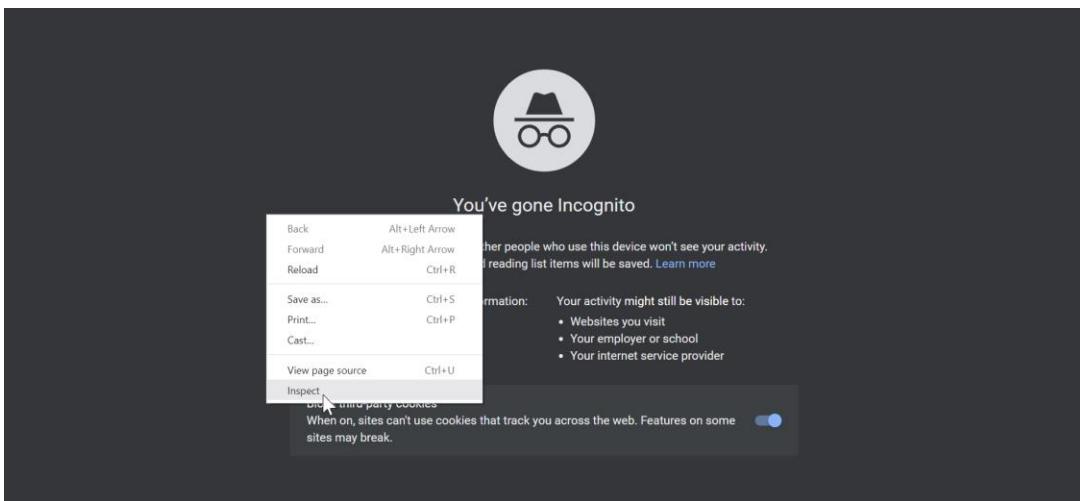
Warum JavaScript in der Vorlesung?

- Für das Entwickeln von modernen Webseiten benötigen wir JavaScript um Inhalte Dynamisch darzustellen und Server Interaktionen abzubilden
- Mit der Hilfe von NodeJS können wir auch unserer Server Applikationen in JavaScript entwickeln
 - Sie haben also keinen Kontext Wechsel im Bezug auf die Programmiersprache
- Es gibt jede Menge Tools für JavaScript und Sie können im Web alles über die Sprache nachlesen / lernen

JavaScript

Entwicklung

- Sie können jederzeit JavaScript Code Snippets in ihrer Browser Console ausführen



JavaScript

Entwicklung in der Vorlesung

- Für eine bessere Developer Experience werden wir nachfolgend alle Beispiele nicht direkt im Browser implementieren, sondern greifen auf das Programm NodeJS zurück
- Zusätzlich verwenden wir Visual Studio Code für das Editieren der einzelnen Dateien und Ausführung von JavaScript Programmen
 - Sie können sich meine Beispiel Umgebung hier herunterladen:
<https://github.com/wasdJens/dhbw-webprogrammierung-wwi-vorlesung>

JavaScript

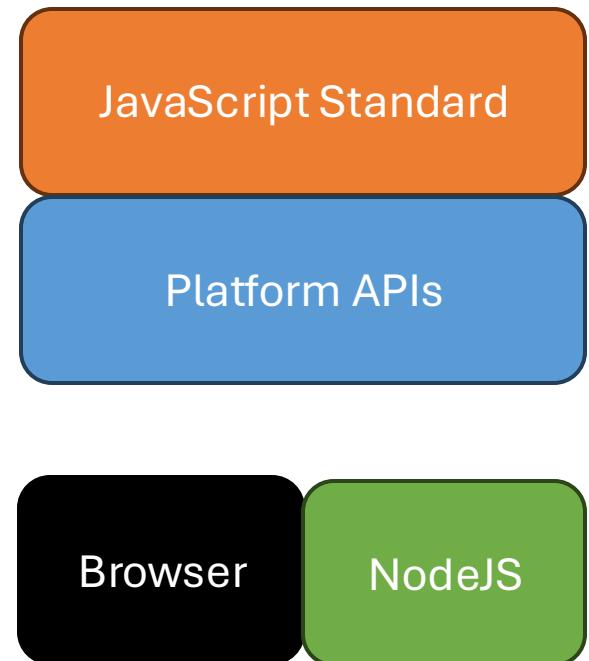
Entwicklung mit NodeJS

- NodeJS ist eine JavaScript Runtime die unabhängig von einem Webbrowser JavaScript Programme ausführen kann
- NodeJS baut auf die von Google entwickelte V8 JavaScript Engine auf
- NodeJS ist open-source und cross-platform
- Mit NodeJS können Server, Web Apps, Command Line Tools und Skripte entwickelt werden

JavaScript

Eigenschaften NodeJS

- NodeJS unterstützt unterschiedliche ECMAScript Versionen (Je nach Node Version) und ist nicht an Browser spezifische Features gebunden
- Im Gegenteil mit NodeJS haben sie eine andere Plattform API und können bspw. Programme entwickeln die auf das File System zugreifen
- Beispiel:
 - Browser APIs können Sie auf Maus Klicks reagieren
 - Node APIs können Sie mit dem Netzwerk interagieren



JavaScript

Eigenschaften NodeJS

- Mehr über die Eigenschaften und Vorteile von NodeJS im späteren NodeJS Kapitel wenn wir uns die Server Implementierungen genauer anschauen

JavaScript

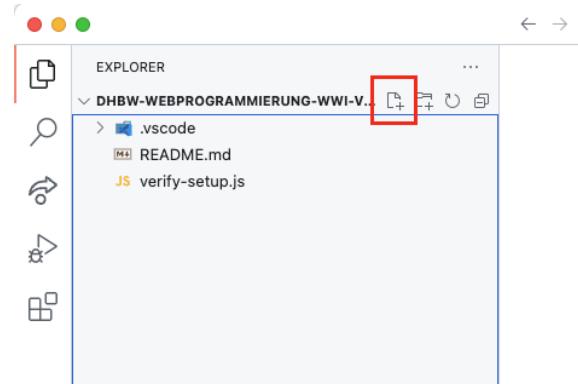
Hinweise & Organisatorisches

- Die nachfolgenden Folien setzen voraus, dass NodeJS sowie Visual Studio Code installiert sind
- Sie können das Referenz Projekt während der Vorlesung als Basis für ihre Aufgaben nutzen und beliebige Ordner Strukturen anlegen
- Alle Beispiele finden Sie auch vollständig im allgemeinem Github Repo und sind in den Folien verlinkt
 - Link: <https://github.com/wasdJens/dhbw-webprogrammierung-wwi>

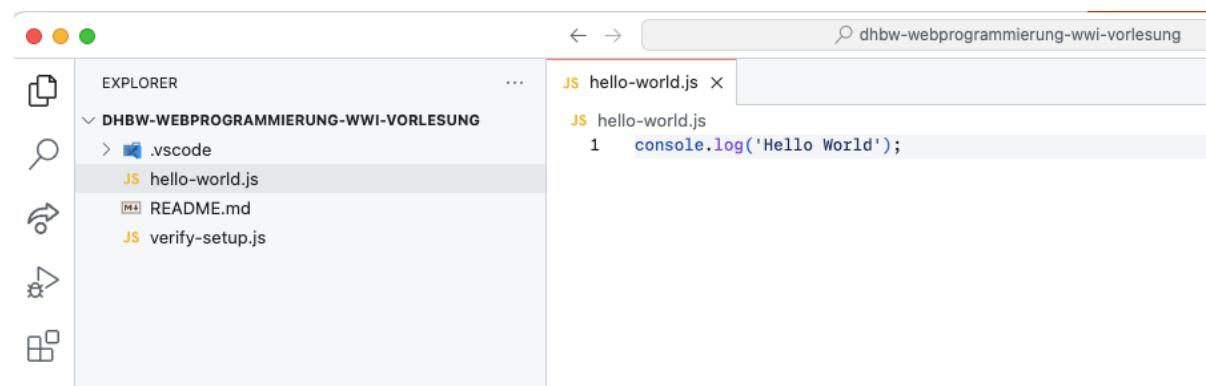
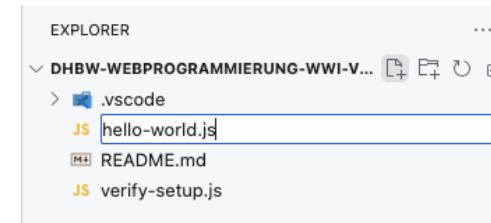
JavaScript

Das erste Programm

1. Erstellen Sie eine neue Datei „hello-world.js“
2. Öffnen Sie die Datei und fügen Sie folgenden Code hinzu:
 - `console.log('Hello World');`
3. Speichern Sie die Datei
 - Shortcut: (strg + s oder cmd + s)
 - In VSCode verschwindet der „Punkt“ im Dateinamen Tab



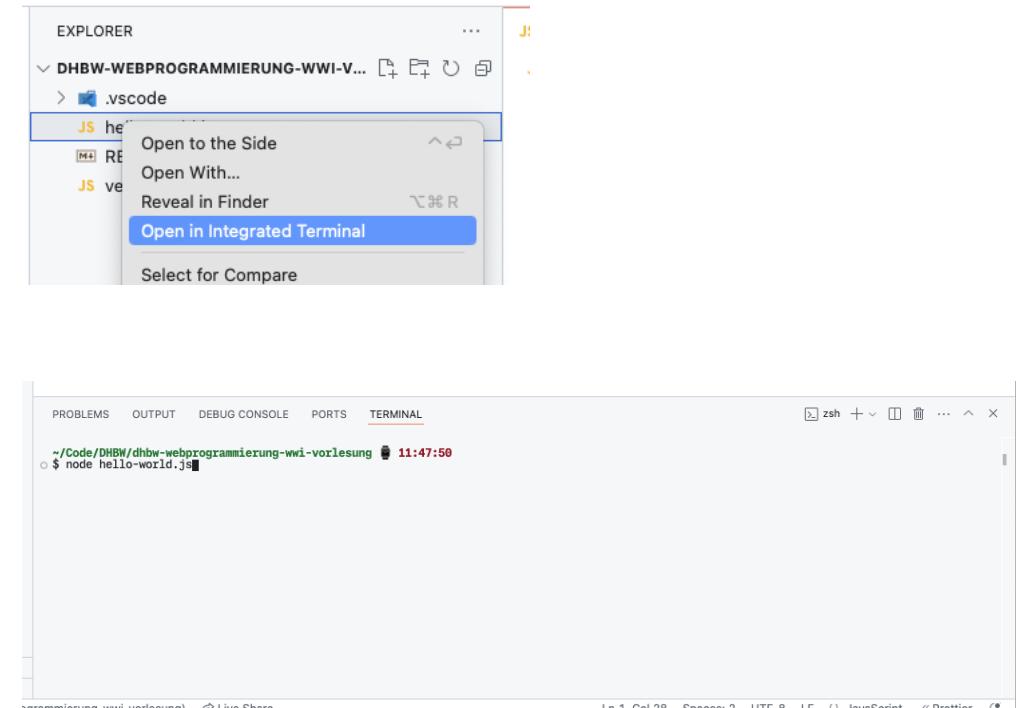
[Github](#)



JavaScript

Das erste Programm

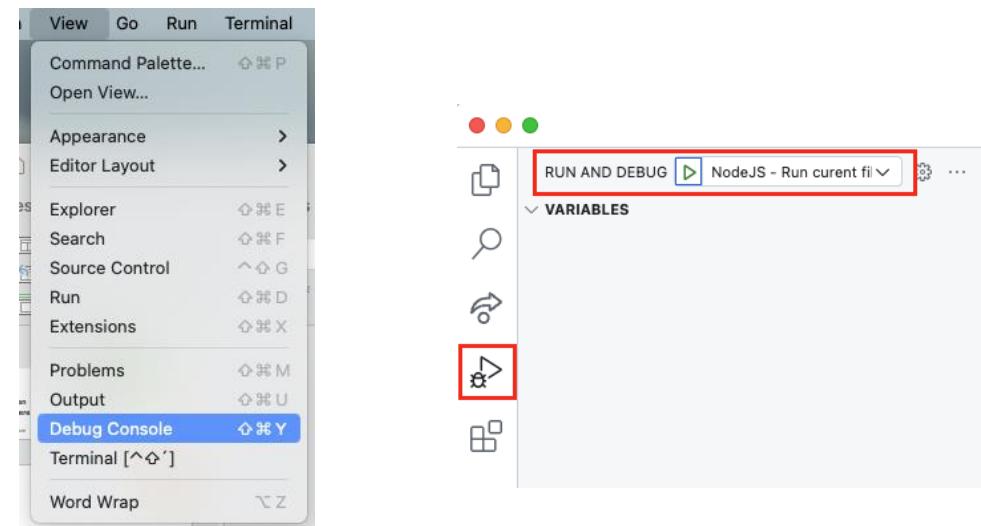
4. Machen Sie einen Rechtsklick auf die Datei und wählen Sie „Open in integrated Terminal“ aus
 - Hinweis: Je nach Systemsprache steht hier ggf. etwas anderes
5. Unterhalb von Ihrem Editor öffnet sich jetzt ein neues Fenster
6. Geben Sie den Befehl „node hello-world.js“ ein und klicken Sie Enter



JavaScript

Das erste Programm

- Alternativ gehen Sie auf „View“ und wählen „Debug Console“ aus
- Gehen Sie dann in VSCode auf „Run and Debug“ links im Editor und prüfen ob im Dropdown „NodeJS – Run current File“ selektiert ist
- Klicken Sie anschließend auf den grünen Play Button um das Programm auszuführen



JavaScript

Das erste Programm

The screenshot shows a Node.js debugging interface. On the left, there's a sidebar with icons for RUN AND DEBUG (highlighted with a red box), VARIABLES, WATCH, CALL STACK, and BREAKPOINTS. The main area displays a code editor with the file `hello-world.js` containing the code

```
JS hello-world.js
1 console.log('Hello World');
```

. Below the code editor is a DEBUG CONSOLE tab (also highlighted with a red box) which shows the output of the program:

```
/Users/jens/.nvm/versions/node/v20.11.1/bin/node ./hello-world.js
Hello World
```

. The line `Hello World` is highlighted with a red box. At the bottom of the interface, there's a status bar with the message "Please start a debug session to evaluate expressions".

JavaScript

Wichtige Commands

- Wenn Sie sich Daten in JavaScript ausgeben möchten können Sie immer den **console.log()** Befehl verwenden
- In den nachfolgenden Beispielen verwenden wir diesen Befehl um Daten auszugeben.
- Wichtige weitere Commands ergänzen wir immer nach jedem Kapitel
 - Am Ende bekommen Sie auch noch ein Cheat Sheet als Übersicht

JavaScript

Wichtige Commands

- Code Kommentare können Sie in JavaScript mit `//` für single line comments verwenden
- Alternativ `/* */` für multi block comments
- *Tipp: Kommentieren Sie immer das **why** und nicht das **what***

JavaScript

Wichtige Commands

- JavaScript braucht laut Definition kein ; am Ende einer Zeile
- Es ist aber sehr Empfehlenswert ; an das Ende von jeder Zeile zu machen um Fehler zu vermeiden
- Die meisten Style Formate erzwingen diese Regelung auch automatisch für Sie

JavaScript

Grundlagen: Variablen

- Eine Variable ist ein Container für einen Wert, wie z.B. das Ergebnis einer Rechenoperation oder ein Text, den der Benutzer eingegeben hat.
- Variablen sind *named storages* in JavaScript und können einen Wert speichern, auf den später zugegriffen werden kann.
- In JavaScript gibt es drei Keywords um eine Variable zu definieren:
 - **let, const und var***

JavaScript

Grundlagen: Variablen

- Eine Variable kann, muss aber nicht, einen Wert zugewiesen haben.
- Die Wertzuweisung in JavaScript funktioniert mit dem = Operator
- Variablen können beliebig benannt werden. Es gibt jedoch folgende Ausnahmen:
 - Variablennamen dürfen nicht mit einer Zahl beginnen
 - Variablen sind Case Sensitive
 - [Reserved Keywords](#)

JavaScript

Grundlagen: Variablen Beispiel



[Github](#)

```
1 const mwst = 0.19;
2 let priceWithMwst;
3
4 priceWithMwst = 210;
5
6 console.log(mwst, priceWithMwst);
```

JavaScript

Variablen: let, const oder var?

- Variablen die mit **let** deklariert werden, können ihren Wert ändern.
- Variablen die mit **const** deklariert werden, können ihren Wert nicht ändern.
- Historisch bedingt gibt es das Keyword **var** noch in JavaScript. Es sollte aber nicht mehr verwendet werden, da es zu unerwarteten Verhalten führen kann.
 - [A note about var](#)
- In dieser Vorlesung (und allgemein) sollten Sie deshalb bitte immer **let** oder **const** verwenden

JavaScript

Variablen: let oder const?

- Folgende Regel sollte man verwenden, um zu entscheiden, ob eine Variable mit **let** oder **const** deklariert werden soll
- **const** definiert eine unveränderliche Bindung, der Wert dieser Variable wird sich nicht mehr verändern
- **let** definiert eine lose Bindung, der Wert dieser Variable kann sich beliebig verändern.
- Regel: **const over let**. Let nur in Fällen wo man const nicht benutzen kann

JavaScript

Grundlagen: Datentypen

- JavaScript ist eine dynamisch typisierte Sprache
- Das bedeutet, dass Variablen nicht mit einem Datentyp deklariert werden müssen
- Der Datentyp wird automatisch zur Laufzeit von JavaScript ermittelt
- Jeder Wert in JavaScript ist immer ein Datentyp zugeordnet.
- **Der Datentyp kann sich aber zur Laufzeit ändern!**

JavaScript

Grundlagen: Dynamische Datentypen



[Github](#)

```
1 let userInput = "Foo";
2 console.log(typeof userInput);
3
4 userInput = 42;
5 console.log(typeof userInput);
6
```

JavaScript

Grundlagen: Dynamische Datentypen

- Was für Probleme können, auftreten bei dynamischen Datentypen?

JavaScript

Grundlagen: Datentyp – Number



[Github](#)

- In JavaScript können Zahlen Integer und Float Werte sein
- Es gibt keine spezielle Unterscheidung zwischen Integer und Float Werten
- Ein besonderer Zahlenwert ist der BigInt (Zahlen > $2^{53}-1$).
 - BigInts werden mit einem n am Ende der Zahl gekennzeichnet.

```
1 const number = 1337;
2 const pi = 3.14159265359;
3
4 const bigInt = 1234567890123456789012345678901234567890n;
5
6 console.log(number, pi, bigInt);
```

JavaScript

Grundlagen: Datentyp – String



[Github](#)

- Strings in JavaScript können entweder mit „“ oder ‘ ’, deklariert werden
- Strings beinhalten jede beliebige Zeichenkette, wie z.B. Text, Zahlen, Sonderzeichen etc.

```
1 const endpoint = 'https://api.example.com';
2 const version = "v1";
3 const emojiString = "🚀 😊";
4
5 console.log(endpoint, version, emojiString);
```

JavaScript

Grundlagen: Datentyp – String



[Github](#)

- Strings können mit + konkateniert werden
- Strings können auch als Template Strings definiert werden
- Ein Template String kann direkt Variablen enthalten, die mit \${} umschlossen werden
- Template Strings können auch Funktionen bzw. Operatoren ausführen
 - Mehr dazu später

```
1 const h = "Hello";
2 const w = "World";
3
4 console.log(h + " " + w);
5
6 const number = 1000;
7 const price = `The price of the new phone is ${number}`;
8
9 console.log(price);
```

JavaScript

Grundlagen: Datentyp – Boolean



[Github](#)

- Boolean Werte sind logische Datentypen die nur zwei Zustände annehmen können: **true** oder **false**

```
1 const confirmed = true;
2 const approved = false;
3
4 console.log(confirmed, approved);
```

JavaScript

Grundlagen: Datentyp – Null und Undefined

- In JavaScript gibt es zwei besondere Datentypen: **null** und **undefined**
- **null** ist ein expliziter Wert, der angibt, dass eine Variable keinen Wert hat
 - Gesprochen: nothing, empty oder wert unbekannt
- **Undefined** ist ein impliziter Wert, der angibt, dass eine Variable noch keinen Wert hat
 - Gesprochen: Dieser Variable wurde noch kein Wert zugewiesen

JavaScript

Grundlagen: Datentyp – Null und Undefined



[Github](#)

- Variablen die keine Wert Zuweisung haben sind in JavaScript immer **undefined**
- Wenn ich aber eine Variable deklariere und bspw. auf eine User Eingabe noch warte kann ich explizit **null** verwenden

```
1 let username = null; // Variable hat einen Wert aber der User hat noch keine Eingabe gemacht.  
2  
3 let password; // Variable hat keinen Wert und der User hat noch keine Eingabe gemacht:  
4  
5 console.log(username, password);
```

JavaScript

Grundlagen: Was ist eigentlich ein Datentyp?

- JavaScript unterscheidet zwischen Primitiven Werten und Objekten bei Datentypen
- Primitive Werte (Primitive Values) sind alle Elemente vom Typ *undefined*, *null*, *boolean*, *number*, *bigint*, *string* und *symbol*
 - (Symbol lassen wir in der Vorlesung weg)
- Alle anderen Werte sind Objekte
- Es gibt damit insgesamt 8 Datentypen in JavaScript

JavaScript

Grundlagen: Primitive Values vs Objects

- Primitive Values sind Bausteine für Daten in JavaScript
 - Vgl. mit Atomen bspw.
- Objekte hingegen sind eine Sammlung von beliebigen Daten

JavaScript

Grundlagen: Primitive Values vs Objects

- Primitive Values werden mit *pass by value* übergeben
 - Wenn eine Primitive Value als Parameter an eine Funktion übergeben wird, wird der Inhalt von JavaScript **kopiert**
- Objekte werden mit *pass by reference* übergeben
 - Wenn ein Objekt als Parameter an eine Funktion übergeben wird, wird die Speicheradresse (Pointer) vom Objekt weitergegeben

JavaScript

Grundlagen: Datentyp – Object

- Objekte sind der Datentyp, den wir am meisten in JavaScript finden
- Objekte können verschiedene Properties beinhalten, die wir frei definieren können.
- Jede Property wird mit einem **key** definiert und jedem key kann eine **value** zugeordnet werden
- Wenn wir eine Property lesen möchten, können wir mithilfe des **keys** den Wert ausgeben

JavaScript

Grundlagen: Datentyp – Object



[Github](#)

```
1 const phone = {  
2   manufacturer: "Apple",  
3   model: "iPhone",  
4   storage: 128,  
5   isReleased: false,  
6 };  
7  
8 console.log(phone);
```

JavaScript

Grundlagen: Datentyp – Object

- Eine Property kann dabei jeden Datentyp als Wert annehmen
 - Bspw. können wir auch Objekte verschachteln und weitere Objekte als Properties definieren
- Objekte können auch leer definiert werden mit {}

JavaScript

Grundlagen: Datentyp – Object

- Mit . (Dot Notation) können wir einen Wert für eine bekannte Property auslesen
- Wenn die Property noch nicht existiert können wir mit . diese auch hinzufügen
- Properties löschen kann man mit dem **delete** keyword
- Bestehenden Properties kann man . auch modifizieren und andere Werte zuweisen

JavaScript

Grundlagen: Datentyp – Object



[Github](#)

```
1 const phone = {  
2   manufacturer: "Apple",  
3   model: "iPhone",  
4   storage: 128,  
5   isReleased: false,  
6 };  
7  
8 console.log(phone);  
9  
10 console.log('Phone Model: ', phone.model);  
11  
12 phone.price = 1400;  
13 console.log('New Price Property: ', phone);  
14  
15 phone.isReleased = true;  
16 console.log('Now the phone is released: ', phone);  
17  
18 delete phone.storage;  
19 console.log('Now the phone has no more storage property', phone);
```

JavaScript

Grundlagen: Datentyp – Object



[Github](#)

- Der Key von einer Property kann auch ein **multi-word** sein
 - Diese Art von Keys definiert man einfach als strings
- Auf den Wert von einer **multi-word** property kann dann mit der *Square Bracket Notation* zugegriffen werden []
 - Wichtig: Für multi-word properties funktioniert die . Notation nicht.

```
1 const factory = {  
2   "construction site": "Heidenheim an der Brenz"  
3 }  
4  
5 console.log(factory["construction site"]);
```

JavaScript

Grundlagen: Datentyp – Object



[Github](#)

- Als Key kann auch eine **computed property** zum Einsatz kommen
- In einer **computed property** übergeben Sie eine Variable welche Sie als Key nutzen möchten für ein Object

```
1 const fruit = "apple";
2 const bag = {
3   [fruit]: 5
4 }
5
6 console.log(bag);
```

JavaScript

Grundlagen: Typetyp – Object

- JavaScript Objects werden uns jetzt die gesamte Vorlesung begleiten
- Mehr vertiefendes Wissen und Details werden Schritt für Schritt in der Vorlesung vorgestellt

JavaScript

Grundlagen: Datentyp ermitteln



[Github](#)

- JavaScripts dynamische Typisierung erlaubt es, das Variablen zur Laufzeit unterschiedliche Datentypen annehmen können
- Wenn wir jetzt prüfen möchten ob ein Wert / Variable ein bestimmter Datentyp ist, gibt es den **typeof** Operator in JavaScript

```
1 const price = 25;
2 console.log(typeof price); // number
3 const name = "Jens";
4 console.log(typeof name); // string
5 const userInput = null;
6 console.log(typeof userInput); // object
```

JavaScript

Übung

- Definieren Sie sich eine Variable die Informationen über die DHBW abbildet.
- Verwenden Sie die zuvor vorgestellten Konzepte aus Objekten und Primitiven Values
- Geben Sie sich verschiedene Properties von der DHBW in der Konsole aus

JavaScript

Grundlagen: Operatoren



[Github](#)

- Was glauben Sie ist das Ergebnis der Variablen:
 - resultPlus** und **resultMultiply**?

```
1 const input = '5';
2 const multiplier = 5;
3
4 const resultPlus = input + multiplier;
5 const resultMultiply = input * multiplier;
6
7 console.log(resultPlus);
8 console.log(resultMultiply);
```

JavaScript

Grundlagen: Operatoren

- JavaScript Operatoren (+, -, * etc.) wirken auf den ersten Blick Seltsam
- Es gibt aber zwei Regeln, die Sie sich im Umgang mit Operatoren merken sollten:
 1. Operatoren erzwingen eine Typ Konvertierung auf einen passenden Datentyp
 2. Fast alle Operatoren funktionieren nur mit primitiven Datentypen

JavaScript

Grundlagen: Operatoren Regel 1 Beispiel

- In anderen Programmiersprachen würde die folgende Zeile einen Fehler produzieren, da die beiden Datentypen nicht zusammen passen:
 - Ein String kann nicht auf eine Zahl addiert werden
- JavaScript konvertiert aber beide Variablen zu einem String, um eine Verkettung von zwei Strings auszuführen

```
1 const input = '5';
2 const multiplier = 5;
3
4 const resultPlus = input + multiplier;
```

JavaScript

Grundlagen: Operatoren „+“



- JavaScript hat zwei Modi für den „+“ Operator:
 - String Mode – Wenn ein Wert ein String ist wird der andere Wert in einen String umgewandelt und verkettet.
 - Number Mode – Andernfalls werden beide Werte in eine Zahl umgewandelt und das Ergebnis zurückgegeben

```
1 const hello = 'Hello';
2 const world = 'World';
3
4 console.log(hello + " " + world);
```

JavaScript

Grundlagen: Operatoren Arithmetik



[Github](#)

- Alles anderen Mathematischen Operatoren funktionieren wie erwartet

```
1 const number1 = 5;
2 const number2 = 5;
3
4 // Here both values are a number therefore the "expected" arithmetic operator is done.
5 console.log(number1 + number2);
6 console.log(number1 * number2);
7 console.log(number1 - number2);
8
9 const dividend = 20;
10 const divisor = 4;
11
12 console.log(dividend / divisor);
```

JavaScript

Grundlagen: Operatoren Logische Operatoren



[Github](#)

- Mit Logischen (Oder binären) Operatoren können Sie Werte miteinander vergleichen

```
1 console.log(3 > 2);
2 console.log(3 < 2);
3
4 console.log(25 >= 10);
5 console.log(100 <= 50);
6
```

JavaScript

Grundlagen: Binäre Operatoren



[Github](#)

- Mit Binär Operatoren können Sie Boolesche Werte vergleichen

```
1 console.log(true && false);
2 console.log(true && true);
3
4 console.log(true || false);
5 console.log(false || false);
```

JavaScript

Grundlagen: Unary Operator



[Github](#)

- Mit dem Unary (!) Operator können Sie boolesche Bedingungen „invertieren“

```
1 console.log(!true);
2 console.log(!false);
```

JavaScript

Grundlagen: Operatoren Equality



[Github](#)

- Welcher dieser Werte ist gleich?

```
1 const number1 = '123';
2 const number2 = 123;
3
4 console.log(number1 == number2);
5
6 const isApproved = false;
7 const zero = 0;
8
9 console.log(isApproved == zero);
10
11 const emptyString = '';
12
13 console.log(emptyString == zero);
```

JavaScript

Grundlagen: Operatoren Equality

- JavaScript unterscheidet zwischen
 - Loose Equality (==), (!=)
- und
 - Strict Equality (===), (!==)
- Loose Equality wird auch eine Datentyp Umwandlung ausgeführt
- Strict Equality verändert niemals den Datentyp und prüft auf „echte“ Gleichheit

JavaScript

Grundlagen: Operatoren Equality



[Github](#)

```
1 const number1 = '123';
2 const number2 = 123;
3
4 console.log(number1 === number2);
5
6 const isApproved = false;
7 const zero = 0;
8
9 console.log(isApproved === zero);
10
11 const emptyString = '';
12
13 console.log(emptyString === zero);
```

JavaScript

Grundlagen: Operatoren Equality Rule Of Thumb

- Nutzen Sie bitte **IMMER ===** in JavaScript es gibt nur ganz wenige Ausnahmen, wo Sie auf andere Equality Regeln setzen müssen
 - Vergleiche von +0 und -0 (Float Zahlen betrachten Positive und Negative Null)
 - Wenn Sie mit Float Zahlen arbeiten und bspw. Negative Unendlichkeit auf die positive Unendlichkeit addieren
 - Hier gibt es auch noch den Object.is Operator

JavaScript

Grundlagen: Operatoren ?



[Github](#)

- Haben Sie eine Idee, was folgender Befehl ausgibt?

```
1 console.log(true ? 1 : 2);
```

JavaScript

Grundlagen: Ternary Operator ?

- Der Ternary (?) Operator nimmt eine Bedingung und wertet diese aus
 - Vgl. mit einem if / else (Später)
- Ist die Bedingung **true** wird der **left hand operator** ausgeführt
- Ist die Bedingung **false** wird der **right hand operator** ausgeführt

JavaScript

Weiterführende Links

- Operators in JavaScript https://exploringjs.com/js/book/ch_operators.html
- JavaScript StrictEquality Table Live im Browser <https://dorey.github.io/JavaScript-Equality-Table/>
- MDN JavaScript Referenz für Equality Comparisons
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness

JavaScript

Control Flow: IF Statement

- Bedingungen (bspw. ob zwei Werte gleich sind) prüft man in JavaScript mit **if / else** Statements
- **If (...)** evaluiert die Bedingung (Auch Ausdruck genannt) und springt in den definierten Block, wenn die Bedingung zutrifft (true)
- Alternativ kann man mit einem **else** einen Block definieren der ausgeführt wird wenn die Bedingung falsch ist (false)

JavaScript

Control Flow: IF Statement



[Github](#)

```
1 const price = 800;
2 const isCheapPrice = 600;
3
4 if (price <= isCheapPrice) {
5   console.log('Buy Buy Buy')
6 } else {
7   console.log('Wait...')
8 }
```

JavaScript

Control Flow: IF Statement



[Github](#)

- Die zu prüfende Bedingung wird immer in einen Boolean Wert umgewandelt
- *0, „“, null, undefined und NaN werden immer false*

```
1 const userInput = null;
2 if (userInput) {
3   console.log('Has User Input')
4 } else {
5   console.log('Has no user input')
6 }
```

JavaScript

Control Flow: IF Statement

- Sie können `else` Teile auch mit einer Bedingung belegen `else if`
- Sie brauchen keinen `else` Teil sondern können auch nur ein `if` verwenden

JavaScript

Control Flow: Switch Statement



[Github](#)

- Alternativ zu einem if / else kann auch ein **switch** verwendet werden
- Ein **switch** hat einen oder n case Abschnitte und einen optionalen **default** Abschnitt
- **case** Abschnitte werden mit break beendet (Alternativ *return*)

```
1 const price = 2000;
2 switch (price) {
3     case 1000:
4         console.log('Price is 1000');
5         break;
6     case 1500:
7         console.log('Price is 1500');
8         break;
9     case 2000:
10        console.log('Price is 2000');
11        break;
12    default:
13        console.log(`Price is ${price}`);
14 }
```

JavaScript

Control Flow: Loops



[Github](#)

- JavaScript unterstützt folgende Loops:
 - for
 - do while
 - while
- Mit *break* kann man die Schleife abbrechen
- Mit *continue* kann man zur nächsten Iteration springen

```
1  console.log('For Loop')
2  for (let i = 0; i <= 10; i++) {
3      console.log(i);
4  }
5
6  console.log('Do While')
7  let i = 0;
8  do {
9      console.log(i);
10     i++;
11  } while (i <= 10);
12
13 console.log('While')
14 let j = 0;
15 while (j <= 10) {
16     console.log(j);
17     j++;
18 }
```

JavaScript

Funktionen

- In JavaScript gibt es zwei Kategorien von Funktionen
 - *ordinary function*
 - *specialized function*
- Die Unterscheidung kam mit der Einführung von ECMAScript 6 speziell als Klassen in JavaScript eingeführt wurden
 - *Ja JavaScript hatte bis ECMAScript 6 keine Klassen!*

JavaScript

Funktionen - Ordinary & Specialized

- Eine *ordinary function* kann verschiedene Rollen einnehmen
 - Reale Funktion
 - Eine Methode
 - Eine Constructor Funktion
- Eine *specialized function* nimmt genau einer dieser Rollen an
 - Eine Pfeil Funktion* ist eine Reale Funktion
 - Eine Methode kann nur eine Methode sein
 - Eine Klasse kann nur eine Constructor Funktion sein

JavaScript

Funktionen – Ein einfaches Beispiel

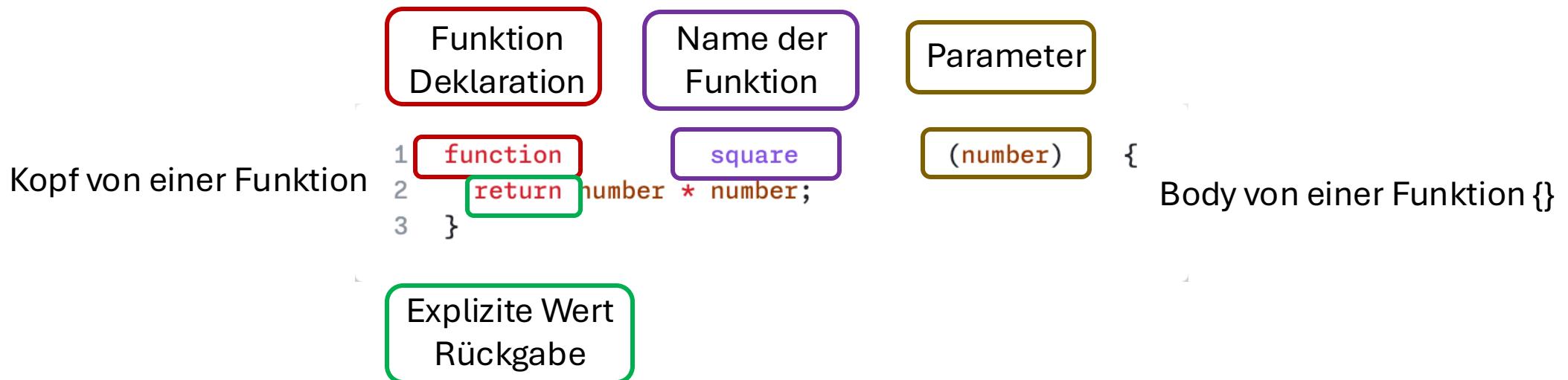


[Github](#)

```
1 function square(number) {  
2     return number * number;  
3 }  
4  
5 const result = square(5);  
6 console.log(result);
```

JavaScript

Funktionen – Aufbau und Terminologie



JavaScript

Funktionen – Anonyme Funktionen



[Github](#)

- Funktionen können auch anonym deklariert werden
- Anonyme Funktionen sind Variablen die eine Funktion definieren

```
1 const square = function (number) {  
2   return number * number;  
3 };  
4  
5 const result = square(5);  
6 console.log(result);
```

JavaScript

Funktionen – Square in unterschiedlichen Rollen



[Github](#)

```
1 function square(number) {  
2   return number * number;  
3 }  
4  
5 // Real Function -> Invoked via a function call  
6 const realFunctionResult = square(5);  
7 console.log(realFunctionResult);  
8  
9 const obj = { squareMethod: square};  
10  
11 // Method Function -> Stored as a property inside an object and invoked as a method  
12 const methodFunctionResult = obj.squareMethod(5);  
13 console.log(methodFunctionResult);  
14  
15 const constructorFunctionResult = new square();  
16 console.log(constructorFunctionResult)
```

JavaScript

Funktionen – Parameter

- An Funktionen können Sie eine beliebige Anzahl an Parametern übergeben
- Nicht übergebene Parameter werden automatisch als **undefined** deklariert
- Sie können Parametern auch mit Default Werten definieren, wenn der Parameter nicht übergeben wird, greift der Default Wert

JavaScript

Funktionen – Parameter Beispiele



[Github](#)

```
1 function addPrefix(input, prefix = 'id: ') {  
2   return `${prefix} ${input}`;  
3 }  
4  
5 console.log(addPrefix('Test'));  
6 console.log(addPrefix('iPhone', 'Pro Max'));  
7
```

JavaScript

Funktionen – Primitive Parameter

- Primitive Parameter (vgl. Primitive Values) werden **pass by value** an die Funktion übergeben
- Nicht Primitive Parameter (vgl. Objects) werden **pass by reference** an die Funktion übergeben

JavaScript

Funktionen – Primitive Parameter



[Github](#)

- Wenn Sie an eine Funktion ein Objekt übergeben und eine Property verändern, wird das übergebene Objekt modifiziert

```
1  function updatePhone(phone) {
2    phone.year = '2024';
3  }
4
5  const phone = {
6    maker: 'Apple',
7    model: 'iPhone',
8    year: '2023'
9  }
10
11 console.log('Phone Year before: ', phone.year);
12 updatePhone(phone);
13 console.log('Phone Year afterwards: ', phone.year);
```

JavaScript

Funktionen – Rückgabewerte



[Github](#)

- Funktionen in JavaScript geben immer **undefined** zurück
 - Das ist auch der Grund wieso bspw. ein `console.log` im Browser ein `undefined` anschließend ausgibt
- Explizite Rückgabewerte können mit einem **return** Statement angegeben werden

```
1 function logError(errorMessage) {  
2   console.error(errorMessage);  
3 }  
4  
5 logError('Something went wrong');  
6  
7 function toUpperCase(input) {  
8   return input.toUpperCase();  
9 }  
10  
11 console.log(toUpperCase('Hello World'));
```

JavaScript

Funktionen – Arrow Functions (=>)



[Github](#)

- In JavaScript können Funktionen auch als *arrow functions* deklariert werden

```
1 const square = (number) => number * number;  
2  
3 const result = square(5);  
4 console.log(result);
```

```
1 const multiply = (number1, number2) => {  
2   return number1 * number2;  
3 }  
4  
5 const result2 = multiply(5, 2);  
6 console.log(result2);
```

JavaScript

Funktionen – Arrow Functions (=>)

- Arrow Functions geben implizit einen Rückgabewert zurück
 - Bedeutet man muss nicht explizit return angeben
- Arrow Functions können nicht als Constructor verwendet werden
 - Vgl. mit Rollen von Funktionen
- Arrow Functions können nicht als Methode in Objekten deklariert werden
 - Technisch möglich, man sollte es aber vermeiden
- Kein Binding auf **this** oder **super**

JavaScript

Funktionen – This?

- **this** ist eine spezielle Variable auf die wir in JavaScript zugreifen können
 - this spielt primär in Objekt Orientierten Sprachen eine primäre Rolle um, einfach gesagt, auf Properties in einer Klasse zuzugreifen
- Mit this können wir bspw. in einer Methode auf das Objekt zugreifen, dass den Methoden Aufruf bekommen hat

JavaScript

Funktionen – This Beispiele



[Github](#)

- In Objekten bekommen Sie das Objekt über „this“
- In Realen Funktionen bekommen Sie `undefined`
 - Wenn JavaScript im Strict Modus ist
 - Bspw. in Node bekommen wir das Globale Objekt zurück

```
1 const obj = {  
2   square(number) {  
3     console.log(this);  
4     return number * number;  
5   }  
6 }  
7  
8 obj.square(5);  
9  
10 function square(number) {  
11   console.log(this);  
12   return number * number;  
13 }  
14  
15 square(5);
```

JavaScript

Funktionen – Funktionen als Parameter

- In JavaScript können Sie auch Funktionen als Parameter übergeben
- Funktionen als Parameter übergeben zu können ist ein wichtiges Merkmal von Funktionalen Programmiersprachen
 - Vgl. mit Objekt Orientierten Programmiersprachen
- Das Konzept eine Funktion an eine andere zu übergeben werden Sie in JavaScript (Und im späteren Verlauf) sehr häufig vorfinden

JavaScript

Funktionen – Funktionen als Parameter



[Github](#)

```
1 const upperCase = (input) => input.toUpperCase();
2
3 const phone = {
4   model: 'iPhone Pro Max',
5   printModel(transformFn) {
6     return transformFn(this.model);
7   }
8 }
9
10 console.log(phone.printModel(upperCase));
```

JavaScript

Funktionen – Funktionen als Parameter

- Mit diesem Konzept können Sie bspw. ihre eigenen Funktionen schreiben wie der Modell Name ausgegeben werden soll.
- Anbei ein Beispiel, welches jeden Buchstaben mit einem zufälligen Emoji ersetzt
 - Das Beispiel nutzt bereits Konzepte die wir uns noch nicht angeschaut haben

JavaScript

Funktionen – Funktionen als Parameter



[Github](#)

```
1 const replaceWithEmoji = (str) => {
2   const emojis = ['😊', '😂', '😍', '😎', '🤔', '🥳', '😱', '😴', ' CircularProgress', '😴'];
3
4   return str.split('').map(char => {
5     if (/[^a-zA-Z]/.test(char)) {
6       return emojis[Math.floor(Math.random() * emojis.length)];
7     } else {
8       return char;
9     }
10   }).join('');
11 };
12
13 const phone = {
14   model: 'iPhone Pro Max',
15   printModel(transformFn) {
16     return transformFn(this.model);
17   }
18 }
19
20
21 console.log(phone.printModel(replaceWithEmoji));
```

JavaScript

Hinweise

- Sie können sich auf Objekten Funktionen (bzw. Methoden) definieren, aber gerade wenn Sie Objekte aus der Realität modellieren empfiehlt es sich, keine Methoden direkt auf einem Objekt definiert zu haben
 - Warum? Später dazu mehr aber Sie können ihre Repräsentation von Daten dann nicht mehr per JSON schicken
- Stattdessen schreiben Sie sich lieber explizite Funktionen, die Sie aufrufen und wiederverwenden können

JavaScript

Funktionen – Übung

- Schreiben Sie sich ein JavaScript Programm, welches für ein übergebenes Objekt an einer gewählten Property eine Value verändern kann. Geben Sie anschließend das überarbeitete Objekt wieder zurück
- Schreiben Sie sich ein JavaScript Programm, welches zwei Zahlen entgegen nimmt und eine Funktion. Die übergebene Funktion soll eine Mathematische Operation abbilden, die auf die zwei Zahlen angewandt wird. Definieren Sie sich die Mathematischen Operationen als Anonyme Funktionen

JavaScript

Funktionen – Weiterführende Links

- JavaScript Callable Values
https://exploringjs.com/js/book/ch_callables.html#ch_callables
- MDN Function https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function
- MDN Arrow Function Expressions https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

JavaScript

Arrays – Deklaration



[Github](#)

- Arrays in JavaScript können als Array Literal definiert werden

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen']
2 console.log(cars);
```

JavaScript

Arrays – Eigenschaften

- Arrays bieten gegenüber Objekten eine geordnete Struktur mit der Möglichkeit Daten an beliebiger Stelle zu modifizieren
 - Jedes Element in einem Array hat einen eindeutigen Index, an diesem Index finden wir dann den Wert
- Arrays sind nicht typisiert in JavaScript
- Arrays können eine beliebige Länge annehmen und die Länge (Größe) muss nicht bei der Deklaration angegeben werden

JavaScript

Arrays – Lesen von Werten



[Github](#)

- Mit der Square Bracket Notation und einem Index können wir den aktuellen Wert lesen
- Mit der gleichen Notation können wir Elemente modifizieren

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen'];
2 const result = cars[1];
3
4 console.log(result);
5
6 cars[1] = 'Tesla';
7
8 console.log(cars);
```

JavaScript

Array – Properties



[Github](#)

- Ein JavaScript Array ist im Hintergrund einfach ein Objekt
- Wir können deshalb auch auf Object Properties zugreifen die speziell für ein Array sind bspw. `.length` gibt uns die Aktuelle Länge vom Array zurück

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen'];
2
3 for (let i = 0; i < cars.length; i++) {
4   console.log(cars[i]);
5 }
```

JavaScript

Array – Methoden

- Das Array Objekt bietet uns deshalb auch eine Reihe an Methoden an, um mit einem Array unkompliziert arbeiten zu können
- Anbei die wichtigsten Methoden die man kennen sollte
 - Eine vollständige Liste befindet sich am Ende vom Kapitel

JavaScript

Array – Methoden Push & Pop



[Github](#)

- Mit **push** kann ein neues Element an das **Ende** von einem Array hinzugefügt werden
- Mit **pop** kann das **letzte** Element aus einem Array entfernt werden

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen']
2 console.log('Before: ', cars);
3
4 cars.push('Volvo');
5 console.log('Added Volvo as last element: ', cars);
6
7 cars.pop();
8 console.log('Removed Last Element', cars)
```

JavaScript

Array – Methoden Unshift & Shift



[Github](#)

- Mit **unshift** kann ein neues Element an den **Anfang** von einem Array hinzugefügt werden
- Mit **shift** kann das **erste** Element aus einem Array entfernt werden

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen']
2 console.log('Before: ', cars);
3
4 cars.unshift('Volvo');
5 console.log('Added Volvo as first Element: ', cars);
6
7 cars.shift();
8 console.log('Removed first Element', cars)
```

JavaScript

Arrays – Elemente suchen



[Github](#)

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen']
2
3 const hasMercedes = cars.includes('Mercedes');
4 console.log('Has cars a mercedes: ', hasMercedes);
5
6 const indexOfAudi = cars.indexOf('Audi');
7 console.log('Audi is at index: ', indexOfAudi);
8
9 const volkswagen = cars.find((car) => car === 'Volkswagen');
10 console.log('Volkswagen Element: ', volkswagen);
```

JavaScript

Arrays – Elemente suchen

- Ist Ihnen etwas in der vorherigen Implementierung aufgefallen?

JavaScript

Arrays – Elemente suchen

- Viele Array Methoden unterstützen eine Funktion als Parameter
- In diesem Beispiel konnten wir der **find** Methode eine Funktion mitgegeben nach was wir suchen möchten
- Wir können also selbst definieren nach welchem Element wir suchen indem wir eine Funktion mitgeben und die **find** Methode jedes Element gegen die Funktion ausführt
- Wenn unsere Funktion ein **true** zurück gibt wird das Gefundene Element zurückgegeben

JavaScript

Arrays – Methoden mit Callbacks erweitern

- Methoden die eine Funktion als Parameter entgegen nehmen werden auch als Callback Funktionen bezeichnet
- Die Callback Funktion kann dabei komplett selbst definiert werden und die Array Methode wendet auf jedes Element die Callback Methode an
- Dieses Konzept ist auch bekannt als Higher Order Functions und finden Sie häufig in funktionalen Programmiersprachen
 - Anstatt, dass Sie komplizierte Parameter / Methoden definieren übergeben Sie einfach eine eigene Funktion an eine andere

JavaScript

Arrays – Methoden die Arrays transformieren



[Github](#)

- Die Folgenden Methoden nehmen eine Callback Funktion entgegen und transformieren ein gegebenes Array in ein neues Array.

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen']
2
3 const carsAsObjects = cars.map((car) => {
4   const carObj = {
5     manufacturer: car,
6     isCheap: false,
7   }
8   return carObj;
9 })
10
11 console.log('List of Cars but as objects: ', carsAsObjects);
```

JavaScript

Arrays – Methoden die Arrays transformieren



[Github](#)

- Mit der **filter** Methode können Werte aus einem bestehenden Array ermittelt werden die true für eine Filter Funktion zurückgeben

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen']
2
3 const isPartOfVolkswagen = cars.filter((car) => car === 'Audi' || car === 'Volkswagen');
4 console.log(isPartOfVolkswagen);
```

JavaScript

Arrays – Methoden Elemente prüfen



[Github](#)

- Zwei spezielle Array Methoden sind **some** und **every** und können verwendet werden ob bspw. alle Elemente von einem Array true zurück geben oder nur einzelne
- Wichtig beide Funktionen geben nur true oder false zurück. Nicht aber bspw. welche Elemente die Bedingung nicht erfüllen

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen', { manufacturer: 'Volvo', model: 'Polestar'}];
2
3 const allCarsAreStrings = cars.every((car) => typeof car === 'string');
4 console.log('Are all cars of type string? ', allCarsAreStrings);
5
6 const hasAudi = cars.some((car) => car === 'Audi');
7 console.log('Is some car an audi? ', hasAudi);
```

JavaScript

Arrays – Weitere Methoden

- **concat** – Kombiniert zwei Arrays zu einem Array
- **slice** – Erstellt eine Kopie von einem bestehenden Array (neue Referenz)
 - Optional: Von Start bis End Index
- **reverse** – Dreht den Inhalt von einem Array um
- **sort** - Sortiert den Inhalt anhand einer Sortierfunktion welche übergeben werden kann
 - Tipp: Sortierfunktionen bspw. Werte vergleichen kann ChatGPT ganz gut definieren
- **splice** – Entfernt ein Element und kann gleichzeitig das Entfernte Element mit einem neuem ersetzen
 - Optional: Von Start Bis End Index
 - Optional: Element welches anstelle eingesetzt werden soll

JavaScript

Arrays – Loops

- Arrays können klassisch mit einer **for** loop durchlaufen werden indem man den Index in jedem Durchlauf erhöht und über die **[]** Syntax auf das aktuelle Element zugreift
- Arrays in JavaScript bieten aber auch noch eine spezielle **forEach** Methode an.
- Hier kann eine eigene Arrow Function definiert werden die auf jedes Element angewandt wird

JavaScript

Arrays – Loops



[Github](#)

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen']
2 for (let i = 0; i < cars.length; i++) {
3   console.log(cars[i]);
4 }
5
6 cars.forEach((car) => {
7   console.log(car);
8 })
```

JavaScript

Arrays – for of loop



- Eine spezielle Loop die es in anderen Programmiersprachen seltener gibt ist die **for of** Loop in JavaScript

```
1 const cars = ['Mercedes', 'BMW', 'Audi', 'Volkswagen']
2
3 for (const car of cars) {
4   console.log(car);
5 }
6
```

JavaScript

Arrays – Übung

- Schreiben Sie sich ein JavaScript Programm, dass eine Liste von Autos als Objekte mit unterschiedlichen Eigenschaften speichert. Definieren Sie sich anschließend eine Funktion, dass aus dieser Liste genau ein Fahrzeug findet anhand einer übergebener Property und Richtwert, nach welcher gesucht werden soll. Geben Sie das gefundene Fahrzeug zurück.
- Schreiben Sie anschließend das Ergebnis in ein neues Array zurück
- Beispiel: Ich möchte das Auto suchen, welches einen Preis von 25000 hat

JavaScript

Arrays – Hinweise

- Es gibt noch die for ... in Loop in JavaScript
 - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in>
- Allerdings iteriert diese Schleife auch über alle Prototype Properties von einem Objekt was häufig nicht gewünscht ist, wenn Sie nur ihre Objekt Properties auswerten möchten
- Deshalb empfiehlt es sich auch nicht diese Schleife zu verwenden
 - Siehe https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in#iterating_over_own_properties_only

JavaScript

Arrays – Weiterführende Links

- Arrays https://exploringjs.com/js/book/ch_arrays.html
- MDN Array Definition https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
- MDN Array Instance Methoden https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#instance_methods

JavaScript

Umsetzung einer Software Anforderung

Vorlesung

Hinweis

- Im weiteren Verlauf der Vorlesung werden wir jetzt gemeinsam eine komplette Softwarelösung bauen.
- Wir beginnen mit einfachen Meilensteinen und erweitern unsere Lösung um immer neuere Use Cases / Anforderungen
- Am Ende der Vorlesung haben Sie dann eine vollständige Software Lösung die alle Themenbereiche der Vorlesung abdeckt
- Die weiteren Konzepte / Themen etc. werden in dieser Softwarelösung untergebracht
- Die Komplette Lösung, Meilensteine etc. finden Sie natürlich auch alle auf Github

JavaScript

Software Anforderung

Wir möchten eine neue App für Bier Trinker auf den Markt bringen. Mit dieser App kann man alle seine Lieblings Biersorten eintragen und bewerten. Darüber hinaus soll es die Möglichkeit geben festzuhalten wie viel man von der jeweiligen Biersorte bereits getrunken hat.

In einem ersten Meilenstein soll dafür ein Datenmodell definiert werden und die ersten Funktionen geschrieben werden. Für den Anfang möchten wir neue Biere hinzufügen können. Für ein bestehendes Bier festhalten wie viel von dieser Sorte getrunken wurde und eine Ausgabe über das Bier mit der besten Bewertung und am meisten getrunken bereitstellen.

JavaScript

Bier App – Anforderungen bestimmen

- In einem ersten Meilenstein soll dafür ein Datenmodell definiert werden und die ersten Funktionen geschrieben werden. Für den Anfang möchten wir neue Biere hinzufügen können. Für ein bestehendes Bier festhalten wie viel von dieser Sorte getrunken wurde und eine Ausgabe über das Bier mit der besten Bewertung und am meisten getrunken bereitstellen.
- Anforderung Datenmodell definieren
- Anforderung Bier erstellen
- Anforderung Biere speichern
- Anforderung Bier Anzahl getrunken
- Anforderung Bier Bewertung
- Anforderung Biere auswerten

JavaScript

Bier App – Datenmodell Definieren

- Ein Datenmodell für ein Bier können wir in JavaScript als Objekt abbilden.
- Ein Objekt Bier hat dabei unterschiedliche Properties
- Aus der Anforderung: Anzahl getrunken, Bewertung
- Welche Properties fallen Ihnen noch ein?

JavaScript

Bier App – Bier erstellen

- Die Anforderung ein Bier zu erstellen, bedeutet wir müssen eine Funktion schreiben die Properties für ein neues Bier entgegen nimmt und uns ein Objekt Bier erstellt.
- Als Datenmodell sollten wir das zuvor definierte Bier verwenden
- Als Parameter erwarten wir die Eigenschaften, die das Bier identifizieren also bspw:
 - Name, Hersteller und Geschmack

JavaScript

Bier App – Bier erstellen

```
1 function createBeer(beerProps) {  
2   const beer = {  
3     name: beerProps.name,  
4     taste: beerProps.taste,  
5     brand: beerProps.brand,  
6     amount: 0,  
7     rating: 1,  
8   };  
9  
10  return beer;  
11}  
12
```

JavaScript

Bier App – Biere Speichern

- Für das Speichern von einem Bier verwenden wir einfach ein Array (Liste) und fügen jedes Bier dieser Liste hinzu.

```
1 const beers = [];
2
3 function addBeer(beer) {
4   beers.push(beer);
5 }
6
7 function getBeers() {
8   return beers;
9 }
```

```
1  function createBeer(beerProps) {
2    const beer = {
3      name: beerProps.name,
4      taste: beerProps.taste,
5      brand: beerProps.brand,
6      amount: 0,
7      rating: 1,
8    };
9
10   addBeer(beer);
11
12   return beer;
13 }
```

JavaScript

Bier App – Anzahl getrunken

- Wie oft ein Bier getrunken wurde, speichern wir uns auf dem Bier Objekt in einer property „amount“
- Um ein Bier zu trinken, ergänzen wir eine neue Funktion, die ein bestehendes Bier entgegen nimmt und den amount Wert hochzählt

```
1 function drinkBeer(beer) {  
2   beer.amount++;  
3   return beer;  
4 }  
5
```

JavaScript

Bier App – Bewertung von einem Bier

- Wir verwenden das gleiche Konzept, um ein Bier zu bewerten. Erweitern aber die Funktion um eine Bewertung die nur Werte zwischen 1 und 5 zulässt

```
1 function rateBeer(rating = 1, beer) {  
2     if (rating >= 1 && rating <= 5) {  
3         beer.rating = rating;  
4     }  
5     return beer;  
6 }
```

JavaScript

Bier App – Auswertung bestes Bier

- Objekte werden in JavaScript als **pass by reference** an Funktionen übergeben
 - JavaScript erstellt keine Kopie von unserem Objekt sondern übergibt die Adresse im Speichern
- Dieses Konzept nutzen wir, um direkt das bestehende Objekt zu modifizieren
- Dadurch wird auch automatisch unsere Liste, wo alle Biere gespeichert werden, modifiziert

JavaScript

Bier App – Auswertung bestes Bier

- Für die Auswertung, welches Bier am besten ist erstellen wir eine neue Funktion und definieren die Logik, dass jedes Bier mit einer Bewertung ab 4 zurück gegeben werden soll

```
1 function getBestRatedBeers() {  
2   const bestRated = beers.filter((beer) => beer.rating >= 4);  
3   return bestRated;  
4 }
```

JavaScript

Bier App – Am meisten getrunken

- Wir definieren uns wieder eine Funktion, die über die Liste mit allen Bieren geht und jedes Bier mit der amount Property vergleicht

```
1 function getMostDrankBeer() {  
2     if (beers.length === 0) {  
3         return null;  
4     } else {  
5         let mostDrankBeer = beers[0];  
6  
7         for (let i = 1; i < beers.length; i++) {  
8             if (beers[i].amount > mostDrankBeer.amount) {  
9                 mostDrankBeer = beers[i];  
10            }  
11        }  
12  
13        return mostDrankBeer;  
14    }  
15}
```

JavaScript

Bier App – Ausprobieren

```
1 const nattheimerSpezial = createBeer({  
2   name: "Spezial",  
3   taste: "Good",  
4   brand: "Nattheimer",  
5 });  
6 const goldOchsenKellerbier = createBeer({  
7   name: "Kellerbier naturtrüb",  
8   taste: "Good",  
9   brand: "Ulmer Gold Ochsen",  
10});  
11  
12 drinkBeer(nattheimerSpezial);  
13 drinkBeer(nattheimerSpezial);  
14 drinkBeer(nattheimerSpezial);  
15 rateBeer(5, goldOchsenKellerbier);  
16  
17 console.log("All Beers: ", getBeers());  
18 console.log("Most Drank Beer: ", getMostDrankBeer());  
19 console.log("Best Rated Beer: ", getBestRatedBeers());
```

JavaScript

Bier App – Bewertung der Software Lösung

- Was fällt ihnen auf bei der Implementierung?
- Was sind Dinge, die wir ggf. besser implementieren können?
- Was kann die Software Lösung noch nicht?
- Wie finden Sie die Struktur vom Code?

JavaScript

Bier App – Source Code

- Der vollständige Source Code für Meilenstein finden Sie hier
<https://github.com/wasdJens/dhbw-webprogrammierung-wwi/blob/master/10-Beer-Solution/01-milestone/beer-app.js>

JavaScript

Module in JavaScript

JavaScript

Module – Warum Module?

- Größere Softwarelösungen sind in der Regel in unterschiedliche Bereiche unterteilt
- Häufig werden Funktionalitäten, Use Cases, Anbindungen etc. gruppiert und in einzelnen Dateien gespeichert
 - Gerade Komplexere Software Lösungen implementieren eine Funktion in genau einer Datei bspw. eine Klasse, die sich um eine Aufgabe kümmert
- Was könnten wir in unserer Meilenstein 01 Bier App Lösung bspw. auslagern?

JavaScript

Bier App – Optimierung und Aufteilen in Module

- Das Speichern von einem Bier in einer Liste sowie die Auswertung der Biere könnte in eine Datei ausgelagert werden
 - Auf die Variable beers soll auch nicht von außen zugegriffen werden
- Das Anlegen von einem Bier sowie den Funktionen ein Bier zu trinken könnte in eine Datei ausgelagert werden
- Das Programm verwenden also ein Bier anlegen und damit arbeiten könnte in eine spezielle „Start“ Datei ausgelagert werden

JavaScript

Module – Eigenschaften von Modulen

- Mit Modulen kann Code gruppiert und wiederverwendet werden
- Module können exportiert und importiert werden
- Module können Abhängigkeiten zu anderen Modulen haben
- Module können als Bausteine verwendet werden, um unterschiedliche Funktionalitäten abzubilden

JavaScript

Module – Eigenschaften von Modulen

- Auf Modul Funktionen oder Variablen kann nur zugegriffen werden, wenn diese vom Modul exportiert werden

JavaScript

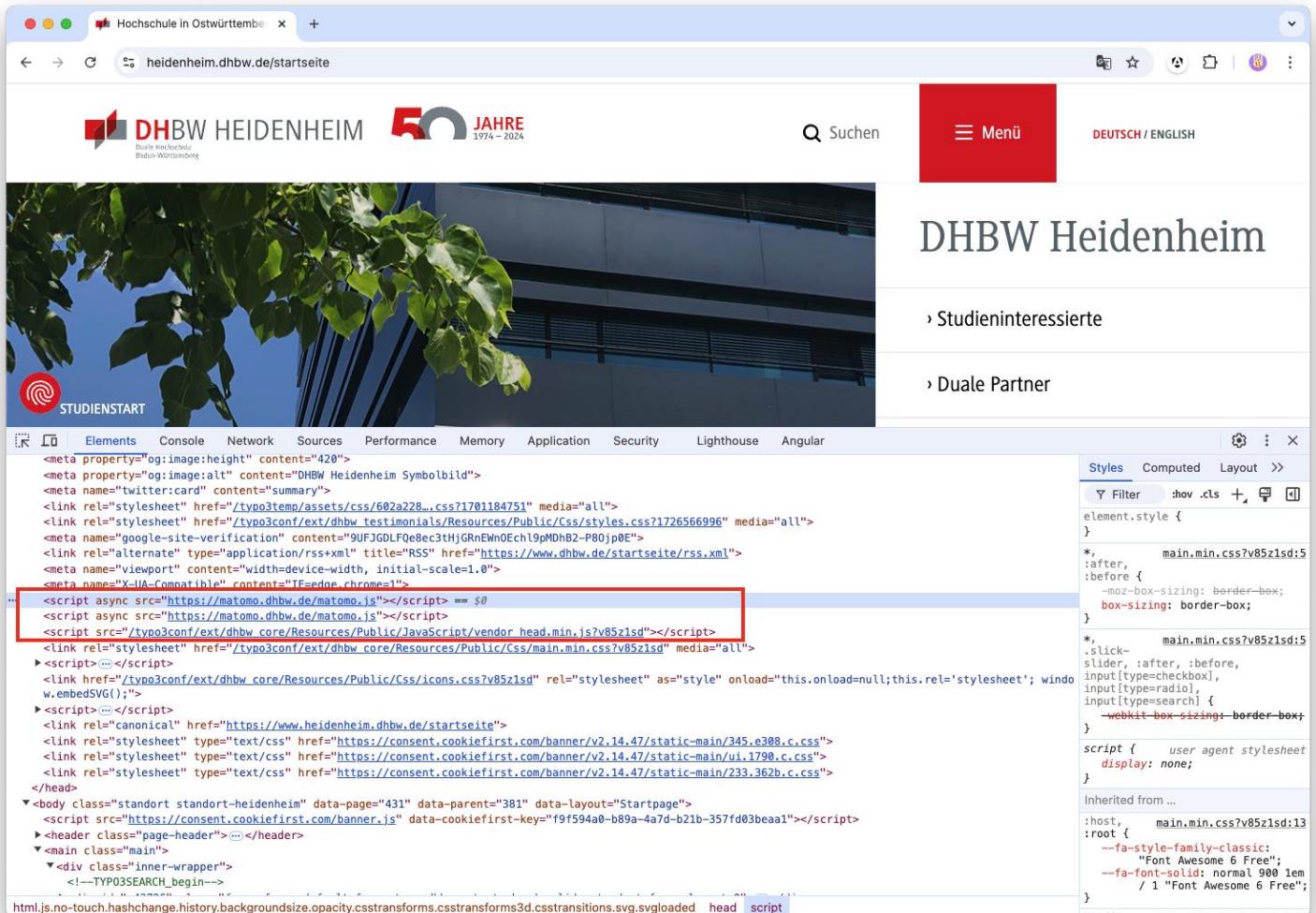
Module – Source Code Formate

- JavaScript hatte bis ECMAScript 6 keine eingebaute Modul Funktionalität
 - Bedeutet Code aus einer anderen Datei importieren wurde von der Sprache nicht unterstützt
 - Der Namespace in JavaScript war immer global
- Zusätzlich hat man unterschieden wo JavaScript Code ausgeführt wird
 - Im Browser
 - Auf Servern (bspw. NodeJS)

JavaScript

Module – Browser Module

- Scripts – Code Fragmente die im Browser immer im Global Scope ausgeführt wurden
- Über Global definierte Funktionen könnten Module ähnliche Konstrukte gebaut werden
 - Siehe https://exploringjs.com/js/boook/ch_modules.html#scripts



JavaScript

Module – Community Module Formate

- Für NodeJS (Und andere JS-Implementierungen) hat die Community eigene Modul ähnliche Formate definiert mit den bestehenden Funktionen von JavaScript
- CommonJS (Kurz CJS)
- AMD (Betrachten wir nicht in der Vorlesung)

JavaScript

Module – CommonJS Format

- Das Originale NodeJS Module System wurde komplett mit CommonJS entworfen
- CommonJS hat folgende Eigenschaften:
 - Designed for servers (Geht nicht im Browser)
 - Module werden **synchron** geladen (Wenn ich ein Modul importiere warte ich bis das Modul geladen und ausgeführt wurde)
 - Kompakte Syntax

JavaScript

Module – CommonJS Format



[Github](#)

- CommonJS kann verwendet werden indem man in NodeJS die Dateiendung **.cjs** verwendet
- Variablen, Funktionen etc. können über ein spezielles module Objekt exportiert werden
- Importieren bietet NodeJS eine spezielle require() Funktion an

```
1 module.exports = {  
2   addBeer,  
3   getBeers,  
4   getBestRatedBeers,  
5   getMostDrankBeer  
6 }
```

```
1 const beersModule = require('./beers.cjs');
```

JavaScript

Bier App – CommonJS Beispiel



[Github](#)

```
1 const beersModule = require('./beers.cjs');
2 const beerModule = require('./beer.cjs');
3
4 const nattheimerSpezial = beerModule.createBeer({
5   name: "Spezial",
6   taste: "Good",
7   brand: "Nattheimer",
8 });
9 const goldOchsenKellerbier = beerModule.createBeer({
10   name: "Kellerbier naturtrüb",
11   taste: "Good",
12   brand: "Ulmer Gold Ochsen",
13 });
14
15 beerModule.drinkBeer(nattheimerSpezial);
16 beerModule.drinkBeer(nattheimerSpezial);
17 beerModule.drinkBeer(nattheimerSpezial);
18 beerModule.rateBeer(5, goldOchsenKellerbier);
19
20 console.log("All Beers: ", beersModule.getBeers());
21 console.log("Most Drank Beer: ", beersModule.getMostDrankBeer());
22 console.log("Best Rated Beer: ", beersModule.getBestRatedBeers());
23
```

JavaScript

Module – Eigenschaften

- Ein Modul in JavaScript ist also einfach eine einzelne JavaScript Datei
- In dieser JavaScript Datei können dann Funktionen, Klassen (Später) oder Variablen deklariert und exportiert werden
- Alle nicht exportierten Werte sind von außen nicht sichtbar
- Importieren erfolgt über eine Pfad Angabe zum Modul, welches importiert werden soll

JavaScript

Module – ECMAScript Modules

- Mit der Einführung von ECMAScript 6 gibt es ein offizielles Module Format im JavaScript Standard: ESModules (Kurz esm)
- Die Syntax wurde noch weiter vereinfacht und ähnelt anderen Programmiersprachen
- Module können statische Strukturen aufweisen (Kann zur Runtime nicht verändert werden) wodurch bspw. dead code elimination implementiert werden kann
 - Code der nicht verwendet wird, wird auch nicht ausgeliefert
- ESM Format wird vom Browser unterstützt!

JavaScript

Module – ECMAScript Modules

- Die Notation **import** und **export** wurden als Standard in die JavaScript Notation aufgenommen
- Im Browser können **script** tags mit **type=module** deklariert werden
- ESM Module werden nur einmalig ausgeführt und können asynchron geladen werden

JavaScript

Module – ESM Format

- ESM kann verwendet werden indem man .mjs als Dateiendung nutzt
- Über das export Statement können Funktionen, Variablen etc. exportiert werden
- Über das import Statement können Funktionen, Variablen etc. importiert werden

```
1  export { addBeer, getBeers, getBestRatedBeers, getMostDrankBeer };
```

```
1  import { addBeer } from "./beers.mjs";
```

JavaScript

Bier App – ESM Beispiel



[Github](#)

```
1 import { createBeer, drinkBeer, rateBeer } from "./beer.mjs";
2 import { getBeers, getMostDrankBeer, getBestRatedBeers } from "./beers.mjs";
3
4 const nattheimerSpezial = createBeer({
5   name: "Spezial",
6   taste: "Good",
7   brand: "Nattheimer",
8 });
9 const goldOchsenKellerbier = createBeer({
10   name: "Kellerbier natürtrüb",
11   taste: "Good",
12   brand: "Ulmer Gold Ochsen",
13 });
14
15 drinkBeer(nattheimerSpezial);
16 drinkBeer(nattheimerSpezial);
17 drinkBeer(nattheimerSpezial);
18 rateBeer(5, goldOchsenKellerbier);
19
20 console.log("All Beers: ", getBeers());
21 console.log("Most Drank Beer: ", getMostDrankBeer());
22 console.log("Best Rated Beer: ", getBestRatedBeers());
23
```

JavaScript

Module – Eine Übersicht

- Seit 2015 gibt es das ESM Format welches offiziell in JavaScript unterstützt wird und die Eigenschaften von Modulen implementiert
- Für NodeJS gibt es das CommonJS Format welches speziell für Server Anwendungen eingesetzt wird
 - Eine Community Implementierung die Module Funktionalitäten in JavaScript anbietet

JavaScript

Module – In der Vorlesung

- Wir werden das ESM Format verwenden in der gesamten Vorlesung
- NodeJS kann beide Formate (ESM und CJS) und wir können Node mitgeben, welches Format genutzt wird:
 - Mit der Dateiendung **.cjs** oder **.mjs**
 - Über die **type** Property in der package.json Datei (Später)
 - Über das Flag **--input-type** wenn man den Node Befehl nutzt
 - *Automatisch – NodeJS kann anhand der Dateien erkennen welches Format genutzt werden soll*

JavaScript

Module – in der Vorlesung

- Sie werden speziell im NodeJS Kontext viele Beispiele finden die noch CommonJS einsetzen
- Es gibt auch noch Libraries die kein ESM können (Selten)
- ChatGPT und andere AI Tools geben auch gerne Antworten im CommonJS Format zurück
- Wichtig: Verständnis für die Module Formate und erkennen in einer Code Base anhand der **require** Funktion oder **import** Statements

JavaScript

Modules – Weiterführende Links

- Modules https://exploringjs.com/js/book/ch_modules.html
- MDN Referenz <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules>
- NodeJS CJS Referenz <https://nodejs.org/api/modules.html>
- NodeJS ESM Referenz <https://nodejs.org/api/esm.html>

Verteilte Systeme

Das Internet als Verteiltes System

Verteilte Systeme

Bier App – Neue Anforderungen

Für die Bier App soll eine Schnittstelle bereitgestellt werden die es anderen Software Herstellern und Usern erlaubt auf die Daten zuzugreifen. Die Schnittstelle soll über das Internet erreichbar sein.

Im zweiten Meilenstein soll unsere Bier App erweitert werden und alle Funktionalitäten über eine Server Schnittstelle angeboten werden. Andere Clients oder Server können über diese Schnittstelle die zuvor implementierten Funktionen verwenden. Wir möchten die Bier Funktionen in anderen Programmen nutzen.

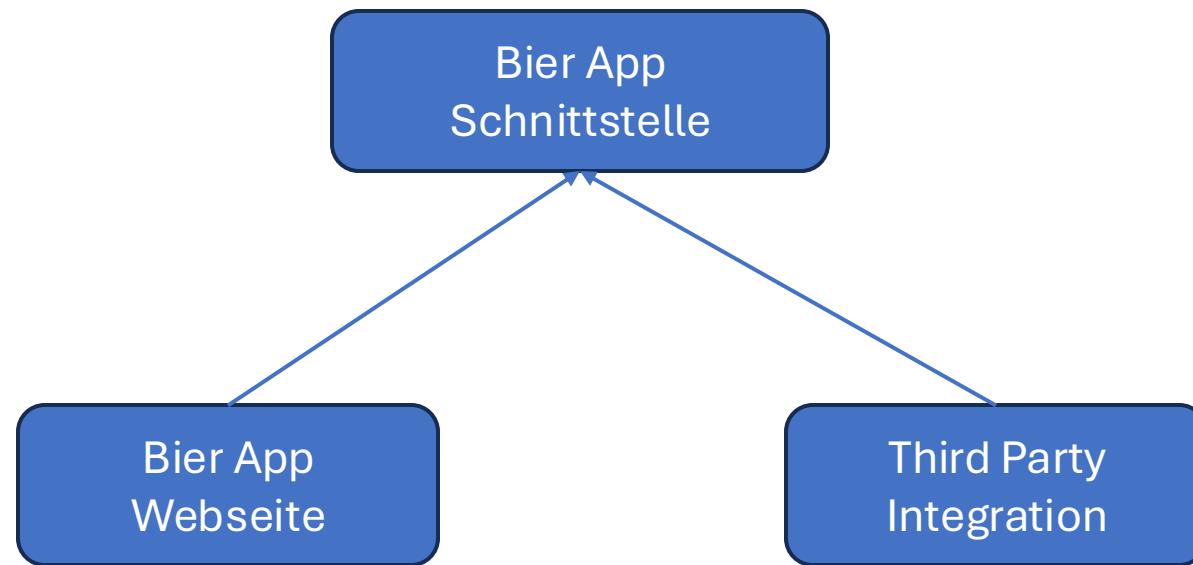
Verteilte Systeme

Bier App – Erster Lösungsentwurf

- Wie würden Sie die neuen Anforderungen umsetzen?
- Was für Protokolle kennen Sie?
- Was für Server Schnittstellen kennen Sie?
- Haben Sie schonmal mit solchen Software Lösungen Kontakt gehabt?
 - Im Betrieb? Wenn ja welche?

Verteilte Systeme

Bier App – Erster Lösungsentwurf



Verteilte Systeme

Definition

- *Ein verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheint*
 - Tannenbaum, van Steen (2002)
- *A distributed system is one which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages*
 - Coulouris, Dollimore & Kindberg (2005)
- *Ein verteiltes Dateisystem ist eines, in dem mehrere autonome Prozessoren und Datenspeicher [...] so kooperierend zusammenarbeiten, dass ein gemeinsames Ziel erreicht wird. Die Prozesse koordinieren ihre Aktivitäten und tauschen Informationen über ein Kommunikationsnetzwerk aus*
 - Sloman, Kramer (1989)

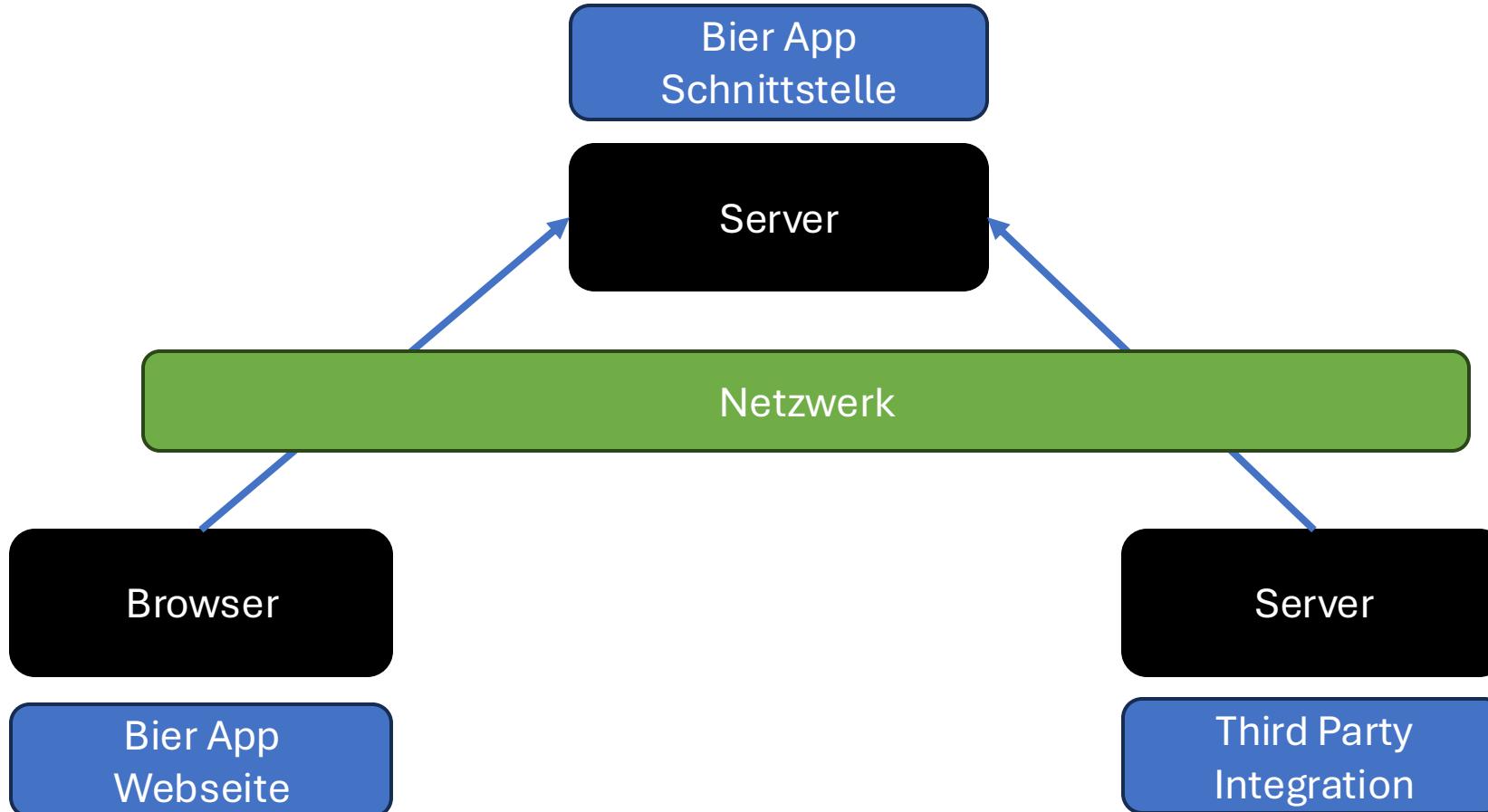
Verteilte Systeme

Eigenschaften

- Mehrere durch ein Netzwerk verbundene Systeme (= Computer), welche unabhängig voneinander ausfallen können
- Arbeiten ko-operativ für ein gemeinsames Ziel bzw. gemeinsamer Aufgabe z.B. Bereitstellung einer Dienstleistung
- Tauschen Nachrichten untereinander aus

Verteilte Systeme

Bier App – Lösungsdesign



Verteilte Systeme

Weitere Beispiele

- E-Mail Client und E-Mail Server mit Postfach
- Heizungssteuerung mit Kontroll-Einheit im Wohnbereich
- Auto mit verschiedenen Steuerelementen
- Geldautomat und zentraler Bankserver
- Smartphone-App und Rechenzentrum
- Smartwatch und Smartphone
- Industrieanlagen und Steuerung dieser Anlagen
- Internet World / Wide Web

Verteilte Systeme

Herausforderungen

Herausforderung	Erklärung	Beispiel Geldautomat
Ausfall	Netzwerkverbindung kann ausfallen oder fehleranfällig sein	Keine Verbindung zum zentralen System
Verbindungsaufbau	Aufbau und Ablauf der Netzwerkverbindung	Zwei Geldautomaten zum Geld abheben genutzt
Asynchronität	Asynchrone Ausführung durch Verzögerungen	Lokaler Händler bucht Geld später ab
Protokoll	Gemeinsames Protokoll zur Kooperation und Datenaustausch	Kommunikation der Zahlungsteilnehmer oder Hersteller der Automaten
Zuverlässigkeit	Toleranz bei Ausfall oder Skalierungsproblemen	Verfügbarkeit wenn ein Automat oder eine ganze Region ausfällt

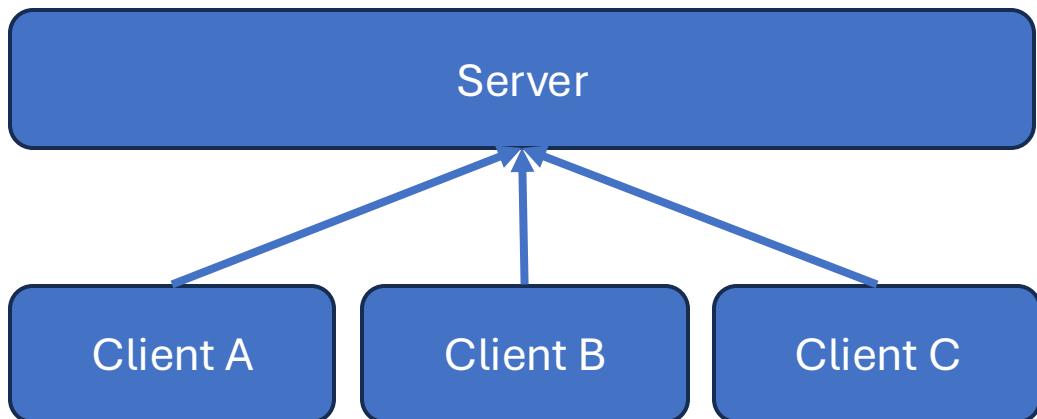
Verteilte Systeme

Zusammenfassung

- Verteilte Systeme sind verbundene Systeme, welche miteinander an einer Aufgabe arbeiten jedoch unabhängig voneinander ausfallen können
 - Bspw. Ein zentrales Banksystem mit Geldautomaten, Smartwatch und Smartphone
- Verteilte Systeme haben spezifische Herausforderungen
 - z.B die Kommunikation und Ausfallsicherheit

Verteilte Systeme

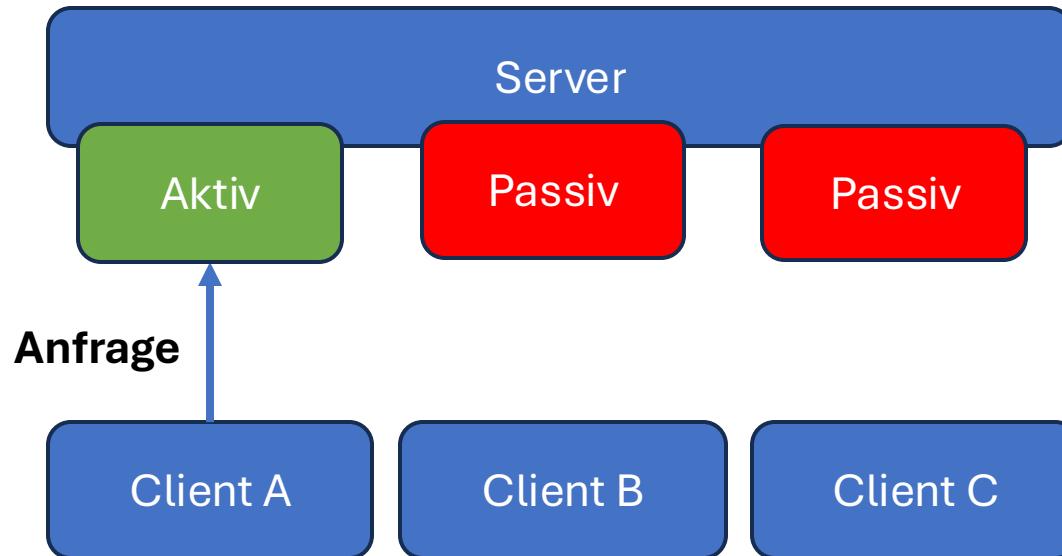
Client-Server Modell



- **Server** = Programm zur Bereitstellung einer Ressource oder eines Services für einen oder mehrere Clients
- **Client** = Programm die Ressource oder Service vom Server anfordert
- Verbunden in einem Netzwerk
- Anfrage des Clients ist ein **Request**, Antwort des Servers eine **Response**

Verteilte Systeme

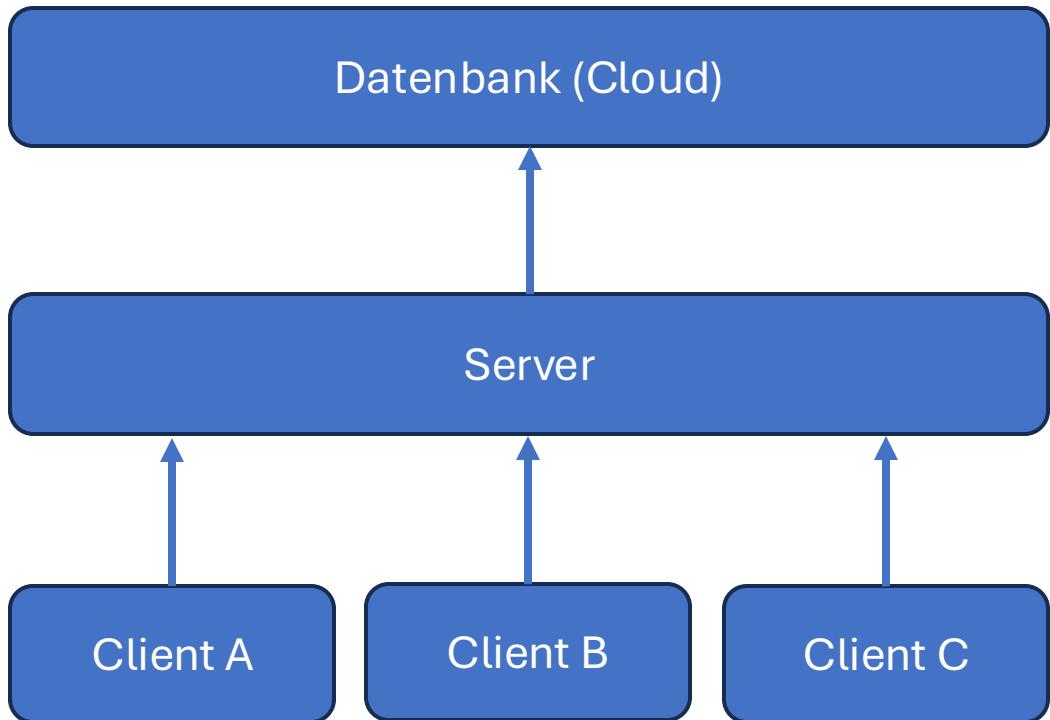
Client-Server Modell



- Der Server ist immer passiv – Der Client immer aktiv
 - Ein Client muss aktiv eine Antwort (Response) vom Server anfordern (Request)
 - Ohne einen Request, bleibt der Server inaktiv
- Die Rollen sind klar definiert
 - Server und Client sind getrennt

Verteilte Systeme

Client-Server Modell



- Ein Server kann aber gleichzeitig für einen anderen Service / Ressource Client sein
- Bspw. Abfrage an einen Datenbank Server

Verteilte Systeme

Client-Server Modell – Kommunikation

- Für die Kommunikation zwischen Client und Server können unterschiedliche Protokolle zum Einsatz kommen
- Im World Wide Web bspw. wäre es HTTP (Später mehr)
- Server und Clients tauschen Nachrichten über HTTP miteinander aus

Verteilte Systeme

Architektur für verteilte Systeme

- Das WWW ist das Größte und erfolgreichste verteilte System
- Wir nutzen Web-Technologien, um verteilte Systeme zu bauen
 - Standards, Protokolle und Technologien
 - Architekturen und Patterns
- Das reduziert das Risiko in der Entwicklung und führt zu besseren Ergebnissen

Verteilte Systeme

Architektur für verteilte Systeme

- *Webdienste können durch eine Sammlung von Standards leicht kommunizieren*
 - Booth et al. (2004); Tannenbaum, van Steen (2001)
- HTTP ermöglicht Kommunikation in einem Netzwerk
- Standards erfüllen Basis Anforderungen verteilter Systeme
- Hoher Verbreitungsgrad ermöglicht Integration verschiedener Webdienste

Verteilte Systeme

Architektur für verteilte Systeme

- *Webdienste zur Bereitstellung von verteilten Anwendungen*
 - Alonso et al. (2004)
- Bereitstellung von beliebigen Anwendungen, auch außerhalb des Browsers
- Bereitstellung von Schnittstellen (Daten)

Verteilte Systeme

Webtechnologien lösen Herausforderungen

Herausforderung	Lösungsansatz	Beispiel
Ausfall	Das WWW als dezentrales System hat viele Server und Schichten zur Redundanz	Webserver sind schnell entwickelt und es gibt viele Anbieter
Verbindungsaufbau	Gemeinsames Protokoll	HTTP auf TCP/IP Basis
Koordination	Logik der Koordination erfolgt auf dem jeweiligen Server	HTTP ist Zustandslos
Asynchronität	Anfragen können parallel gesendet werden	HTTP ist Zustandslos
Protokoll	Wir sprechen über HTTP zwischen Diensten	HTTP kann beliebige Datenformate übertragen (Video, JSON, HTML etc.)
Zuverlässigkeit	Protokolle sind einfach, robust und erprobt	Schwache Server können bereits Tausende Anfragen pro Sekunde Verarbeiten

Verteilte Systeme

Bier App – Als Verteiltes System mit Web Technologien

Anforderung	Umsetzung Eigenentwicklung	Umsetzung Web
Kommunikation zwischen Webseite und Server	TCP Verbindung über ein Netzwerk mit eigenem Protokoll für den Austausch von Daten	HTTP Server, Client und Server tauschen über HTTP Daten aus
Server Entwicklung	Spezifikation der Protokolle, Umsetzung auf Netzwerkebene, Low Level Implementierung	HTTP Bibliotheken verfügbar für jede Programmiersprache
Anbindung an andere Dienste	Andere Dienste müssten mit unserem Protokoll interagieren und integriert werden	Können gleichen HTTP Server ansprechen und über HTTP Daten austauschen

Verteilte Systeme

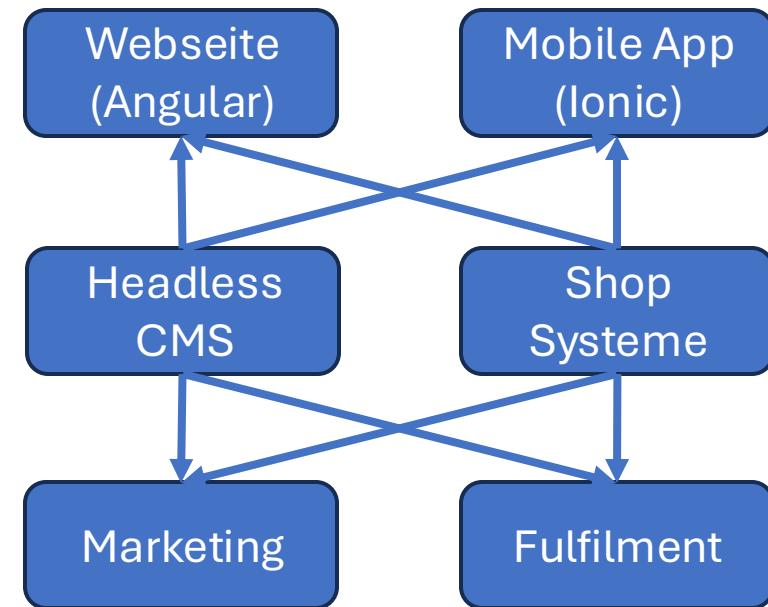
Bedeutung der Web Programmierung

- Webanwendungen – nicht nur Webseiten – sind fester Bestandteil unseres Alltags mit extremer Bedeutung
- Die erprobten Technologien und Architekturen des Webs werden in der Entwicklung von verteilten Systemen allgemein genutzt

Verteilte Systeme

Beispiele – Onlineshop mit Mobiler App

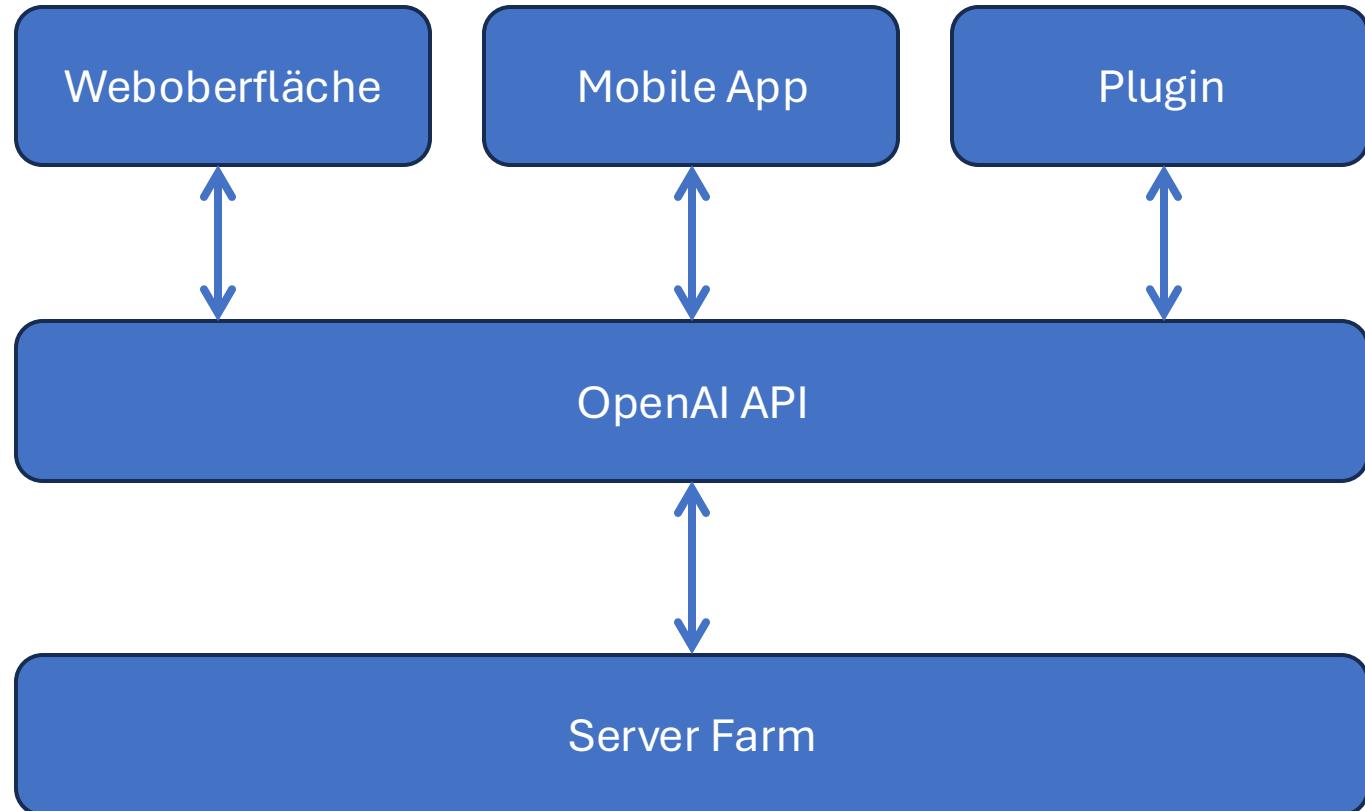
- Webseite spricht mit Content und Shopsystem
- Mobile App ruft Bestellstatus vom Shopsystem ab
- Übergabe Versandinformationen an Logistikdienstleister
- Verarbeitung von Emails zur Werbung über externen Marketing Dienstleister
- Tracking des Kunden für personalisierte Werbung



Verteilte Systeme

Beispiel – ChatGPT

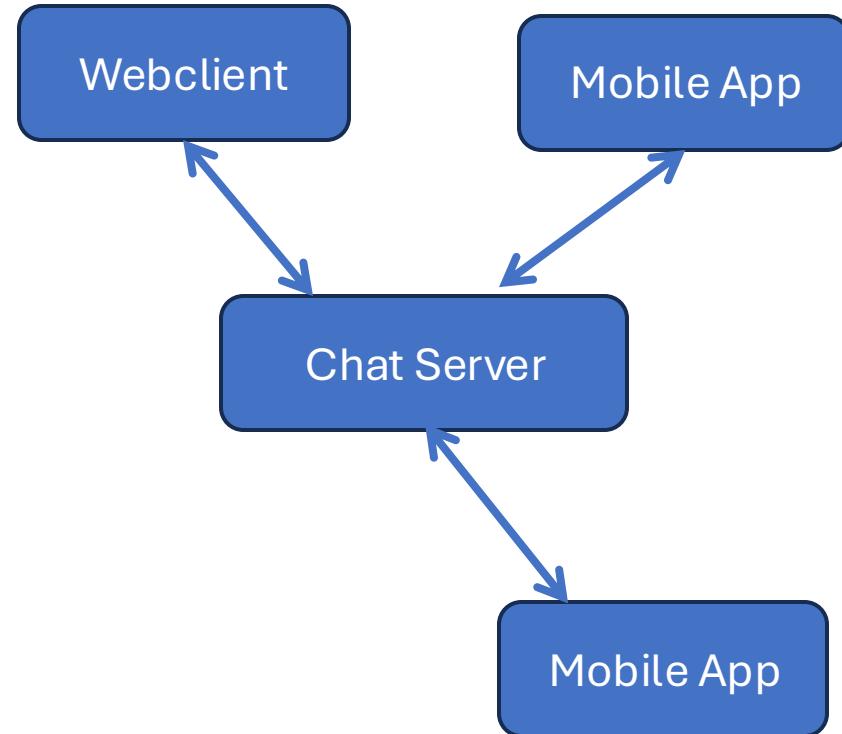
- Anwender interagiert mit Chat auf Webanwendung (oder Mobiler App)
- Streaming der Modell Antwort an Anwender
- Aufruf von Plugins
- Aufruf von Webseiten für Suchen



Verteilte Systeme

Beispiele – Chat

- Web Interface im Browser oder Mobiler App
- Server zum Austausch der Nachrichten innerhalb eines Servers oder mit anderen Systemen



Verteilte Systeme

Wann machen Web Technologien wenig Sinn?

- Hohes Maß an Integration wenn ich die Kontrolle habe
- Performance und Latenzen
- Ressourcen und Umfeld nicht darauf ausgelegt
- Perfekte native Ergebnisse erforderlich sind
- Definierte Anforderungen und Erfahrungen

Verteilte Systeme

Bier App – Lösungsdesign mit Web Technologien

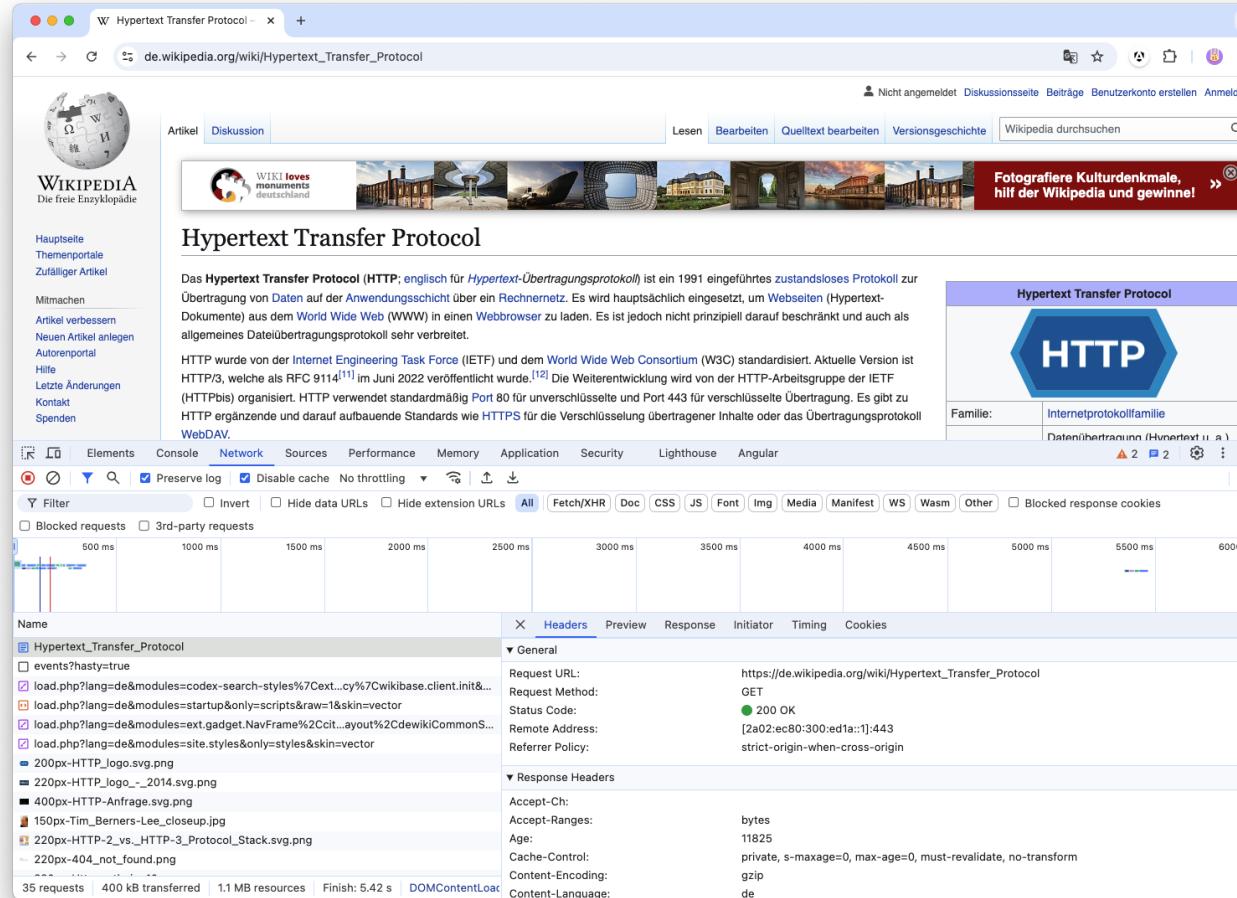
- Wir werden die Daten als HTTP-Schnittstelle über einen NodeJS Server bereitstellen
- User können über eine Webanwendung mit dem Server interagieren
- Webseite und Server kommunizieren mit dem HTTP-Protokoll
- Andere Dienste können auf dieselbe HTTP-Schnittstelle zugreifen

HTTP

Definition und Überblick

HTTP

Im Alltag



HTTP

Einordnung

- Protokoll zur Übertragung von Daten auf Anwendungsebene
 - ISO-Layer 5-7
 - Basierend auf TCP-Kommunikation
- Ermöglicht Kommunikation in einem verteilten System
 - Einsatz im „größten“ verteilten System World Wide Web
 - Definiert Regeln und „Sprache“ beim Informationsaustausch

HTTP

Einordnung

- Offener, freier und kostenloser Standard
 - Forschungsprojekt, erfunden von Tim Berners-Lee 1989 am CERN Institut (Schweiz)
 - Standardisiert von der Internet Engineering Task Force (IETF, RFC1945) und dem World Wide Web Consortium (W3C)
- [RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0 \(ietf.org\)](https://www.ietf.org/rfc/rfc1945.txt)

HTTP

Verbreitung

- Es gibt ca. (Stand 2018!)
 - 5 Milliarden WWW-fähige Endgeräte
 - 1,6 Milliarden Webseiten
 - 110 Millionen physikalische Webserver die über das Internet erreichbar sind
- Offene Standards und einfache Implementierung
 - In C mit ca. 150 Zeilen (mit Standard-Libraries) funktionsfähig
 - Unter 30 Bytes im Arbeitsspeicher
 - Hohe Performance: 10T gleichzeitige Verbindungen mit >500kb Speicher (LWAN)

HTTP

Verbreitung

- Das World Wide Web basiert auf dem HTTP-Protokoll und einer Client-Server-Architektur.
- Technologien in der Web-Programmierung wandeln sich schnell. Kernkonzepte bleiben gleich.

HTTP

Einführung

- HTTP steht für HyperText-Transfer-Protocol
- Anwendungsprotokoll in OSI-Layer 5-7, basierend auf TCP
- HTTP ist text-basiert* und somit zunächst unverschlüsselt („Klartext“)
 - Jedoch mit Erweiterung durch SSL zur verschlüsselten Kommunikation zwischen Client und Server geeignet (“HTTPS”)
 - *... und das ist Teil der Sicherheits-Vorlesungen später.*
- Wir betrachten in der Vorlesung HTTP/1.1

HTTP

Einführung

- HTTP ist eine Sprache um Informationen zwischen einem Client (Browser) und einem Server auszutauschen
 - Grundlage für das World Wide Web
- Protokoll für das Laden von verschiedenen Ressourcen (z.B. HTML Dokumenten, Bildern, Videos etc.)
- Client und Server kommunizieren über abgeschlossene Nachrichten (Kein Stream an Daten)
 - Client schickt eine Anfrage und bekommt genau eine Antwort

HTTP

Definition

- HTTP ist **nachrichtenbasiert**.
 - Jede Anfrage zwischen Client und Server hat ein definiertes Schema aus Header und Body
 - Bestimmte Informationen müssen enthalten sein und vom Server/Client bereitgestellt werden
- HTTP ist ein **zustandsloses** Protokoll
 - Informationen aus vorherigen Anfragen gehen nach jeder Anfrage verloren. Jede Anfrage ist „neu“ und unabhängig von anderen Abfragen.
 - Sollen Informationen zwischen Anfragen konsistent bleiben benötigt man einen Identifier, z.B. eine Session-Identifier. Dies ist Aufgabe der Anwendungsebene

HTTP

Definition

- HTTP kann **erweitert** und angepasst werden (Nicht nur für „Hypertext“)
 - Transfer von beliebigen Datenarten möglich, z.B. Bilder
 - Beispiel: WebDAV für Dateiaustausch, GIT für Quellcode-Tausch

HTTP

Ablauf

- Eine TCP Verbindung mit einem Server wird hergestellt
 - Die selbe TCP Verbindung kann für mehrere Anfragen verwendet werden
 - Parallelle TCP Verbindungen sind möglich
- Eine HTTP Nachricht wird vom Client an den Server übermittelt
- Die Nachricht besteht dabei aus:
 - Anfragemethode
 - Pfad
 - HTTP Version
 - Host Adresse
 - Beliebige weitere Felder

HTTP

Ablauf



```
* GET http://www.heidenheim.dhbw.de/
GET / HTTP/1.1
User-Agent: PostmanRuntime/7.29.2
Accept: */*
Postman-Token: a49c4c3b-d277-48fe-8bae-ad21dd602fe8
Host: www.heidenheim.dhbw.de
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

HTTP/1.1 301
Date: Fri, 30 Sep 2022 15:12:46 GMT
Server: Apache
Location: https://www.heidenheim.dhbw.de/
Content-Length: 239
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1
```

```
* GET https://www.heidenheim.dhbw.de/startseite
GET / HTTP/1.1
User-Agent: PostmanRuntime/7.29.2
Accept: */*
Postman-Token: f34affa1-85b9-4619-bd1f-50d6b462bda9
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: https://www.heidenheim.dhbw.de/
Host: www.heidenheim.dhbw.de

HTTP/1.1 200 OK
Date: Fri, 30 Sep 2022 15:14:44 GMT
Server: Apache
Content-Language: de
Content-Length: 72858
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

HTTP

Ablauf

- Antwort des Webservers bestehend aus Header für allgemeine Informationen und Body der den gewünschten Inhalt (Hier HTML Dokument) enthält
- Header Informationen
 - Protokoll (HTTP/1.1) und Status (200 OK)
 - Größe und Format der Antwort
 - Beliebe weitere Felder
- Body Informationen
 - Reines HTML (Kann vom Browser dann angezeigt werden)

HTTP

Ablauf

```
<!DOCTYPE html>
<html dir="ltr" lang="de-DE" class="no-js">
<head>

<meta charset="utf-8">
<!--
=====
Umsetzung und Hosting: cron IT GmbH, www.cron.eu =====

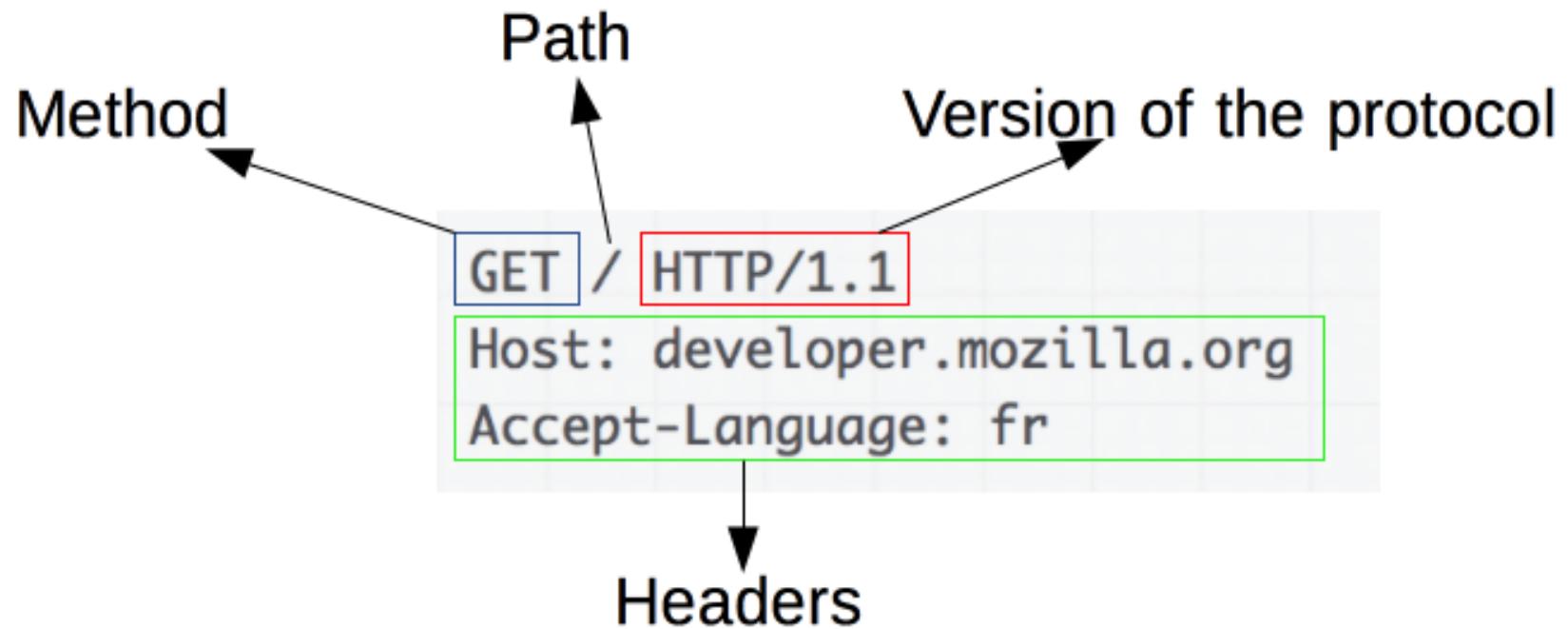
This website is powered by TYPO3 - inspiring people to share!
TYPO3 is a free open source Content Management Framework initially created by Kasper Skaarhoj and licensed under GNU/GPL.
TYPO3 is copyright 1998-2022 of Kasper Skaarhoj. Extensions are copyright of their respective owners.
Information and contribution at https://typo3.org/
-->
|



<title>Hochschule in Ostwürttemberg | DHBW Heidenheim</title>
<meta name="generator" content="TYPO3 CMS" />
<meta name="description" content="Dual studieren mit Aussicht - an der DHBW Heidenheim in 23 Studiengängen mit top Karrierechancen & monatlichem Gehalt. ►Jetzt informieren" />
<meta name="keywords" content="DHBW Heidenheim, Duales Studium, Duale Hochschule Baden Württemberg, Theorie und Praxis, Heidenheim, duale Bachelorstudiengänge, duale Studiengänge, praxisnah, Studium, Studieren, Studieren und Geld verdienen, DH Heidenheim, Dualer Student, Student, Hochschule, Hochschule in Ostwürttemberg, Ostwürttemberg, Bachelor, dual studieren, studieren in heidenheim, DHBW hdh, was ist ein duales Studium, studieren in Baden-Württemberg" />
<meta property="og:title" content="Hochschule in Ostwürttemberg" />
<meta property="og:description" content="Die DHBW Heidenheim ist die Hochschule in Ostwürttemberg, die Geld verdienen und Studentenleben vereint." />
<meta property="og:image" content="https://www.heidenheim.dhbw.de/fileadmin/_processed_/4/a/csm_Startseite_Bewerberboerse_mobil_ffe0be19e8.jpg" />
<meta property="og:image:url" content="https://www.heidenheim.dhbw.de/fileadmin/_processed_/4/a/csm_Startseite_Bewerberboerse_mobil_ffe0be19e8.jpg" />
<meta property="og:image:width" content="640" />
<meta property="og:image:height" content="335" />
<meta property="og:image:alt" content="Zwei Studenten sitzen auf einer Mauer und warten auf ihre nächste Vorlesung an der DHBW Heidenheim." />
<meta name="twitter:card" content="summary" />
<meta name="twitter:title" content="Duales Studium an der DHBW Heidenheim" />
```

HTTP

Request Aufbau



Quelle: MDN Web Docs – An overview of HTTP (2022)

HTTP

Request Aufbau

- **HTTP Methode** – Legt fest welche Operation ein Client ausführen möchte
- **Pfad** – Die URL an welcher Stelle eine Ressource zu finden ist
- **Version** – Die HTTP Protokoll Version
- **Headers** – Optionale Informationen die an den Server übermittelt werden

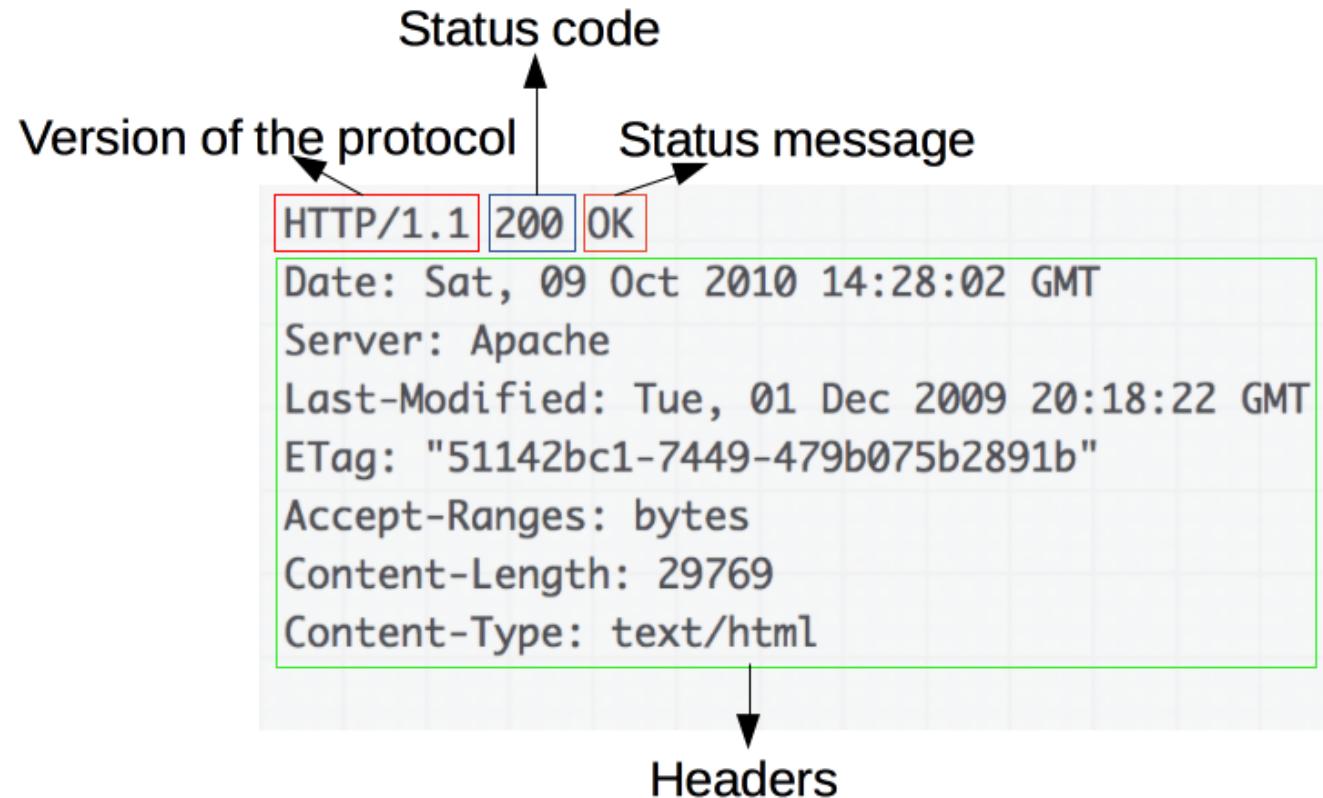
HTTP

Request Aufbau

- **Body** – Einige HTTP Methoden (z.B. POST) übermitteln zusätzliche Informationen
 - Bspw. Sie haben auf einer Webseite ein Formular ausgefüllt

HTTP

Response Aufbau



Quelle: MDN Web Docs – An overview of HTTP (2022)

HTTP

Response Aufbau

- **Version** – Die HTTP Protokoll Version
- **Status Code** – Auskunft darüber ob eine Anfrage erfolgreich war oder nicht (Und warum)
- **Status Message** – Eine kurze Beschreibung für den Status Code
- **Headers** – Optionale Informationen (vgl. mit Request)

HTTP

Response Aufbau

- **Body** – Beinhaltet die Angefragte Ressource (z.B. HTML Dokument)
 - Responses können auch leere Bodys zurück senden bspw. wenn eine Anfrage nicht erfolgreich war

HTTP

Methoden

- Es gibt verschiedene HTTP-Methoden, für verschiedene Arten von Anfragen
- Server kann entsprechend darauf reagieren und anders Verhalten
 - Reine Textangabe in der Afrage. Technisch kein naderes Protokoll

HTTP

Methoden

- HTTP Methoden (Auswahl)
 - **GET** = Fragt eine Ressource unter einer URI ab.
 - Daten sollten dabei nicht verändert werden, die Methode muss „Sicher“ im Aufruf sein
 - **POST** = Übermittelt Daten zur Verarbeitung. Daten sind im BODY der Anfrage
 - **HEAD** = Frägt nur den Header, jedoch nicht den Body ab
 - **PUT / PATCH** = Legt neue Daten an bzw. verändert bestehende Daten
 - **DELETE** = Soll Daten löschen

HTTP

Status Codes

- HTTP Response Status Codes signalisieren dem Client, ob eine HTTP Anfrage Erfolgreich war oder nicht.
- HTTP Status Codes können in 5 Kategorien unterteilt werden
 - Definition von allen Status Codes [RFC 9110 - HTTP Semantics \(httpwg.org\)](https://httpwg.org/rfc9110.html)

HTTP

Status Codes

- 100 – 199 = Informational Responses
- 200 – 299 = Erfolgreiche Responses
- 300 – 399 = Redirection Messages
- 400 – 499 = Client Error Responses
- 500 – 599 = Server Error Responses

HTTP

Status Codes



- Die wichtigsten Status Codes im Überblick
 - 200 – OK
 - 201 – Created
 - 301 – Moved Permanently
 - 400 – Bad Request
 - 401 – Unauthorized
 - 403 – Forbidden
 - 404 – Not Found
 - 500 – Internal Server Error
 - 503 – Service Unavailable

HTTP

Übung

- Welche HTTP Methode und welchen Status Code würden Sie für folgende Anfragen verwenden?
 - Als Benutzer möchte ich ein neues Benutzerkonto auf einer Webseite erstellen
 - Als Benutzer möchte ich die Hauptseite einer Webseite anzeigen
 - Als Benutzer möchte ich zur neuen Adresse der Webseite weitergeleitet werden, wenn die Seite umgezogen ist
 - Als Benutzer, der sein Passwort vergessen hat, möchte ich mein Passwort zurücksetzen
 - Als Benutzer möchte ich Zugang zu einem bestimmten Dokument erhalten, ich bin bereits eingeloggt

HTTP

Übung

- Welche HTTP Methode und welchen Status Code würden Sie für folgende Anfragen verwenden?
 - Als Benutzer möchte ich Zugang zu einem bestimmten Dokument erhalten, ich bin nicht eingeloggt
 - Als Benutzer möchte ich Zugang zu einem bestimmten Dokument erhalten, ich bin aber kein Admin
 - Als Benutzer möchte ich eine Ressource abrufen, die nicht mehr existiert
 - Als Benutzer möchte ich einen Fehler angezeigt bekommen, wenn der Server einen Fehler macht

HTTP

URI und URLs

- Wie navigieren wir zwischen Ressourcen im Web?
- Uniform Ressource Identifier = Kompakte Sequenz von Zeichen die abstrakte oder physische Ressource beschreiben
- Eine URI tritt in Erscheinung als ..
 - Uniform Ressource Name = Der Name der Ressource („Das Haus der Maiers“)
 - Uniform Ressource Location = Die Adresse der Ressource („Gabelweg 2“)
 - .. Oder als beides

HTTP

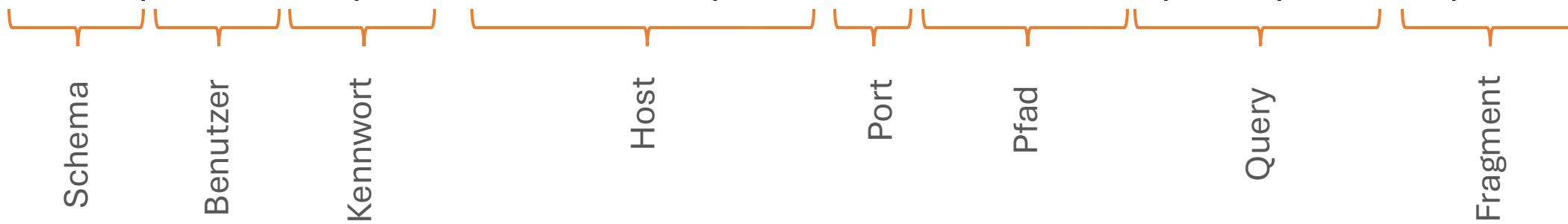
URI und URLs - Beispiele

- ISBN = Beschreibt ein bestimmtes Buch, aber nicht wo es zu finden ist (URN)
- Web-Adresse bzw. Pfad = Beschreibt wo eine Webressource zu finden ist (URL)

HTTP

Aufbau von URLs im Web

- `https://muster:pwd123@www.example.com:1337/index.html?p1=A&p2=B#chapter-1`



- Schema muss nicht unbedingt gleich dem technischen Protokoll im Netzwerk sein
- Benutzer und Kennwort sind nicht Teil der HTTP-Spezifikation (1). Meisten Browser akzeptieren jedoch diese Schreibweise
- Der “Query” Teil wird meist auch als “GET-Parameter” angelehnt an die HTTP-Methode bezeichnet

HTTP

Übung

- Öffnen Sie das Projekt 02-http-explorer im Ordner 30-miscellaneous in den Vorlesung Unterlagen
- Öffnen Sie das Integrierte Terminal im Ordner 02-http-explorer
 - Tipp: Rechtsklick auf den Ordner -> Open in integrated Terminal
- Tippen Sie den Befehl **npm install** ein und bestätigen Sie mit enter

```
~/Code/DHBW/dhbw-webprogrammierung-wwi/30-miscellaneous/02-http-explorer on ✘ master! ┈ 20:51:32
$ npm i

added 56 packages, and audited 57 packages in 851ms
6 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

~/Code/DHBW/dhbw-webprogrammierung-wwi/30-miscellaneous/02-http-explorer on ✘ master! ┈ 20:53:53
$ ┌
```

HTTP

Übung

- Tippen Sie anschließend den Befehl **npm run start** im selben Terminal Fenster ein und bestätigen Sie mit enter

```
~/Code/DHBW/dhbw-webprogrammierung-wwi/30-miscellaneous/02-http-explorer on ✘ master! 20:53:53
$ npm run start

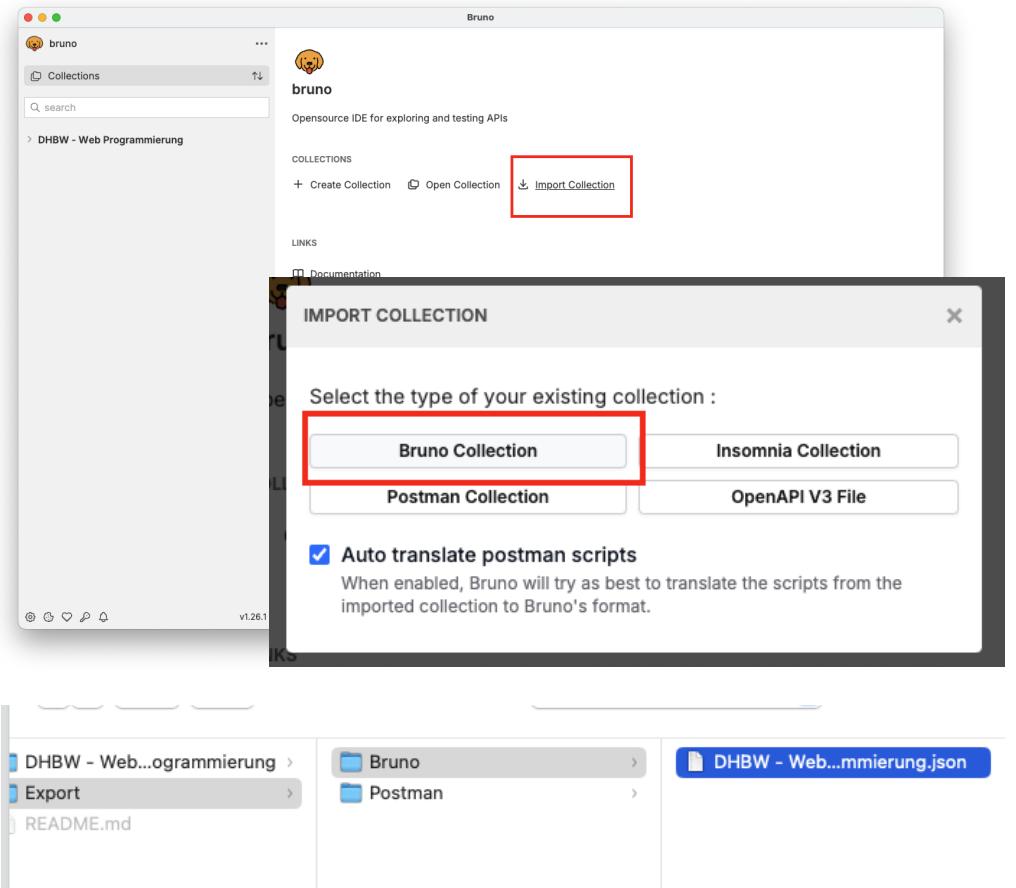
> dhbw-webprogrammierung-wwi-http-explorer@1.0.0 start
> node server.js

{"level":30,"time":1728932327795,"pid":74496,"hostname":"Jenss-MBP","msg":"Server listening at http://[::1]:8080"}
{"level":30,"time":1728932327796,"pid":74496,"hostname":"Jenss-MBP","msg":"Server listening at http://127.0.0.1:8080"}
```

HTTP

Übung

- Importieren Sie sich die Bruno Collection (Alternativ mittels Postman)
- Wählen Sie die .json Datei im Ordner 20-Bruno-Collections Export aus
- Wählen Sie einen Speicherort auf Ihrem System aus
 - Klicken Sie anschließend auf Import



HTTP

Übung

- Öffnen Sie den Ordner 30-miscellaneous und anschließend 02-http-explorer und versuchen Sie die unterschiedlichen HTTP Requests aus
- Modifizieren Sie auch den Inhalt der Anfragen – Der Server sollte Ihnen Tipps darüber geben

HTTP

Zusammenfassung

- HTTP ist ein Protokoll für den Austausch von Informationen und ist der Standard für die Kommunikation im World Wide Web
- HTTP ist Zustandslos und definiert Methoden, um auf Ressourcen zuzugreifen
- Die HTTP Kommunikation besteht immer aus einem Header (Mit Statuscode) und ggf. einem Body
- Ressourcen im Internet werden immer über URLs identifiziert

Browser

Software Client für das WWW

Browser

Einführung

- Browser = Software Client für HTTP und Rendering von HTML-Inhalten
 - (und noch viel mehr ...)
- Ermöglicht Navigation zwischen Webseiten, Kommunikation der Nutzereingabe mit einem Webserver
- Vereinfach die Komplexität des verteilten Systems
 - Bspw. Führt alle 200 Anfragen zur korrekten Darstellung von wikipedia.org aus

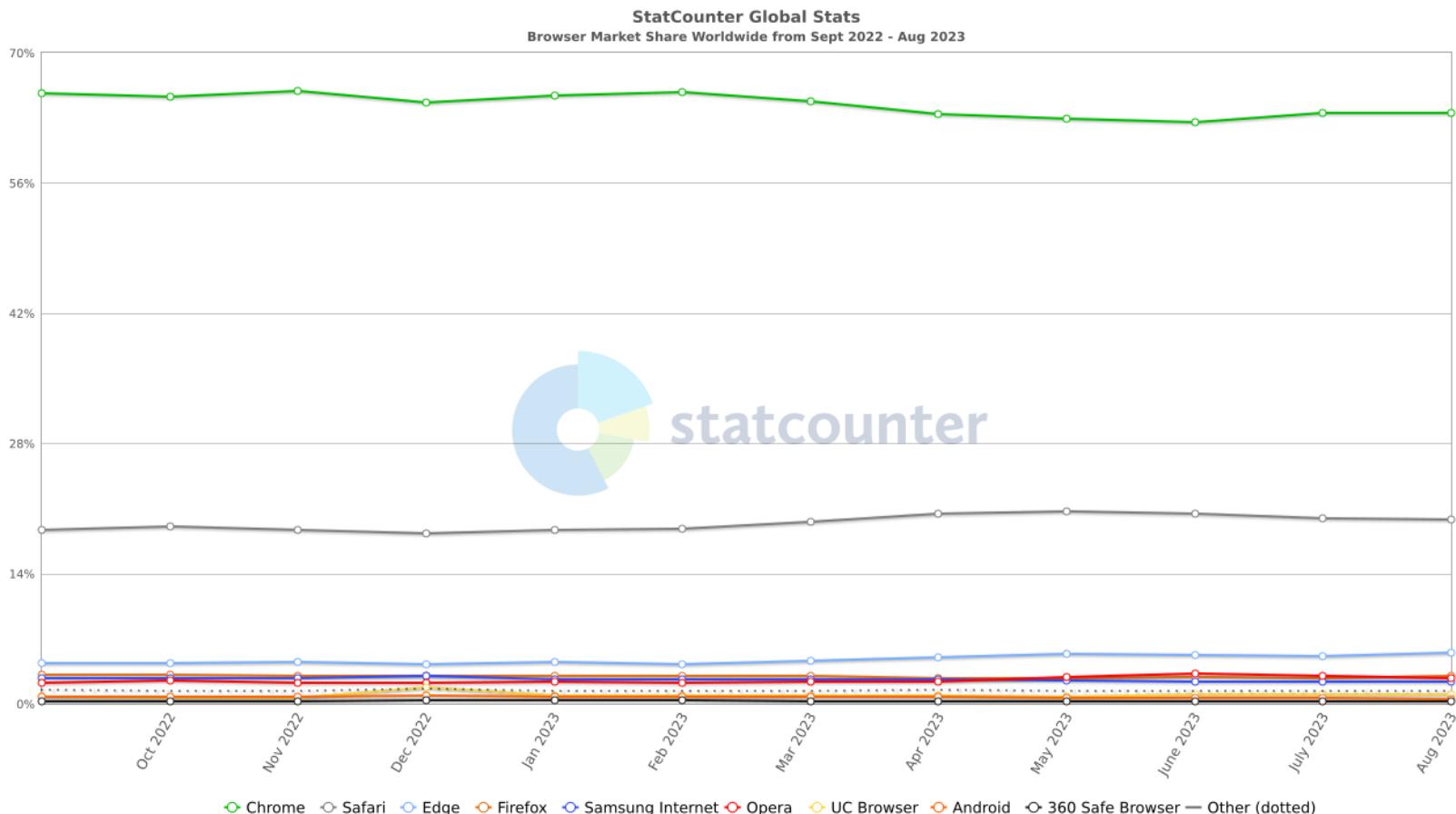
Browser

Einführung

- Rendert die Antwort des Servers, führt JavaScript Code aus, zeigt Bilder an und löst verknüpfte Dokumente auf
 - Verfolgt Links z.B. zu Stylesheets oder anderen Ressourcen automatisch
 - Kann durch Plugins erweitert werden, z.B. Anzeige von PDFs
- Verschiedene Versionen sind auf dem Markt verfügbar
 - Mobil und für den Desktop

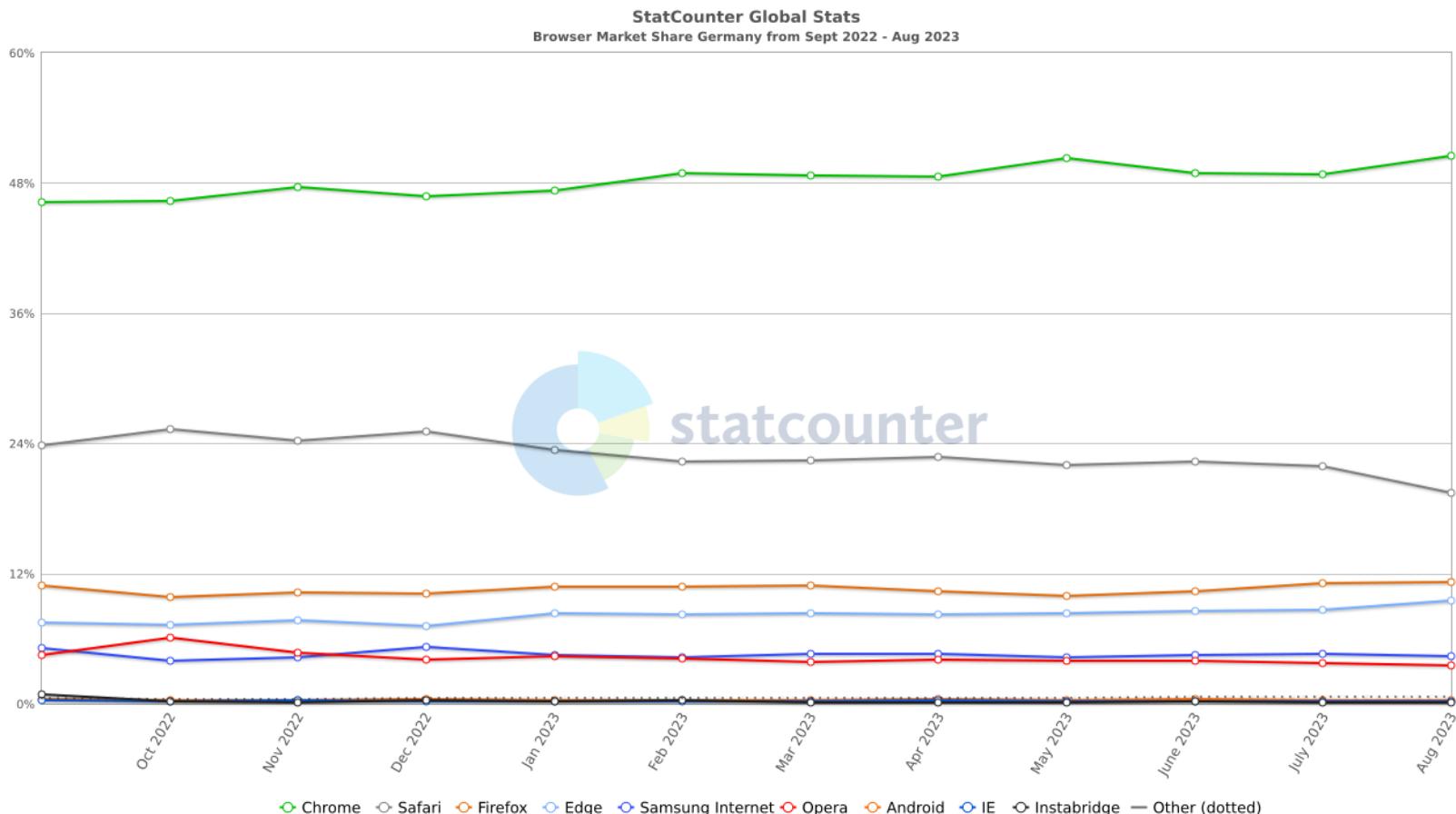
Browser

Verbreitung (Weltweit)



Browser

Verbreitung (Deutschland)



Browser

Bedeutung

- Ermöglichen einfachen Zugang zum verteilten System „World Wide Web“
 - Auf jedem Endgerät
 - Für Milliarden von Anwendern
- Große Verbreitung einzelner Hersteller (bspw. Chromium Engine)
- Können damit neue Standards definieren und beeinflussen
 - Google Chrome Manifest V3

Browser

But for Data Exchange?

- Wie können Clients untereinander kommunizieren?
- Gibt es einen Standard wie Clients Daten austauschen?
- HTTP überträgt nicht nur HTML Dokumente
- HTTP kann von jedem Software System eingesetzt werden
 - Nicht nur Browsers bspw. auch Maschinen sind mittlerweile HTTP enabled

HTTP

More than just HTML

- Für Menschen werden Inhalte in HTML übertragen und dann vom Browser korrekt visuell aufbereitet und dargestellt
- Für andere Systeme müssen Daten in einem anderen Format strukturiert zur Verfügung gestellt werden
 - HTML ist zwar maschinen-lesbar, jedoch nicht optimiert dafür
 - Enthält viele Informationen für die Darstellung

HTTP

More than just HTML

- Per HTTP können beliebige Datenformate übertragen werden und je nach Anwendungsfall können über einen HTTP Header das Format angefragt werden
 - Beispiel: XML, JSON
 - Nicht jeder Server unterstützt jede Content Type Anfrage
- Wir nutzen HTTP zur Kommunikation zwischen Systemen
 - Unsere Bier Schnittstelle kann Daten für ein anderes System als JSON bereitstellen

HTTP

Ressourcen im Browser abrufen

Vom Browser angezeigt

GET /books/restful.html

RESTful Web APIs

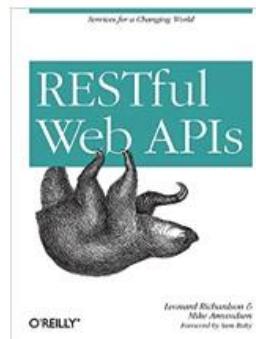
Autor: **Leonard Richardson**

Erscheinungsjahr: **2013**

Seiten: **404**

O'Reilly and Associates

ISBN-13: **978-1449358068**



Als HTML

GET /books/restful.html

```
<!-- ID: 42 -->
<h1>RESTful Web APIs</h1>
<img src=„./cover-42.jpg“ alt=„Cover“
style=„float:right;“>
<p>Autor: Leonard Richardson<p>
<p>Erscheinungsjahr: 2013</p>
<p>Seiten: 404</p>
<p><small>O'Reilly and Associates</small></p>
<p> ISBN-13: 978-1449358068 </p>
```

HTTP

Ressourcen in anderen Datenformaten

Als XML

GET /api_xml/books/42

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <id>42</id>
  <coverUrl>/covers/42.jpg</coverUrl>
  <author>Leonard Richardson</author>
  <pageCount>404</pageCount>
  <publishYear>2013</publishYear>
  <publisher> O\'reilly and Associates</publisher>
  <isbn13>978-1449358068</isbn13>
</book>
```

Als JSON

GET /api_json/books/42

```
{
  "id": 42,
  "coverUrl": "/covers/42.jpg",
  "author": "Leonard Richardson",
  "pageCount": 404,
  "publishYear": 2013
  "publisher": "O\\'reilly and Associates",
  "isbn13": "978-1449358068",
}
```

HTTP

Bier App – Biere über HTTP austauschen

- Wir möchten jetzt mit NodeJS und JavaScript einen Webserver implementieren, der über HTTP-Methoden unsere zuvor implementierten Funktionen anbietet
- Über einen Client können wir dann Biere anlegen, Bier Informationen abfragen etc.

Webserver Development

Der erste Webserver mit NodeJS

Webserver Development

Definition

- Als Webserver bezeichnen wir eine Software, welche Anfragen über HTTP entgegennimmt, um Dokumente für Clients bereitzustellen
- In den meisten Fällen ist der Client ein Webbrowser und Dokumente typische Webinhalte (HTML, Bilder, Videos etc.)
- Der Webserver kann diese Daten über ein Netzwerk (Lokal oder WWW) an unterschiedliche Clients ausliefern

Webserver Development

Aufgaben

- HTTP-Anfragen von einem oder mehreren Clients entgegen nehmen
- HTTP-Anfrage verarbeiten und statische Inhalte als Antwort zurückgeben

Webserver Development

Hintergrund

- Wir wollen jetzt gemeinsam unseren ersten Webserver mit NodeJS implementieren, um die Konzepte von HTTP und Client Server Interaktion näher zu bringen

Webserver Development

NodeJS HTTP Module

- NodeJS bietet ein HTTP Module welches eine Low Level API für die Interaktion mit dem HTTP Protokoll bereitstellt
- Low Level bedeutet es übernimmt nur das Streaming und Message Parsing in einen Header und einen Body
- Wir betrachten in der Vorlesung nicht die komplette Low Level API sondern nur einen Teil für das Erstellen von einem HTTP Server
 - Die Gesamte Dokumentation finden sie aber bspw. hier <https://nodejs.org/api/http.html>

Webserver Development

HTTP Server mit NodeJS



[Github](#)

```
1 import http from "node:http";
2
3 const server = http.createServer();
4
5 server.on("request", (request, res) => {
6   res.writeHead(200, { "content-type": "application/json" });
7   res.end(
8     JSON.stringify({
9       data: "Hello World!",
10      })
11    );
12  });
13
14 server.listen(8080);
```

Webserver Development

HTTP Server mit NodeJS

- Wir importieren zuerst das **HTTP-Module** von **node:http**
 - Hinweis: Wir verwenden das ESM Format und speichern die Datei als .mjs
- Mit der **createServer** Methode erstellen wir uns eine Server Instanz
 - Siehe <https://nodejs.org/api/http.html#httpcreateserveroptions-requestlistener>
- Mit einer Server Instanz können wir auf Events hören die NodeJS für uns bereit stellt – In diesem Fall möchten wir auf **request** Events hören also reagieren wenn eine Anfrage an unseren Server gestellt wird

Webserver Development

HTTP Server mit NodeJS

- Das **request** Event übergibt uns zwei Parameter:
 - Ein request Objekt welche Informationen über die HTTP-Anfrage beinhaltet
 - Ein response Objekt welches intern von unserem HTTP-Server erstellt wird.
- Mit dem **response** Objekt können wir jetzt die HTTP-Antwort erstellen
 - Mit der Methode **writeHead** definieren wir den HTTP Status Code und fügen noch einen **content-type** Header hinzu
 - Mit der Methode **end** können wir dann noch Daten im Body schicken. In unserem Beispiel möchten wir ein JSON an den Client zurück geben
- Mit der **end** Methode terminiert der HTTP-Server intern auch die Verbindung mit dem Client und wir können keine weiteren Daten schicken

Webserver Development

HTTP Server mit NodeJS

- Zum Abschluss starten wir noch unseren HTTP-Server mit der **listen** Methode hier können wir noch einen Port angeben, auf welchen der Server hören soll
- Wenn wir jetzt unser Node Programm starten fällt einem sofort auf, dass unser Programm nicht sofort wieder beendet wird sondern der Debugger „offen“ bleibt
- Das liegt daran, dass unser Server jetzt ausgeführt wird und auf Verbindungen hört

Webserver Development

HTTP Server mit NodeJS

The screenshot shows a Node.js development environment. The main area displays the following code:

```
JS playground.mjs > ...
JS playground.mjs > ...
1 import http from 'node:http';
2
3 const server = http.createServer();
4
5 server.on('request', (request, res) => {
6   res.writeHead(200, { 'content-type': 'application/json' });
7   res.end(JSON.stringify({
8     data: 'Hello World!',
9   }));
10 });
11
12 server.listen(8080);
```

The sidebar on the left contains the following sections:

- VARIABLES
- WATCH
- CALL STACK
- LOADED SCRIPTS
- BREAKPOINTS

The bottom bar includes icons for file operations, a search bar, and tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, PORTS, and TERMINAL. The DEBUG CONSOLE tab is active, showing the command: /Users/jens/.nvm/versions/node/v20.11.1/bin/node ./playground.mjs.

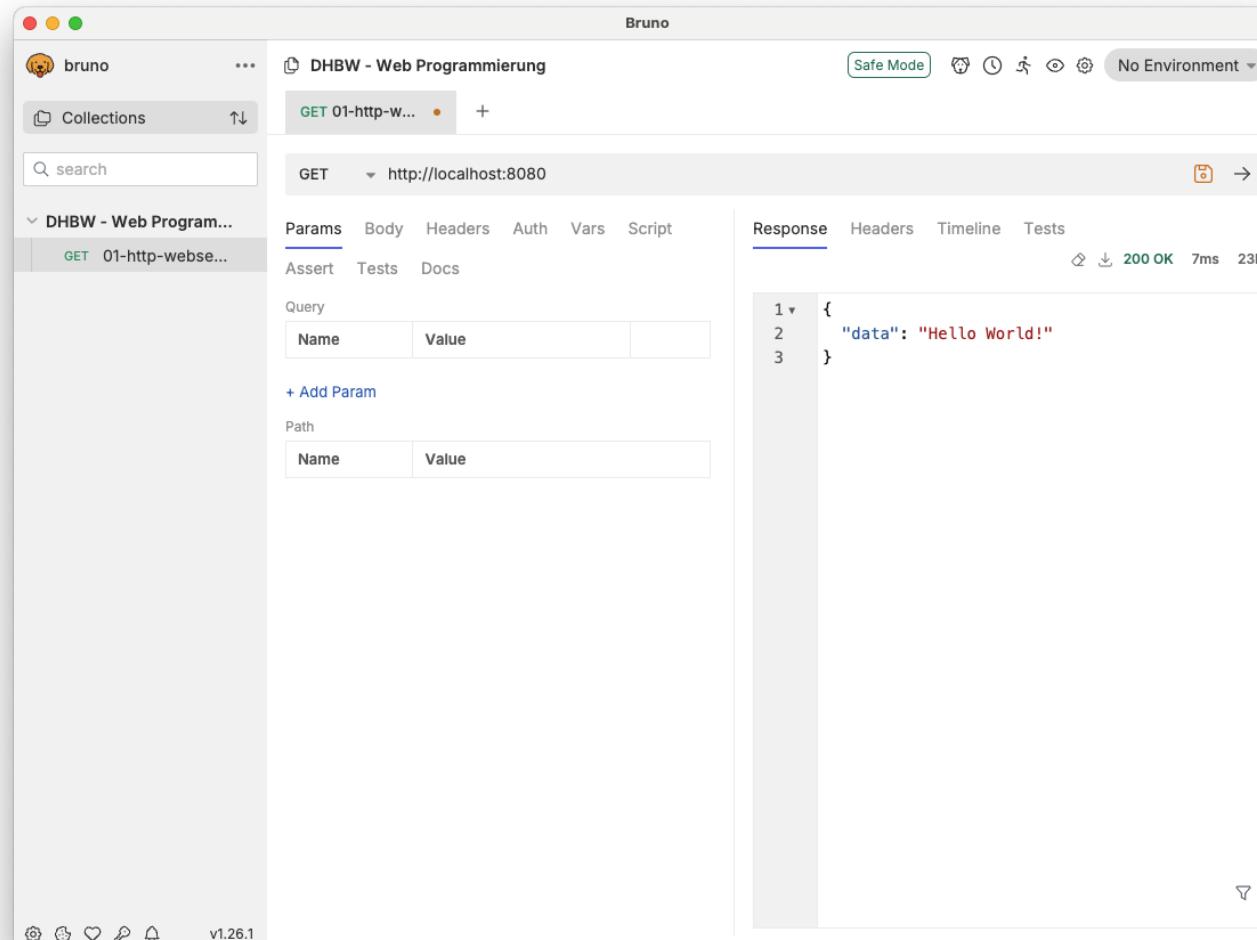
Webserver Development

Kommunikation mit unserem HTTP Server

- Mit einem HTTP-Client können wir jetzt mit unserem Server „sprechen“
- Wir verwenden das Programm Bruno als einen API-Client und erstellen einen neuen Request
- Unser Server läuft unter folgender Adresse: <http://localhost:8080>
- Wenn wir jetzt eine Anfrage abschicken sollten wir von unserem Server eine Antwort bekommen als JSON mit dem Inhalt: „data“: „Hello World!“

Webserver Development

Bruno als API Client



Webserver Development

Bruno als API Client

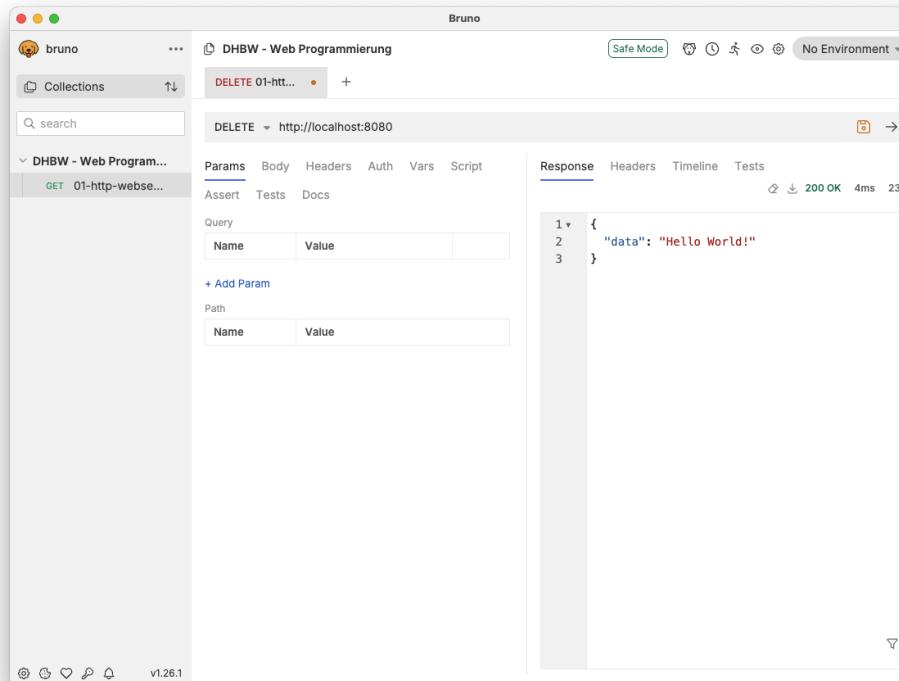
- Im Response Header sehen wir jetzt auch den **content-type** header Wert

Response		Headers	Timeline	Tests
		200 OK	7ms	23B
NAME				VALUE
content-type				application/json
date	Sun, 29 Sep 2024 21:04:45 GMT			
connection	keep-alive			
keep-alive	timeout=5			
transfer-encoding	chunked			

Webserver Development

Bruno als API Client

- Was passiert wenn wir jetzt als Client eine andere HTTP-Methode auswählen?



Webserver Development

HTTP Server mit NodeJS

- Wir bekommen dieselbe Antwort zurück
- Das liegt daran, dass unser aktueller Webserver einfach nur auf Verbindungen „hört“ und nicht zwischen HTTP-Methoden unterscheidet

Webserver Development

HTTP Server mit NodeJS



[Github](#)

- In einer zweiten Version erweitern wir jetzt unseren Webserver, um die Funktionalität auf unterschiedliche Methoden zu reagieren

```
1 res.setHeader("content-type", "application/json");
2 switch (request.method) {
3   case "GET":
4     res.writeHead(200);
5     res.end(JSON.stringify({ message: "GET request received" }));
6     break;
```

Webserver Development

HTTP Server mit NodeJS



[Github](#)

- Für alle Methoden, die wir nicht kennen, wird ein HTTP 405 zurück gegeben

```
1 default:  
2   res.writeHead(405);  
3   res.end(  
4     JSON.stringify({ message: `Method ${request.method} not allowed` })  
5   );  
6 break;
```

Webserver Development

HTTP Server mit NodeJS

- Wenn wir jetzt einen HTTP Request senden und die Methoden wechseln bekommen wir eine passende Antwort zurück

The screenshot shows a web-based testing interface with three main panels:

- Left Panel:** Shows a list of HTTP methods: GET, POST, PUT, DELETE, PATCH, OPTIONS, HEAD. The "POST" method is selected and highlighted with a red border.
- Middle Panel:** Displays a POST request configuration. The URL is set to `http://localhost:8080`. The "Params" tab is active, showing an "Assert" section with the message: `"message": "POST request received"`.
- Right Panel:** Shows the response details. The status is `200 OK`, time is `5ms`, and size is `35B`. The response body is displayed as:

```
1 {  
2   "message": "POST request received"  
3 }
```

Webserver Development

Hintergrund JSON

- Unser Server setzt den Content Type Header auf **application/json**

```
1 res.setHeader("content-type", "application/json");
```

- Wenn wir eine Antwort an den Client schicken nutzen wir die Methode **.stringify()** auf dem Globalen JSON Object und übergeben ein JavaScript Object

```
1 res.end(JSON.stringify({ message: "GET request received" }));
```

- Aber warum eigentlich?

Webserver Development

Hintergrund JSON

- JSON ist ein Text-basiertes Format, welches für den Austausch von Daten verwendet wird

```
1  {
2    "users": [
3      {
4        "name": "Jens Reiner",
5        "age": 28,
6        "roles": ["READ", "UPDATE", "DELETE", "CREATE"]
7      }
8    ],
9    "version": "2.4.5",
10   "softwareName": "DHBW Sample",
11   "active": true
12 }
```

Webserver Development

Hintergrund JSON

- JSON repräsentiert Strukturelle Daten und lehnt sich an die JavaScript Object Syntax an.
- JSON in JavaScript hat ein Globales Namespace Object **JSON**, dass uns Methoden anbietet für das Lesen und Schreiben von JSON
- JSON wurde 2001 von Douglas Crockford veröffentlicht und später als ECMA-404 Standardisiert (2013 & 2017)
 - <https://ecma-international.org/publications-and-standards/standards/ecma-404/>
- JSON ändert sich nicht mehr. Am Standard werden keine Änderungen mehr vorgenommen wodurch man eine extreme Stabilität beim Austausch von Daten erhält

Webserver Development

Hintergrund JSON

- JSON ist Programmiersprachen unabhängig und Sie können in *jeder* Programmiersprache mit JSON arbeiten
 - Bspw. Python <https://docs.python.org/3/library/json.html>
 - Bspw. C# .NET Serialize <https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to>

Webserver Development

Vorteile von JSON

- Aber warum hat sich JSON gegenüber anderen Datenformaten wie bspw. XML durchgesetzt?
- JSON kann komplett als ein String abgebildet werden, wodurch wir Daten sehr einfach über das Netzwerk übertragen können
 - Zeichenketten über eine Netzwerkverbindung schicken ist trivial
- JSON ist Maschinen und Menschen lesbar
 - Anders als bspw. Binär Formate
 - XML Tag Struktur ist aufwendiger zu parsen (CPU Cycles)

Webserver Development

Vorteile von JSON

- JSON ist stark limitiert in der Flexibilität Daten abzubilden
- Dadurch stellen Sie aber auch sicher, dass Sie valides JSON immer identisch interpretieren können
 - Keine Versionen, Keine eigenen Spezifikationen, Keine eigenen Datenformate

Webserver Development

Struktur JSON

- JSON ist nur ein String
 - Beinhaltet nur Properties, kann keine Funktionen abbilden
- JSON ist Format-Sensitive
 - Falsches Komma und das JSON kann nicht mehr gelesen werden

Webserver Development

Struktur JSON

- Aus JavaScript übernimmt JSON folgende Konzepte:
- Object Literals
 - Property Keys sind mit ““ definiert
 - Property Values sind immer JSON Werte
 - Kein Trailing Komma
- Array Literals
 - Elemente sind immer JSON Werte
 - Kein Trailing Komma

Webserver Development

Struktur JSON

- Aus JavaScript übernimmt JSON folgende Konzepte:
- Datentypen:
 - null
 - Booleans
 - Numbers (ohne NaN, +Infinity, -Infinity)
 - Strings (Müssen „ sein und nicht ,)

Webserver Development

JSON-Methoden in JavaScript

- Ein JavaScript Objekt kann mit **JSON.stringify** in einen JSON-String umgewandelt werden
- Ein JSON-String kann mit **JSON.parse()** in ein JavaScript Objekt umgewandelt werden

Webserver Development

JSON-Methoden in JavaScript



[Github](#)

```
1 const user = {  
2   "users": [  
3     {  
4       "name": "Jens Reiner",  
5       "age": 28,  
6       "roles": ["READ", "UPDATE", "DELETE", "CREATE"]  
7     }  
8   ],  
9   "version": "2.4.5",  
10  "softwareName": "DHBW Sample",  
11  "active": true  
12};  
13  
14 const userAsJson = JSON.stringify(user);  
15 console.log('User as json: ', userAsJson);  
16  
17 const userAsObject = JSON.parse(userAsJson);  
18 console.log('User as object: ', userAsObject);
```

Webserver Development

Warum nutzen wir JSON für unsere Schnittstelle?

- Unsere Datenstrukturen, die wir mit JavaScript Objekten darstellen, können wir direkt übersetzen und als JSON im HTTP-Body an einen Client übermitteln
- Daten die uns ein Client schickt (Bspw. Eingaben) können wir direkt in ein JavaScript Objekt übersetzen und damit weiter arbeiten

Webserver Development

HTTP Server mit NodeJS

- Was aber wenn wir als Client ein anderes Datenformat vom Server erhalten möchten?
- Angenommen wir sind ein Webbrowser und möchten die Daten als HTML erhalten

Webserver Development

Content Negotiation

- Ein spezifisches Dokument wird als Ressource bezeichnet
- Ein Client kann eine Ressource über einen Request an eine bestimmte URL anfragen
- Der Server sucht nach der Ressource an der übergebenen URL und gibt eine Variation an den Client zurück – Eine Repräsentation der Ressource
- Mit dem Konzept der Content Negotiation kann ich diese Repräsentation einer Ressource explizit anfragen als Client

Webserver Development

Content Negotiation – Beispiel DHBW Webseite

The screenshot shows the Network tab of a browser developer tools window. A single request to 'startseite' is highlighted. The request URL is 'https://www.heidenheim.dhbw.de/startseite'. The request method is 'GET', and the status code is '200 OK'. The response headers section is expanded, showing the 'Accept' header set to 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7'. The request headers section is also expanded, showing ':authority: www.heidenheim.dhbw.de', ':method: GET', ':path: /startseite', and ':scheme: https'. The Accept header is highlighted with a red box.

Anfrage an die Ressource

Header (Werden vom Browser gesetzt)

Accept Header – Bspw. text/html

Webserver Development

Content Negotiation – Beispiel DHBW Webseite

- Der Server Antwortet mit einer Repräsentation im text/html Format mit Charset UTF-8

▼ Response Headers	
Content-Language:	de
Content-Type:	text/html; charset=utf-8
Date:	Thu, 03 Oct 2024 09:21:13 GMT
Server:	Apache
Strict-Transport-Security:	max-age=15552000;includeSubdomains
X-Content-Type-Options:	nosniff
X-Frame-Options:	SAMEORIGIN
X-Xss-Protection:	1; mode=block

Webserver Development

Content Negotiation – Mechanismus

- Die Repräsentation einer Ressource kann anhand von zwei Mechanismen identifiziert werden:
- Client HTTP Headers
 - Standard
 - server-driven negotiation or proactive negotiation
- Server HTTP Status Codes *300 (Multiple Choices)*, *406 (Not Acceptable)*, *415 (Unsupported Media Type)*
 - agent-driven negotiation or reactive-negotiation

Webserver Development

Content Negotiation – HTTP Server in NodeJS



[Github](#)

- Anbei eine neue Implementierung von unserem HTTP Server, welcher Content Negotiation implementiert

Webserver Development

Content Negotiation – HTTP Server in NodeJS



[Github](#)

```
1 const acceptHeader = request.headers["accept"];
2
3 // Client did not specify an accept header - Fallback to 300 (Agent Driven Negotiation)
4 if (!acceptHeader || acceptHeader.includes("text/*")) {
5   res.writeHead(300);
6   res.end();
7 } else if (acceptHeader) {
8   // Client prefers HTML
9   if (acceptHeader.includes("text/html")) {
10     res.writeHead(200, { "content-type": "text/html" });
11     res.end(`\n      <html>\n        <head><title>Auto</title></head>\n        <body>\n          <h1>Auto Informationen</h1>\n          <p>Name: ${car.name}</p>\n          <p>Hersteller: ${car.hersteller}</p>\n          <p>Farbe: ${car.farbe}</p>\n        </body>\n      </html>\n    `);
12 }
13 // Client prefers JSON
14 if (acceptHeader.includes("application/json")) {
15   res.writeHead(200, { "content-type": "application/json" });
16   res.end(
17     JSON.stringify({
18       name: car.name,
19       hersteller: car.hersteller,
20       farbe: car.farbe,
21     })
22   );
23 }
24 }
```

Webserver Development

Content Negotiation – HTTP Server in NodeJS

- Mit Bruno können wir jetzt einen Accept Header definieren und erhalten die Ressource in Unterschiedlichen Repräsentationen

The image shows two separate cURL command-line interfaces demonstrating content negotiation.

Session 1 (Top): The user is requesting a resource from `http://localhost:8080` using the `GET` method. In the `Headers` tab, the `accept` header is set to `text/html`. The response, visible in the `Response` tab, is a rendered HTML page titled "Auto Informationen" containing the text "Name: EQS", "Hersteller: Mercedes Benz", and "Farbe: Schwarz".

Name	Value
accept	text/html

Session 2 (Bottom): The user is requesting the same resource from `http://localhost:8080` using the `GET` method. In the `Headers` tab, the `accept` header is set to `application/json`. The response, visible in the `Response` tab, is a JSON object with the following structure:

```
1 {  
2   "name": "EQS",  
3   "hersteller": "Mercedes Benz",  
4   "farbe": "Schwarz"  
5 }
```

Webserver Development

Content Negotiation – HTTP Server in NodeJS

- Fällt Ihnen in der vorgestellten Lösung ein Fehler auf?

Webserver Development

Content Negotiation – HTTP Server in NodeJS

- Unser Server kann keine Antwort generieren wenn wir einen accept header mitschicken, welchen wir nicht implementiert haben

```
>     } else if ('.'){
>       // Client prefers HTML
>       if (acceptHeader.includes("text/html")) {
>         res.writeHead(200, { "content-type": "text/html" });
>         res.end(``);
>       }
>       // Client prefers JSON
>       if (acceptHeader.includes("application/json")) {
>         res.writeHead(200, { "content-type": "application/json" });
>         res.end(``);
>       }
>     }
```

Webserver Development

Fazit – Eigener Webserver

- Möchte man einen vollwertigen Webserver mit NodeJS implementieren wird es sehr schnell sehr komplex und umfangreich.
 - Es ist auch nicht empfehlenswert
<https://web.archive.org/web/20190821102906/https://medium.com/intrinsic/why-should-i-use-a-reverse-proxy-if-node-js-is-production-ready-5a079408b2ca>
- Das Node HTTP Module liefert uns zwar eine Schnittstelle für die Netzwerk Kommunikation aber alle typischen Server Interaktionen müssen wir selbst implementieren
- Bspw. alleine für die Content Negotiation gibt es unterschiedliche Implementierungen
 - Siehe Apache Content Negotiation <https://httpd.apache.org/docs/current/en/content-negotiation.html#algorithm>

Webserver Development

Fazit – Eigener Webserver

- In der Vorlesung möchten wir auch keinen vollwertigen Webserver implementieren
 - In der Praxis gibt es dafür auch bereits sehr sehr gute Software Lösungen bspw:
 - Nginx <https://nginx.org/en/>
 - Apache Tomcat <https://tomcat.apache.org>
 - Microsoft IIS Web Server <https://learn.microsoft.com/en-us/iis/get-started/introduction-to-iis/iis-web-server-overview>
- Stattdessen wollen wir eine HTTP-Schnittstelle (API) implementieren, welche Daten als JSON für Clients bereitstellt
 - Als einfacher Webserver kommt aber NodeJS zum Einsatz (Später mehr dazu)

Webserver Development

Weiterführende Links

- MDN Content Negotiation https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation#principles_of_content_negotiation
- MDN JSON https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON
- Creating and Parsing JSON https://exploringjs.com/js/book/ch_json.html#ch_json
- NodeJS HTTP <https://nodejs.org/api/http.html#http>

API

Application Programming Interface

API

Definition

- Eine API (Application Programming Interface) ermöglicht Softwareanwendungen miteinander zu kommunizieren
- Mit einer API kann eine Anwendung A auf Daten bzw. Funktionen von Anwendung B zugreifen und diese nutzen, ohne interne Details (bspw. Implementierungen) zu kennen
- Beispiel: Microsoft PowerPoint möchte diese Präsentation auf meinem Dateisystem speichern

API

Eigenschaften

- Abstraktionen – APIs sind immer eine Abstraktion die Details der Funktionsweise von einem System (Oder anderem Dienst) verbirgt. Dadurch können Interaktionsprozesse leichter abgebildet werden.
 - Als Anwendung muss ich nicht wissen, wie mein Betriebssystem Dateien auf dem System speichert
- Interoperabilität – Viele APIs können Sie mit unterschiedlichen Programmiersprachen oder sogar unterschiedliche Plattformen ansprechen
 - Eine Webseite kann mit JavaScript auf einen Webserver zugreifen, welcher in C# entwickelt wurde

API

Eigenschaften

- Wiederverwendbarkeit – Eine API für einen bestimmten Dienst kann beliebig wiederverwendet werden.
 - Das Speichern von Dateien auf dem System muss nur einmal entwickelt werden und kann dann von beliebigen Anwendungen wiederverwendet werden
- Sicherheit – APIs können Mechanismen implementieren, um den Zugriff auf Ressourcen und Diensten zu steuern

API

Arten von APIs

- APIs können für unterschiedliche Systeme entwickelt werden bspw. Web-API, Betriebssystem-API, Hardware-API
- In dieser Vorlesung konzentrieren wir uns auf das Entwickeln von Web-APIs (Server) die Daten für unterschiedliche Dienste (Clients) in einem Verteilten System (WWW) bereitstellt.
- Als Client möchten wir Daten von einem Server haben

API

Bier App – Neue Anforderungen

Für die Bier App soll eine Schnittstelle bereitgestellt werden die es anderen Software Herstellern und Usern erlaubt auf die Daten zuzugreifen. Die Schnittstelle soll über das Internet erreichbar sein.

Im dritten Meilenstein soll unsere Bier App erweitert werden und alle Funktionalitäten über eine Server Schnittstelle angeboten werden. Andere Clients oder Server können über diese Schnittstelle die zuvor implementierten Funktionen verwenden. Wir möchten die Bier Funktionen in anderen Programmen nutzen.

API

Beer API – Eine API für Biere

- Wir werden jetzt Schritt für Schritt die Funktionen aus der ursprünglichen Beer App als API implementieren
- Am Ende können wir über HTTP Anfragen an diese API stellen und Informationen abrufen
 - Bspw. als Client möchte ich alle Biere aufgelistet haben

Hinweis

Begriffs Definition

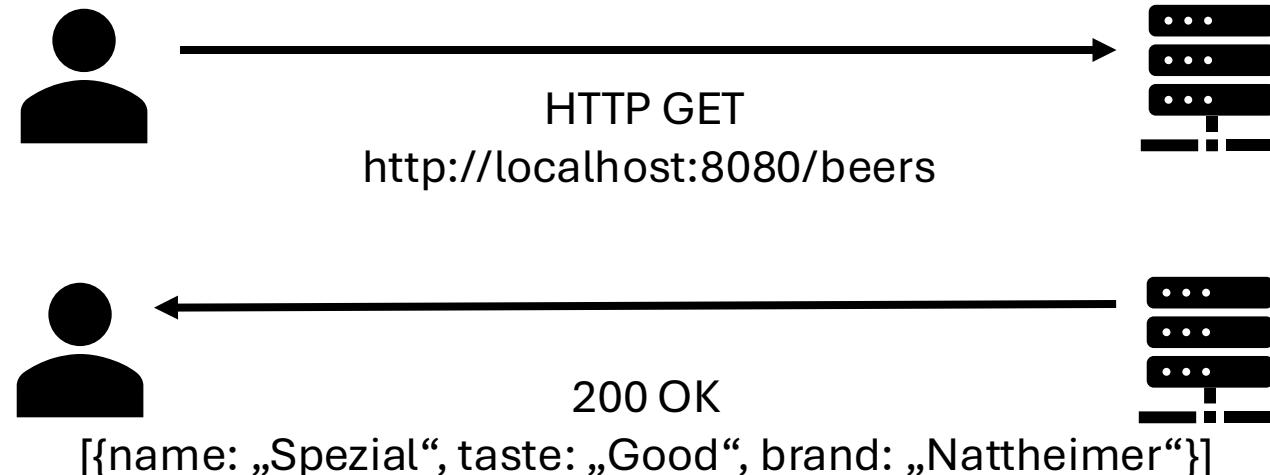
- Wenn wir Client Server Anwendungen implementieren spricht man häufig von Frontend und Backend
- Das Backend bezeichnet hier üblicherweise eine Server Schnittstelle die als API angeboten wird
- Das Frontend ist eine Client Applikation, die eine Benutzeroberfläche bereitstellt und mit dem Backend kommuniziert

Backend Development

Ein erster Entwurf für die Beer API

Backend Development

Beer API – Ein erster Software Entwurf



Backend Development

Beer API – Ein erster Software Entwurf

- Unsere Ressource ist ein Bier
- Die Repräsentation von Bier ist immer ein JSON
- Die Ressource Bier kann angelegt, bearbeitet, verändert oder gelöscht werden
 - CREATE, READ, UPDATE, DELETE (CRUD)
- Die Ressource Bier kann über eine eindeutige URL identifiziert werden

Backend Development

Beer API – Ein erster Software Entwurf

- Wie würden Sie die URLs definieren, um die gewünschte Beer API darzustellen?
- Wie würden Sie die Operationen Create, Read, Update und Delete abbilden?
- Wie kann eine Mögliche Implementierung in NodeJS aussehen, wenn Sie das HTTP-Module benutzen?

Backend Development

Beer API – Ein erster Software Entwurf



[Github](#)

```
1 server.on("request", (request, res) => {
2   res.setHeader('content-type', 'application/json');
3
4   if (request.url === '/beers') {
5     res.statusCode = 200;
6     const nattheimerSpezial = createBeer({
7       name: "Spezial",
8       taste: "Good",
9       brand: "Nattheimer",
10      });
11     res.end(JSON.stringify(nattheimerSpezial))
12   }
13 });
```

DHBW - Web Programmierung

GET 00-api-dr... × +

GET http://localhost:8080/beers

Params Body Headers Auth Vars

Script Assert Tests Docs

Query

Name	Value

+ Add Param

Path

Name	Value

Response Headers Timeline Tests

200 OK 13ms 76B

```
1 {
2   "name": "Spezial",
3   "taste": "Good",
4   "brand": "Nattheimer",
5   "amount": 0,
6   "rating": 1
7 }
```

Backend Development

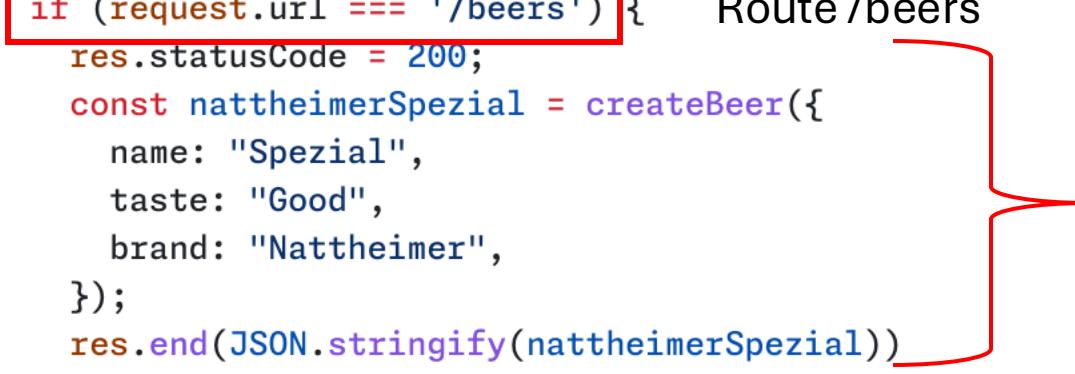
Grundlagen: Routen

- Die eindeutige URL, um eine Ressource zu identifizieren, wird in der Backend Entwicklung als Route bezeichnet
- Eine Route ist dabei ein Endpunkt, welcher von einem Client aufgerufen werden kann und dann die HTTP-Anfrage verarbeitet

Backend Development

Grundlagen: Routen

```
1 server.on("request", (request, res) => {
2   res.setHeader('content-type', 'application/json');
3
4   if (request.url === '/beers') {    Route /beers
5     res.statusCode = 200;
6     const nattheimerSpezial = createBeer({
7       name: "Spezial",
8       taste: "Good",
9       brand: "Nattheimer",
10      });
11     res.end(JSON.stringify(nattheimerSpezial))
12   }
13});
```



Verarbeitung der Anfrage

Backend Development

Beer API – Bier anlegen Funktionalität

- Im vorherigen Beispiel haben wir die Route /beers implementiert, welche ein Bier erstellt und uns dieses als JSON zurück gibt
- Wie unterscheiden wir jetzt aber ob ich ein Bier anlegen möchte oder ein bestimmtes Bier lesen möchte?

Backend Development

Beer API – Bier anlegen Funktionalität

- Zwei Möglichkeiten:
- Wir erstellen zwei Unterschiedliche Routen /beers und /createBeer
- Wir arbeiten mit HTTP-Methoden auf derselben Route
 - POST /beers -> Anlegen
 - GET /beers -> Bier lesen

Backend Development

Beer API – Bier anlegen Funktionalität



[Github](#)

```
1 server.on("request", (request, res) => {
2   res.setHeader('content-type', 'application/json');
3
4   if (request.url === '/beers') {
5     if (request.method === 'GET') {
6       res.statusCode = 200;
7       res.end(JSON.stringify(beers));
8     }
9     if (request.method === 'POST') {
10       res.statusCode = 201;
11       const nattheimerSpezial = createBeer({
12         name: "Spezial",
13         taste: "Good",
14         brand: "Nattheimer",
15       });
16       res.end(JSON.stringify(nattheimerSpezial))
17     }
18   }
19 });
```

Backend Development

Beer API – Bier anlegen Funktionalität

Alle Biere Abfragen mit GET

GET ▾ http://localhost:8080/beers →

Params	Body	Headers	Auth	Vars
Script	Assert	Tests	Docs	

Query

Name	Value

+ Add Param

Path

Name	Value

Response Headers Timeline Tests

200 OK 4ms 78B

```
1 [  
2 {  
3   "name": "Spezial",  
4   "taste": "Good",  
5   "brand": "Nattheimer",  
6   "amount": 0,  
7   "rating": 1  
8 }  
9 ]
```

Ein Bier anlegen mit POST

POST ▾ http://localhost:8080/beers →

Params	Body	Headers	Auth	Vars
Script	Assert	Tests	Docs	

Query

Name	Value

+ Add Param

Path

Name	Value

Response Headers Timeline Tests

201 Created 4ms 76B

```
1 {  
2   "name": "Spezial",  
3   "taste": "Good",  
4   "brand": "Nattheimer",  
5   "amount": 0,  
6   "rating": 1  
7 }
```

Backend Development

Beer API – Fazit

- Die Ressource Bier wird mit einer eindeutigen Route genau zugeordnet
- Welche Operation wir ausführen möchten können wir über die HTTP-Methoden angeben
- Aber wir müssen aktuell jeden dieser Schritte selbst implementieren und haben einen großen Event Handler (request) der unsere Schnittstelle darstellt

Backend Development

Introduction to Fastify

- Anstatt unsere Beer API mit dem Low Level HTTP Module von NodeJS zu implementieren, möchten wir stattdessen auf ein fertiges Web Framework zurückgreifen, welches viele der benötigten Funktionalitäten bereits fertig anbietet
- In der Vorlesung verwenden wir dafür Fastify <https://fastify.dev>

Backend Development

Fastify

- Fastify ist ein Web Framework für NodeJS
- Es benötigt wenig Overhead für die Umsetzung von Schnittstellen und legt dabei Wert auf Performance und Developer Experience
 - 30000 Anfragen pro Sekunde
- Fastify unterstützt ein Plugin System, wodurch wir weitere Funktionalitäten einfach on-top entwickeln können
- Fastify unterstützt JSON Schema out-of-the-box
 - <https://json-schema.org>

Backend Development

Installation von Fastify

- Fastify ist eine Third Party Dependency und wird von der Community entwickelt (Nicht von NodeJS)
- Um Fastify in NodeJS verwenden zu können müssen wir also zuerst diese Dependency installieren

NPM

Der Package Manager für NodeJS

NPM

Hintergrund

- In den vorherigen Kapiteln haben wir uns mit JavaScript Modulen beschäftigt
- Module können dabei Funktionalitäten exportieren und wir können diese importieren
- Ein Beispiel für ein Modul war das HTTP-Modul, welches direkt mit NodeJS installiert wird
- Wir können aber auch Externe Module installieren und verwenden

NPM

Packages

- Wenn wir eine Funktionalität als Module oder eine Kombination aus Modulen implementiert haben können wir diese als Package für andere bereitstellen
- Diese Packages können dann wiederum in anderen NodeJS Anwendungen installiert werden
- Bspw. unsere Beer App könnten wir als Package bereitstellen

NPM

Packages

- Packages können wir dabei aus verschiedenen Quellen beziehen
- Packages haben auch eigene Versionen und wir können bspw. eine spezielle Version für unser Projekt installieren
- Um Packages zu verwalten, benötigen wir einen Package Manager
- NodeJS installiert automatisch NPM mit (Node Package Manager)
 - Es gibt aber auch noch andere Implementierungen bspw. Yarn

NPM

Grundlagen

- Um in einem NodeJS Projekt externe Packages zu verwenden, gibt es eine spezielle Datei **package.json**
- Jedes NodeJS Projekt ist gleichzeitig auch wieder ein Package und wird über die package.json beschrieben
- Das Anlegen von einer solchen Datei können wir mit “npm init“ machen

NPM

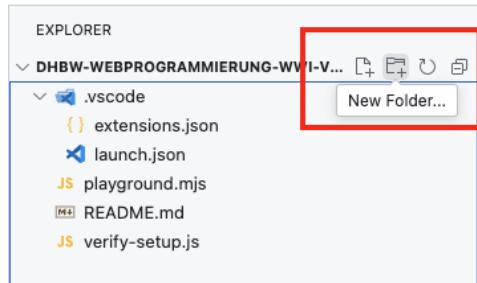
Grundlagen – Package JSON für ein Projekt anlegen

- Ihr NodeJS Projekt sollte immer in einem Ordner liegen
- Auf der obersten Ebene in diesem Ordner liegt dann auch die package.json Datei
- In Visuell Studio Code können Sie folgendermaßen eine package.json in einem Ordner anlegen

NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Öffnen Sie den Explorer in VSCode und klicken Sie auf „New Folder“



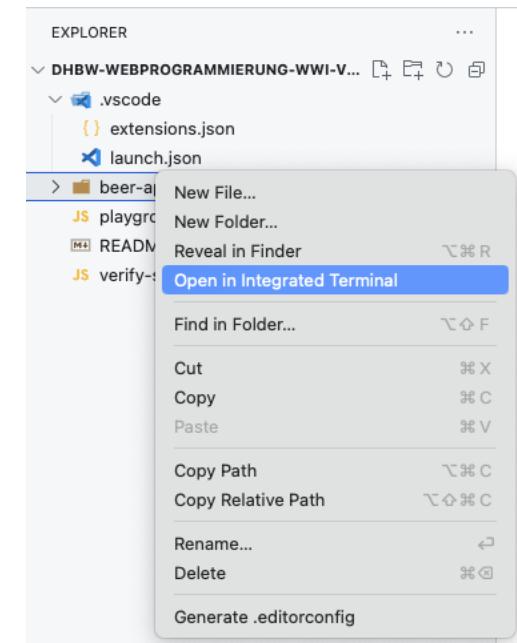
- Geben Sie dem Ordner einen passenden Namen und bestätigen Sie mit Enter



NPM

Grundlagen – Package JSON für ein Projekt anlegen

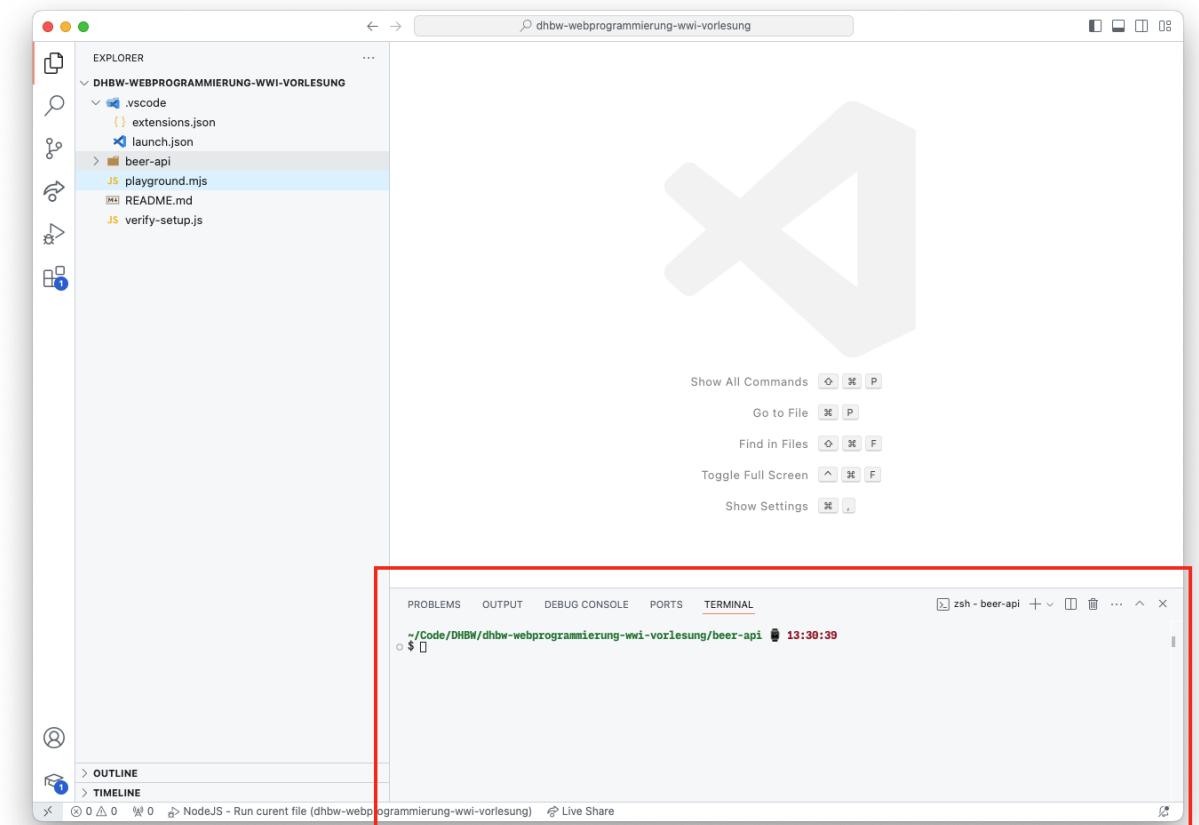
- Machen Sie einen Rechtsklick auf den Ordner und wählen Sie „Open in Integrated Terminal“ aus
 - Text kann je nach Betriebssystem und Sprache anders aussehen



NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Im unteren Bereich von VSCode sollte sich ein neues Fenster öffnen



NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Tippen Sie hier jetzt **npm init** ein und bestätigen mit enter
- Es sollte sich ein Utility Programm öffnen, welche Sie nach bestimmten Informationen fragt
- Für den Moment können Sie einfach mit enter jeden Schritt überspringen

```
~/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/beer-api 13:30:39
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (beer-api)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/beer-api/package.json:

{
  "name": "beer-api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

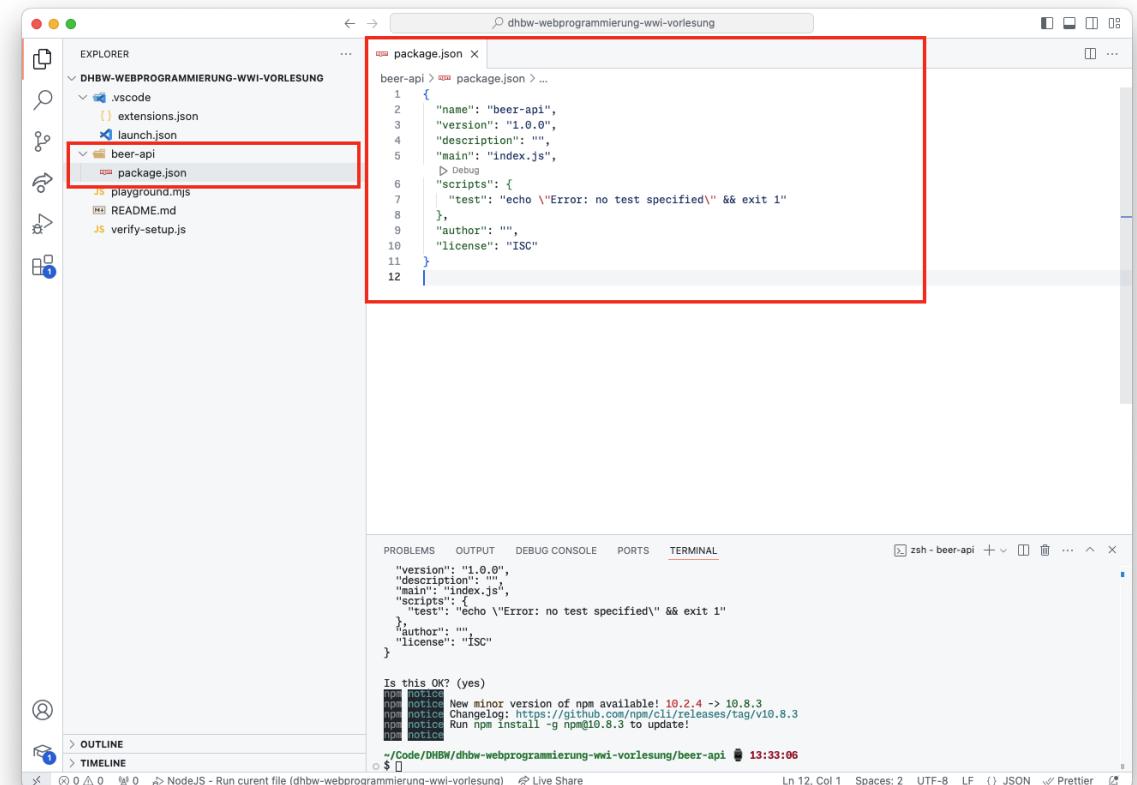
Is this OK? (yes)
npm notice New minor version of npm available! 10.2.4 -> 10.8.3
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.3
npm notice Run npm install -g npm@10.8.3 to update!
npm notice

~/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/beer-api 13:32:49
$
```

NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Anschließend sollten Sie in Ihrem Ordner eine package.json Datei finden mit ähnlichem Inhalt



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

- File Explorer:** Shows a folder named "DHBW-WEBPROGRAMMIERUNG-WWI-VORLESUNG" containing "vscode", "extensions.json", "launch.json", "beer-api", and "package.json". The "package.json" file is selected and highlighted with a red box.
- Code Editor:** Displays the content of the "package.json" file. The code is as follows:

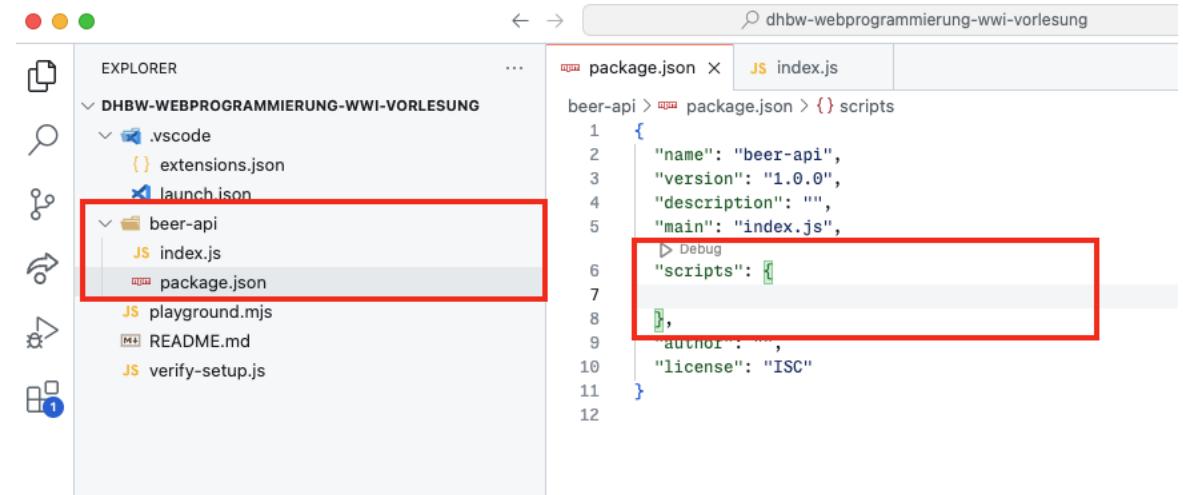
```
beer-api > package.json
{
  "name": "beer-api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

The code editor has syntax highlighting for JSON. A red box highlights the "package.json" file in the Explorer and the entire JSON object in the editor.

NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Erstellen Sie sich jetzt im selben Ordner eine index.js Datei
- Öffnen Sie die package.json Datei und entfernen im Bereich „scripts“ den Eintrag „test“



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

- File Explorer:** Shows the project structure under "DHBW-WEBPROGRAMMIERUNG-WWI-VORLESUNG". It includes a ".vscode" folder containing "extensions.json" and "launch.json", and a "beer-api" folder containing "index.js", "package.json", "playground.mjs", "README.md", and "verify-setup.js".
- Code Editor:** The "package.json" file is open in the editor. The code is as follows:

```
1 {  
2   "name": "beer-api",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "index.js",  
6   "scripts": {  
7     "Debug": "node index.js",  
8     "test": "echo \"Error: no test specified\" & exit 1"  
9   },  
10  "author": "",  
11  "license": "ISC"  
12}
```

Two specific sections of the code are highlighted with red boxes: the "beer-api" folder in the file tree and the "scripts" block in the package.json code.

NPM

Grundlagen – Package JSON für ein Projekt anlegen

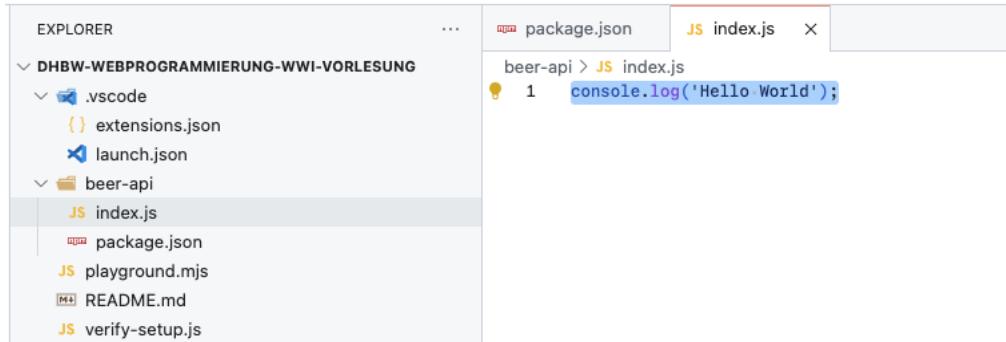
- Fügen Sie in der scripts Property einen neue property **start** hinzu mit dem Wert **node index.js**
- Fügen Sie nach dem license key eine neue Property **type** hinzu mit dem Wert **module**
- Speichern Sie die Datei

```
1  {
2    "name": "beer-api",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start": "node index.js"
8    },
9    "author": "",
10   "license": "ISC",
11   "type": "module"
12 }
```

NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Öffnen Sie die index.js Datei und fügen ein console.log Statement hinzu



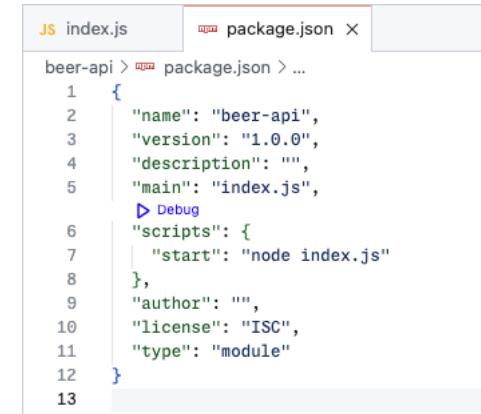
The screenshot shows the VS Code interface. On the left is the Explorer sidebar with a tree view of files and folders. In the center is the Editor tab bar with 'package.json' and 'index.js' selected. The main editor area shows the code for 'index.js':

```
beer-api > JS index.js
1   console.log('Hello World');
```

NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Zum ausführen können Sie jetzt direkt in VSCode auf Debug klicken welches über Scripts erscheint



The screenshot shows the VSCode interface with two tabs open: 'index.js' and 'package.json'. The 'package.json' tab is active, displaying the following JSON code:

```
beer-api > package.json > ...
1  {
2    "name": "beer-api",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start": "node index.js"
8    },
9    "author": "",
10   "license": "ISC",
11   "type": "module"
12 }
13
```

A tooltip labeled 'Debug' is shown over the 'scripts' object in the JSON code.

NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Oder Sie starten das NPM Script über das Terminal mit **npm run start**



The screenshot shows a terminal window with the following content:

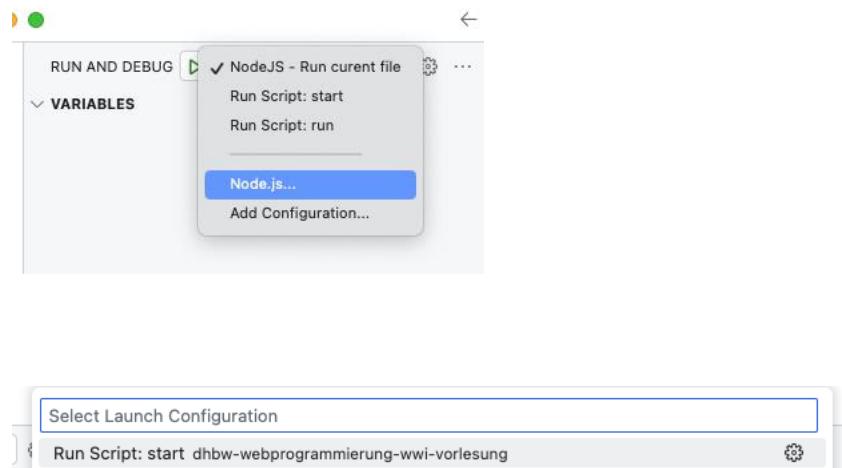
```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
~/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/beer-api 13:50:29
$ npm run start
> beer-api@1.0.0 start
> node index.js
Hello World
~/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/beer-api 13:50:34
$ █
```

The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, PORTS, and TERMINAL. The TERMINAL tab is active. The command \$ npm run start is entered, followed by two line continuations (>) indicating the command is being processed. The output shows the application starting and printing "Hello World". The terminal prompt is then shown again.

NPM

Grundlagen – Package JSON für ein Projekt anlegen

- Oder Sie benutzten das Bekannte Debugger Menu – Müssen aber eine neue Config dafür hinzufügen



NPM

Grundlagen – Package installieren

- Öffnen Sie wieder ein Terminal und führen Sie folgenden Befehl aus **npm install fastify**
- Es sollte sich anschließend die package.json automatisch update und in Ihrem Ordner sollte ein neuer node_modules Ordner vorhanden sein

NPM

Grundlagen – Package Installieren

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar displays a project structure under 'DHBW-WEBPROGRAMMIERUNG-WWI-VORLESUNG'. A red box highlights the 'beer-api' folder, which contains 'node_modules', 'index.js', 'package-lock.json', and 'package.json'. Another red box highlights the 'dependencies' section of the 'package.json' file, which includes the line: `"fastify": "^5.0.0"`. The right side of the screen shows the code editor with the same 'package.json' file open. Below the editor is the 'TERMINAL' panel, which shows the command line history:

```
~/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/beer-api 13:50:29
$ npm run start
> beer-api@1.0.0 start
> node index.js
Hello World
~/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/beer-api 13:52:11
$ npm install fastify
added 56 packages, and audited 57 packages in 4s
6 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
~/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/beer-api 13:54:20
$
```

NPM

Grundlagen – Package.json

- Die package.json ist die wichtigste Datei in einem NodeJS Projekt und beschreibt u.a. was für ein Package Sie implementieren und welche Abhängigkeiten ihr Package hat
- In der package.json können auch Scripte hinterlegt werden die bspw. für die Ausführung notwendig sind oder zum Testen genutzt werden
- Sie definieren auch das JavaScript Module Format in dieser Datei mit der Property **type**

NPM

Grundlagen – Package.json

- Wenn Sie ein neues Package installieren fügt NPM dieses Automatisch in der Property dependencies hinzu
 - Sie sehen den Package Namen und in welcher Version das Package installiert wurde
- Wenn Sie ein bestehendes NodeJS Projekt öffnen müssen Sie zuerst alle Packages installieren
 - Hierfür können Sie den Befehl **npm install** nutzen. Anschließend sollten Sie einen node_modules Ordner haben

NPM

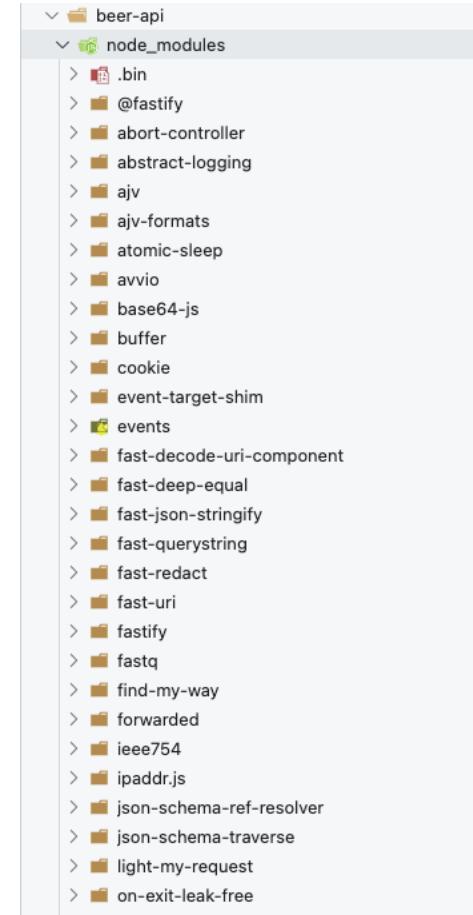
Grundlagen – Package.json

- Packages werden immer nur lokal installiert d.h. wenn Sie ein Projekt über Git teilen möchten stellen Sie sicher, dass node_modules in der .gitignore exkludiert wird
 - Bspw. Laden Sie sich einfach die node gitignore von Github
<https://github.com/github/gitignore/blob/main/Node.gitignore>

NPM

Grundlagen – Node Modules Ordner

- Mit der Installation von Fastify hat NPM auch gleich weitere Abhängigkeiten installiert
- Fastify selbst hat auch wieder Package Dependencies um diese müssen Sie sich aber nicht kümmern sondern NPM installiert diese automatisch mit und legt alle in den node_modules Ordner ab



NPM

Grundlagen – Dependency nutzen

- Wenn Sie jetzt Fastify nutzen möchten können Sie sich einfach die Funktionalitäten importieren (vgl. mit Node HTTP module)

```
1 import Fastify from 'fastify'  
2 const fastify = Fastify({  
3   logger: true  
4 })  
5  
6 // Declare a route  
7 fastify.get('/', async function handler (request, reply) {  
8   return { hello: 'world' }  
9 })  
10  
11 // Run the server!  
12 try {  
13   await fastify.listen({ port: 3000 })  
14 } catch (err) {  
15   fastify.log.error(err)  
16   process.exit(1)  
17 }
```

NPM

Grundlagen – Dependency nutzen

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

- Explorer View:** Shows a project structure for "DHBW-WEBPROGRAMMIERUNG-WWI-VORLESUNG". Inside the "beer-api" folder, there are files: .vscode, index.js, package-lock.json, package.json, .gitignore, playground.mjs, README.md, and verify-setup.js.
- Code Editor:** The "index.js" file is open, displaying the following code:

```
import Fastify from 'fastify'
const fastify = Fastify({
  logger: true
})
// Declare a route
fastify.get('/', async function handler (request, reply) {
  return { hello: 'world' }
})
// Run the server!
try {
  await fastify.listen({ port: 3000 })
} catch (err) {
  fastify.log.error(err)
  process.exit(1)
}
```

- Terminal View:** The "TERMINAL" tab shows the command "npm run start" being run, which starts the application. The output includes the command, the fact that a debugger is attached, and logs indicating the server is listening at `http://127.0.0.1:3000`.
- Status Bar:** The status bar at the bottom shows the file is a JavaScript file (Ln 4, Col 3), has 2 spaces, uses LF line endings, and is a Prettier-formatted file.

NPM

Übung

- Erstellen Sie sich einen neuen Ordner „dates-are-hard“
- Nutzen Sie npm init um ein neues Node Projekt aufzusetzen
- Erstellen Sie eine index.js Datei sowie ein Start Script um das Projekt auszuführen
 - Vergessen Sie nicht den type auf module zu stellen in der package.json
- Installieren Sie sich die date-fns Dependency
 - <https://www.npmjs.com/package/date-fns>
- Übernehmen Sie das Beispiel aus der date-fns Dokumentation und geben Sie ein formatiertes Datum in der Console aus

NPM

Hintergründe

- In der modernen Software Entwicklung ist es üblich mit externen Packages zu arbeiten
- Dabei können externe Packages bspw. intern von einer anderen Abteilung kommen oder Sie nutzen Open Source Software
- In JavaScript ist der bekannteste Package Manager NPM
 - In anderen Programmiersprachen ist es bspw. Nuget (C#), Cargo (Rust), Pip (Python)
- NPM ist auch Zeitgleich die Package Registry
 - In anderen Programmiersprachen bspw. Crates.io (Rust), PyPi (Python)

NPM

Hintergründe

- Viele Software Anforderungen oder Problemstellungen sind bereits gelöst und sie können bspw. Open Source Libraries dafür nutzen
 - Beachten Sie bitte dabei immer den FOSS Compliance Prozess
- Sie finden aber auch bspw. komplette Frameworks, die für einen bestimmten Use Case eine Lösung bereitstellen
 - Bspw. Angular für die Frontend Entwicklung
- Aber Sie finden auch sehr viele Packages die mittlerweile Standard sind oder wirklich nicht gebraucht werden
 - <https://github.com/jezen/is-thirteen>

NPM

Hintergründe

- Third Party Packages sind ein Großartiges Werkzeug um die Software Entwicklung deutlich zu beschleunigen und zu vereinfachen
 - Sie müssen bspw. das Rad nicht jedes Mal neu erfinden
- Viele Open Source Lösungen bieten unglaublich tolle Lösungen an, auf die Sie einfach zurückgreifen können
- Aber in diese Package Registries kann jeder Dinge hochladen und vertreiben
 - Auch Schadsoftware kann über diesen Mechanismus bereitgestellt werden

NPM

Rule of Thumb

- **Installieren Sie nicht blind Pakete aus der NPM Registry**
- Achten Sie immer auf die Metriken Weekly Downloads und Last Publish wenn Sie sich überlegen ein Package zu nutzen
- Öffnen Sie auch das verlinkte Github und schauen Sie wie viele Stars das jeweilige Projekt hat und wie aktiv es ist
 - Letzter Commit 4 Jahre her?
 - Prüfen Sie ob eine Security Notice vorhanden ist d.h. wie wird mit Lücken umgegangen
- Prüfen Sie die License die das Package hat (MIT oder Harte FOSS License?)

NPM

Rule of Thumb

- Die meisten Funktionalitäten kann NodeJS (bzw. JavaScript) mittlerweile out-of-the-box und Bedarf kein externes Package mehr

NPM

Cheat Sheet

- In der package.json Datei werden Dependencies und Scripte für das jeweilige Projekt definiert
- Mit npm install <package-name> kann ein externes Package installiert werden
- Mit “type”: „module“ führt NodeJS das Projekt mit ESModules aus
- Nicht blind jedes Package installieren, was man im Internet findet
- Packages sind immer nur lokal installiert d.h. ich muss npm install ausführen

NPM

Hinweise

- Die nachfolgenden Lösungen verwenden externe Packages d.h. wenn Sie die Beispiele bei sich selbst ausführen möchten müssen Sie einmal npm install ausführen

Fastify

Backend Development mit Fastify

Fastify

Hinweise

- Die Source Code Beispiele von Fastify werden alle im selben Node Projekt (Eine package.json) implementiert
- Sie können die unterschiedlichen Beispiele über die package.json scripte starten
- Wenn Sie die Beispiele bei sich ausprobieren wollen stellen Sie sicher, dass Sie alle Abhängigkeiten mit npm install einmal lokal installiert haben

```
1 "scripts": {  
2   "start:01": "node 01-server.js",  
3   "start:02": "node ./02-plugins/server.js",  
4   "start:03": "node ./03-body/server.js",  
5   "start:04": "node ./04-json-schema/server.js"  
6 },
```

Fastify

Hinweise

- Falls Sie einen solchen Fehler beim Ausführen bekommen deutet es darauf hin, dass Sie eine Dependency noch nicht installiert haben
- Wir benutzen wieder den Port 8080 für unseren Server
 - In der Dokumentation von Fastify wird 3000 als Default Port verwendet

```
throw new ERR_MODULE_NOT_FOUND(packageName, fileURLToPath(base), null);

Error [ERR_MODULE_NOT_FOUND]: Cannot find package 'fastify' imported from /Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/01-fastify/01-server.js
    at packageResolve (node:internal/modules/esm/resolve:853:9)
    at moduleResolve (node:internal/modules/esm/resolve:910:20)
    at defaultResolve (node:internal/modules/esm/resolve:1130:11)
    at ModuleLoader.defaultResolve (node:internal/modules/esm/loader:396:12)
    at ModuleLoader.resolve (node:internal/modules/esm/loader:365:25)
    at ModuleLoader.getModuleJob (node:internal/modules/esm/loader:240:38)
    at ModuleWrap.<anonymous> (node:internal/modules/esm/module_job:85:39)
    at link (node:internal/modules/esm/module_job:84:36) {
      code: 'ERR_MODULE_NOT_FOUND'
}

Node.js v20.11.1
Waiting for the debugger to disconnect...
```

Fastify

Hinweise

- Das Keyword **async** schauen wir uns erst später an und können wir für den Moment ignorieren

Fastify

Installation

- Erstellen Sie sich ein neues Node Projekt mit dem Befehl **npm init**
 - Wechseln Sie ihr Node Projekt auf ESMODules (“type”: „module“ in der package.json)
- Installieren Sie Fastify mit **npm install fastify**
- Legen Sie eine neue JavaScript Datei an (server.js) und fügen Sie ein Script Befehl zur package.json hinzu: „**start**“: „**node server.js**“

Fastify

Installation



[Github](#)

- Importieren Sie folgenden Quellcode
- Starten Sie ihren Server mit **npm run start**

```
1 import Fastify from 'fastify'
2 const fastify = Fastify({
3   logger: true
4 })
5
6 // Declare a route
7 fastify.get('/', async function handler (request, reply) {
8   return { hello: 'world' }
9 })
10
11 // Run the server!
12 try {
13   await fastify.listen({ port: 8080 })
14 } catch (err) {
15   fastify.log.error(err)
16   process.exit(1)
17 }
```

Fastify

Installation

- Eine Ausführliche Erläuterung finden Sie auch hier
<https://github.com/wasdJens/dont-panic-web-edition/blob/main/how-to-guides/fastify/01-first-steps.md>

Fastify

Grundlagen

- Über das Fastify Objekt können wir uns eine neue Server Instanz erzeugen
- Wir können Fastify noch zusätzliche Optionen als Parameter mitgeben
 - In unserem Beispiel schalten wir die Logging Funktionalität an
- Zum Starten gibt es die **listen** Methode die als Parameter einen Port entgegen nimmt

```
1 import Fastify from "fastify";
2 const fastify = Fastify({
3   logger: true,
4 });
```

```
1 try {
2   await fastify.listen({ port: 8080 });
3 } catch (err) {
4   fastify.log.error(err);
5   process.exit(1);
6 }
```

Fastify

Grundlagen

- Wichtig sobald die **listen** Methode ausgeführt wird können Sie keine weiteren Routen, Funktionalitäten etc. zu Fastify hinzufügen weil die Server Instanz dann bereits am Laufen ist
 - Fastify wird beim Starten auch einen Fehler werfen

```
1|:50050"}  
{"level":30,"time":1728028872926,"pid":82400,"hostname":"Jenss-MBP","msg":"Server listening at http://127  
.0.0.1:8080"}  
  
/Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/01-fastify/node_modules/fastify/fastify.js:536  
if (fastify[kState].started) throw new FST_ERR_INSTANCE_ALREADY_LISTENING(msg)  
^  
Waiting for the debugger to disconnect...  
FastifyError [Error]: Fastify instance is already listening. Cannot add route!  
    at throwIfAlreadyStarted (/Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/01-fastify/node_  
modules/fastify/fastify.js:536:49)  
    at Object.route (/Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/01-fastify/node_modules/f  
astify/lib/route.js:199:5)  
    at Object.prepareRoute (/Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/01-fastify/node_mo  
dules/fastify/lib/route.js:147:18)  
    at Object._get [as get] (/Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/01-fastify/node_m  
odules/fastify/fastify.js:259:34)  
    at file:///Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi-vorlesung/01-fastify/01-server.js:24:9  
    at processTicksAndRejections (node:internal/process/task_queues:95:5) {  
      code: 'FST_ERR_INSTANCE_ALREADY_LISTENING',  
      statusCode: 500  
    }  
  
Node.js v20.11.1  
Waiting for the debugger to disconnect...
```

Fastify

Grundlagen – Endpunkte

- Um jetzt einen Endpunkt bereitzustellen können wir eine Route definieren
- Ähnlich wie in unserem HTTP Server für die Beer API möchten wir eine Route definieren, die eine HTTP Anfrage entgegen nimmt und eine Antwort an den Client zurück gibt

```
1 fastify.get("/", async function handler(request, reply) {  
2   return { hello: "world" };  
3 });  
4
```

Fastify

Grundlagen – Routen

- Fastify bietet für das Bereitstellen von Routen fertige Methoden auf dem Fastify Objekt an

```
1 fastify.get("/", async function handler(request, reply) {});
```

- Es fällt auch auf, dass wir direkt ein JavaScript Objekt zurückgeben können und nicht bestimmte HTTP Header setzen müssen oder ähnliches

```
1 return { hello: "world" };
```

Fastify

Grundlagen – Routen

- Fastify übernimmt für uns den Overhead die richtigen HTTP Header oder Status Codes zu setzen und ermittelt selbstständig bspw. den passenden Content Type oder HTTP Status Code
- Mit unserem Fastify Server können wir auch jeden anderen HTTP Methode direkt als Route abbilden
- Routen sind dabei nach folgendem Schema aufgebaut:
 - *fastify.<http-method>(<path>, [options], handler)*

Fastify

Grundlagen – Routen

- Für ein POST verwenden wir also dasselbe Konstrukt und ändern die Methode auf **post**

```
1 fastify.post("/", async function handler(request, reply) {  
2   return { hello: 'POST'};  
3 })
```

Fastify

Grundlagen – Routen

- Der Erste Parameter ist die eindeutige URL, über die unsere Route angesprochen werden kann
- Der Zweite Parameter ist eine Callback Funktion die einen Handler definiert, der ausgeführt wird wenn ein Client diese Route anspricht

Fastify

Grundlagen – Routen

- Im Handler bekommen wir zwei Objekte **request** und **reply**
 - Request beinhaltet die Client Anfrage Informationen
 - Siehe <https://fastify.dev/docs/latest/Reference/Request/>
 - Reply ist ein spezielles Fastify Objekt, um die Antwort zu modifizieren
 - Siehe <https://fastify.dev/docs/latest/Reference/Reply/>

```
1 fastify.get("/test", async function handler(request, reply) {
2   reply.code(500);
3   return { error: 'Something went wrong'};
4 });
5
```

The screenshot shows a browser-based interface for testing Fastify routes. At the top, there's a header with 'GET' and a red box around the URL 'http://localhost:8080/test'. Below the header are tabs for 'Params', 'Body', 'Headers', 'Auth', and 'Vars', with 'Params' being the active tab. There are also tabs for 'Script', 'Assert', 'Tests', and 'Docs'. A 'Query' section with a table for 'Name' and 'Value' is present. On the right side, there's a 'Response' panel. It shows a red box around the status code '500 Internal Server Error'. Below the status code, it says '16ms 32B'. The response body is displayed as a JSON object: { "error": "Something went wrong" }.

Fastify

Grundlagen – Plugins

- Ein gängiges Pattern in der Entwicklung von Schnittstellen ist es Routen die eine Ressource abbilden zu gruppieren
- Angenommen wir haben eine Ressource User
 - Über HTTP Methoden können wir die Ressource User modifizieren
 - Create, Read, Update und Delete
 - Diese Sammlung an Routen können wir als eine Funktion in Fastify abbilden

Fastify

Grundlagen – Plugins

- In einer separaten Datei `user.routes.js` definieren wir die Routen für die Ressource User
- Wir deklarieren eine Funktion die als Parameter eine Fastify Instanz und zusätzliche Optionen entgegen nimmt
- In dieser Funktion können wir dann mit der Fastify Instanz die einzelnen Routen definieren
- Am Ende exportieren wir diese Funktion

Fastify

Grundlagen – Plugins



[Github](#)

```
1 let user = null;
2
3 async function userRoutes(fastify, options) {
4   fastify.get("/", async (request, reply) => {
5     if (user === null) {
6       reply.code(404).send();
7     } else {
8       return user;
9     }
10   });
11
12 fastify.post("/", async (request, reply) => {
13   user = { name: "Jens Reiner", id: 0 };
14   reply.code(201);
15   return user;
16 });
17
18 fastify.put("/", async (request, reply) => {
19   user = { name: "Nicht mehr Jens", id: 0 };
20   return user;
21 });
22
23 fastify.delete("/", async (request, reply) => {
24   user = null;
25   reply.code(200).send();
26 });
27 }
28
29 export default userRoutes;
30
```

Fastify

Grundlagen – Plugins

- In unserer Server Datei definieren wir jetzt keine Routen sondern registrieren ein Plugin
- Dafür importieren wir uns die **userRoutes** Funktion
- Und registrierten diese Routen als neues Plugin, im zweiten Parameter können wir noch einen URL Prefix angeben



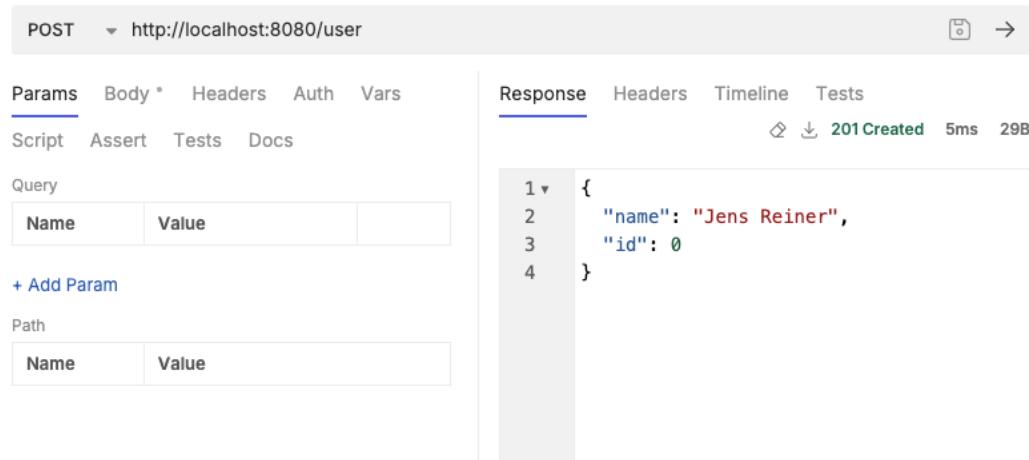
[Github](#)

```
1 import Fastify from "fastify";
2 import userRoutes from "./user.routes.js";
3
4 const fastify = Fastify({
5   logger: true,
6 });
7
8 fastify.register(userRoutes, { prefix: "/user" });
9
10 try {
11   await fastify.listen({ port: 8080 });
12 } catch (err) {
13   fastify.log.error(err);
14   process.exit(1);
15 }
16
```

Fastify

Grundlagen – Plugins

- Wenn wir jetzt eine HTTP Anfrage an unseren Server schicken können wir auf die User Endpunkte zugreifen, indem wir den Prefix /user verwenden



POST <http://localhost:8080/user>

Params Body * Headers Auth Vars

Script Assert Tests Docs

Query

Name	Value
------	-------

+ Add Param

Path

Name	Value
------	-------

Response Headers Timeline Tests

201 Created 5ms 29B

```
1 ▾ {  
2   "name": "Jens Reiner",  
3   "id": 0  
4 }
```

Fastify

Grundlagen – Daten an den Server schicken

- Viele Routen erwarten Daten von einem Client für die Verarbeitung
- In unserem user Beispiel möchten wir einen neuen User anlegen aber die Informationen wie bspw. Name vom User soll uns der Client schicken
- In HTTP können wir solche zusätzlichen Informationen über den Body mitschicken
- Fastify parsed Automatisch alle Anfragen und Antworten als JSON bzw. text/plain
 - <https://fastify.dev/docs/latest/Reference/ContentTypeParser/>

Fastify

Grundlagen – Daten an den Server schicken



[Github](#)

- Wir verändern unsere /user POST-Methode den Namen vom Benutzer aus der HTTP-Anfrage zu lesen
- Den Body erhalten wir über das request Objekt von Fastify

```
1 fastify.post("/", async (request, reply) => {
2   const body = request.body;
3   user = { name: body.name, id: 0 };
4   reply.code(201);
5   return user;
6 });
```

Fastify

Grundlagen – Daten an den Server schicken

- Anschließend können wir über unseren API Client einen Body für das Anlegen von einem User definieren

The screenshot shows a web-based API client interface. At the top, there's a header with the DHBW logo and the text "DHBW - Web Programmierung". Below it, there are buttons for "Safe Mode", "No Environment", and some icons. A tab bar shows "POST 03-user..." is selected. The main area has a "POST" button and the URL "http://localhost:8080/user". On the left, there are tabs for "Params", "Body *", "Headers", "Auth", "Vars", "Script", "Assert", "Tests", and "Docs". The "Body *" tab is active and set to "JSON". It contains a code editor with the following JSON:

```
1 {  
2   "name": "Nicht Jens"  
3 }
```

On the right, there are tabs for "Response", "Headers", "Timeline", and "Tests". The "Response" tab is active, showing a status of "201 Created" with a response time of "6ms" and a size of "28B". The response body is displayed as:

```
1 {  
2   "name": "Nicht Jens",  
3   "id": 0  
4 }
```

Fastify

Grundlagen – Daten an den Server schicken

- Woher weiß der Server aber jetzt, was für Daten ein Client im Body schicken?
- Wie kann ich die Daten, die mir ein Client schickt vorab validieren?

Fastify

Grundlagen – Schema Validierung

- Fastify hat die Möglichkeit alle eingehenden Anfragen zu validieren
- Fastify nutzt hierfür JSON Schema
 - <https://json-schema.org>
- JSON Schema ist eine Spezifikation, um JSON Daten Formate abzubilden
 - Wird u.a. in der Industrie eingesetzt
 - Formate können dann validiert werden etc.

Fastify

Grundlagen – Schema Validierung

- Schemas können über den zweiten Options Parameter an eine Fastify Route übergeben werden
- Zuerst definieren wir uns ein neues Options Objekt mit einer Schema Property
- In der Schema Property können wir den Body beschreiben, welchen wir für unsere Anfrage einen User anzulegen erwarten

Fastify

Grundlagen – Schema Validierung



[Github](#)

- Wir erwarten im Body ein Objekt mit einer Property name und string als Value
- Die Property name ist darüber hinaus required und muss übergeben werden

```
1 const userCreateOptions = {  
2   schema: {  
3     body: {  
4       type: 'object',  
5       properties: {  
6         name: { type: 'string' },  
7       },  
8       required: ["name"]  
9     }  
10   }  
11 }
```

Fastify

Grundlagen – Schema Validierung



[Github](#)

- Das userCreateOptions Objekt übergeben wir dann als zweiten Parameter an unsere Routen Definition

```
1 fastify.post("/", userCreateOptions,async (request, reply) => {
2   const body = request.body;
3   user = { name: body.name, id: 0 };
4   reply.code(201);
5   return user;
6 });
```

Fastify

Grundlagen – Schema Validierung

- Wenn wir jetzt eine Anfrage an unseren Endpunkt schicken und ein falsches JSON im Body mitschicken lehnt der Server die Anfrage automatisch ab

The screenshot shows a web-based testing interface for Fastify. At the top, there's a header with 'DHBW - Web Programmierung' and various status indicators like 'Safe Mode'. Below the header, a tab bar shows 'POST 04-user...' is selected. The main area has tabs for 'Params', 'Body *' (which is selected), 'Headers', 'Auth', and 'Vars'. Under 'Body', there are tabs for 'Script', 'Assert', 'Tests', and 'Docs', with 'JSON' currently selected. A red box highlights the JSON input field, which contains the following code:

```
1 {  
2   "username": "Jens Reiner"  
3 }
```

On the right side, there's a 'Response' tab selected, followed by 'Headers', 'Timeline', and 'Tests'. The response details are shown in a table:

400 Bad Request	17ms	120B
-----------------	------	------

A red box highlights the '400 Bad Request' status code. Below the table, the actual response body is displayed in a code editor-like format:

```
1 {  
2   "statusCode": 400,  
3   "code": "FST_ERR_VALIDATION",  
4   "error": "Bad Request",  
5   "message": "body must have required  
property 'name'"  
6 }
```

Fastify

Grundlagen – Schema Validierung

- Der Mehrwert von dieser Integration mit JSON Schema in Fastify ist, dass wir uns nicht darum kümmern müssen den Body in unserer Route zu validieren, Status Codes zu setzen etc.
- Fastify übernimmt für uns diese Aufgabe komplett und wir können die eigentliche Logik in unserer Route implementieren

Fastify

Grundlagen – Antwort Serialisieren



[Github](#)

- Über die Schema Definition können wir auch unserer Server Antwort serialisieren
- Über die Response Property definieren wir was für ein Objekt im HTTP Status 200 Fall zurückgegeben wird

```
1 const userCreateOptions = {
2   schema: {
3     body: {
4       type: "object",
5       properties: {
6         name: { type: "string" },
7       },
8       required: ["name"],
9     },
10    response: {
11      200: {
12        type: "object",
13        properties: {
14          name: { type: "string" },
15          id: { type: "integer" },
16        },
17      },
18    },
19  },
20};
```

Fastify

Grundlagen – Antwort Serialisieren

- Serialisieren von Daten hat in Fastify zwei wesentliche Vorteile:
- Die Server Antworten werden signifikant schneller (Faktor 2-3)
 - Alleine in unserem Beispiel sind es 1-2 ms schnellere Antworten
- Wir verhindern, dass sensitive Daten versehentlich zurück gegeben werden
 - Fastify gibt nämlich nur die Daten zurück, die im response beschrieben sind
 - Eine nicht definierte Property wird von Fastify in der Antwort entfernt

Fastify

Übung

- Erstellen Sie sich einen Fastify Server
- Erstellen Sie eine Router Funktion, welche eine Ressource Auto abbildet
 - Registrieren Sie diese Route in ihrem Server unter /car
- Definieren Sie sich ein JSON Schema für die Ressource Auto und implementieren Sie die Endpunkte für Create, Read, Update und Delete
- Verwenden Sie einen API Client um ihre Routen zu testen

Fastify

Weiterführende Links

- Plugins <https://fastify.dev/docs/latest/Reference/Plugins/>
- Validation and Serialization <https://fastify.dev/docs/latest/Reference/Validation-and-Serialization/>
- Routes <https://fastify.dev/docs/latest/Reference/Routes/>
- The hitchhikers Guide to Plugins <https://fastify.dev/docs/latest/Guides/Plugins-Guide/>
- Hooks <https://fastify.dev/docs/latest/Reference/Hooks/>

Beer API

Eine mögliche Implementierung mit Fastify

Beer API

Meilenstein 03

- Wir möchten jetzt den dritten Meilenstein für unsere Beer App Software Lösung implementieren
- Zusammen mit Fastify und den zuvor vorgestellten Konzepten von HTTP, Web Servern und co. implementieren wir jetzt eine Beer API

Beer API

Routen Definition



[Github](#)

- Zuerst definieren wir uns eine neue Router Funktion, welche die Funktionen aus Meilenstein 02 als API Endpunkte anbietet

```
1  async function beerRoutes(fastify, options) {  
2    fastify.post("/createBeer", async (request, reply) => {  
3    });  
4  
5    fastify.post("/drinkBeer", async (request, reply) => {  
6    });  
7  
8    fastify.post("/rateBeer", async (request, reply) => {  
9    });  
10  }  
11  
12  export { beerRoutes };
```

Beer API

Routen Definition



[Github](#)

- Und fügen diese Routen als Plugin der Fastify Server Instanz hinzu

```
1 import { beerRoutes } from "./routes/beer.routes.js";  
  
1 fastify.register(beerRoutes, { prefix: "/beer" });
```

Beer API

Schema Definition



[Github](#)

- Als nächstes erstellen wir uns ein Schema für das Beer Objekt
- Damit wir das Bier Objekt nicht jedes mal definieren müssen können wir auch Schema Definition wiederverwenden

```
1 const beerSchema = {  
2   $id: "beerSchema",  
3   type: "object",  
4   properties: {  
5     name: { type: "string" },  
6     taste: { type: "string" },  
7     brand: { type: "string" },  
8     amount: { type: "integer" },  
9     rating: { type: "integer" },  
10    },  
11  };
```

Beer API

Schema Definition



[Github](#)

- Das Allgemeine Beer Schema fügen wir dann unserer Server Instanz hinzu

```
1 import { beerSchema } from "./schemas/beer.schemas.js";  
  
1 fastify.addSchema(beerSchema);
```

Beer API

Schema Definition



[Github](#)

- Anschließend definieren wir uns die Optionen für die zuvor erstellten Routen

```
1 const createBeerOptions = {  
2   schema: {  
3     body: {  
4       type: "object",  
5       properties: {  
6         name: { type: "string" },  
7         taste: { type: "string" },  
8         brand: { type: "string" },  
9       },  
10      required: ["name", "taste", "brand"],  
11    },  
12    response: {  
13      200: {  
14        type: "object",  
15        properties: {  
16          beer: { $ref: "beerSchema#" },  
17        },  
18      },  
19    },  
20  },  
21};
```

```
1 const drinkBeerOptions = {  
2   schema: {  
3     body: { $ref: "beerSchema#" },  
4     response: {  
5       200: {  
6         type: "object",  
7         properties: {  
8           beer: { $ref: "beerSchema#" },  
9         },  
10      },  
11    },  
12  },  
13};
```

```
1 const rateBeerOptions = {  
2   schema: {  
3     body: {  
4       type: "object",  
5       properties: {  
6         rating: { type: "integer" },  
7         beer: { $ref: "beerSchema#" },  
8       },  
9       required: ["rating", "beer"],  
10     },  
11     response: {  
12       200: {  
13         type: "object",  
14         properties: {  
15           beer: { $ref: "beerSchema#" },  
16         },  
17       },  
18     },  
19   },  
20};  
21
```

Beer API

Schema Definition

- Mit \$ref können wir auf das Globale Beer Object Schema zugreifen
- Im Beispiel von drinkBeer erwarten wir das aktuelle Bier Objekt vom User
- Hier referenzieren wir das beerSchema



[Github](#)

```
1 const drinkBeerOptions = {
2   schema: {
3     body: { $ref: "beerSchema#" },
4     response: {
5       200: {
6         type: "object",
7         properties: {
8           beer: { $ref: "beerSchema#" },
9         },
10        },
11      },
12    },
13  };
```

Beer API

Route Handlers

- Die Funktionalität aus Meilenstein 02 können wir wiederverwenden für die eigentliche Logik
- Die benötigten Funktionen importieren wir dann einfach in unserer beer Route

```
1 import { createBeer, drinkBeer, rateBeer } from "../core/beer.js";
```

Beer API

Route Handlers



[Github](#)

- Anschließend können wir die Handler implementieren

```
1 fastify.post("/createBeer", createBeerOptions, async (request, reply) => {
2   const beerProps = request.body;
3
4   const beer = createBeer(beerProps);
5
6   reply.code(201);
7   return { beer: beer };
8 });
```

Beer API

Route Handlers



[Github](#)

- Anschließend können wir die Handler implementieren

```
1 fastify.post("/drinkBeer", drinkBeerOptions, async (request, reply) => {
2   const beer = request.body;
3
4   const updatedBeer = drinkBeer(beer);
5
6   return { beer: updatedBeer };
7 });
```

Beer API

Route Handlers



[Github](#)

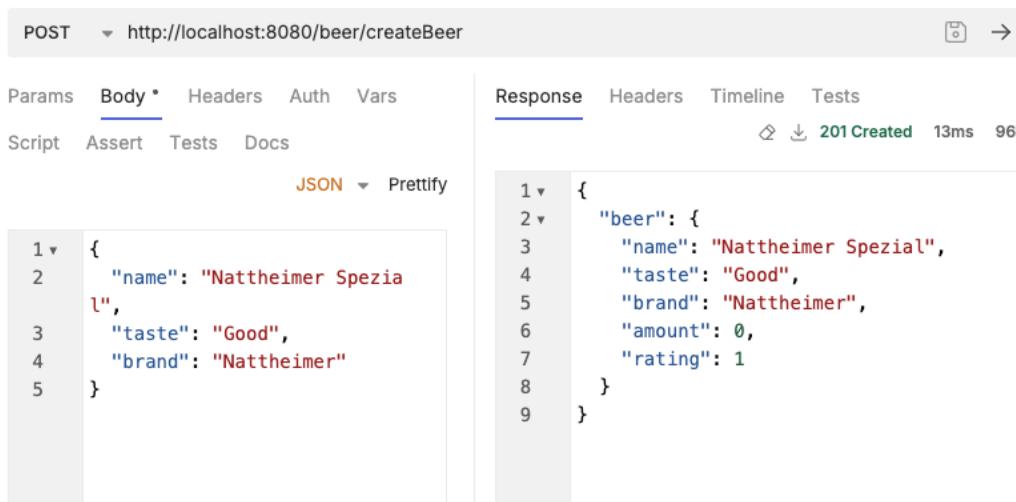
- Anschließend können wir die Handler implementieren

```
1 fastify.post("/rateBeer", rateBeerOptions, async (request, reply) => {
2   const rating = request.body.rating;
3   const beer = request.body.beer;
4
5   const updatedBeer = rateBeer(rating, beer);
6
7   return { beer: updatedBeer };
8 });
```

Beer API

Starten

- Wir können jetzt den Server starten (node server.js) und über einen API Client die entsprechenden Endpunkte ansprechen



POST <http://localhost:8080/beer/createBeer>

Params Body * Headers Auth Vars

Script Assert Tests Docs

JSON Prettify

```
1 {  
2   "name": "Nattheimer Spezial",  
3   "taste": "Good",  
4   "brand": "Nattheimer"  
5 }
```

Response Headers Timeline Tests

201 Created 13ms 96B

```
1 {  
2   "beer": {  
3     "name": "Nattheimer Spezial",  
4     "taste": "Good",  
5     "brand": "Nattheimer",  
6     "amount": 0,  
7     "rating": 1  
8   }  
9 }
```

Beer API

Übung

- Erweitern Sie die Lösung aus Meilenstein 03 um eine neue Router Funktion, welche die Funktionalität aus beers.js (Core Ordner) als API Endpunkte anbietet
- Als User möchte ich:
 - Alle bekannten Biere abfragen
 - Die am besten bewertete Biere abfragen
 - Das Bier, welches am meisten getrunken wurde
- Fallen Ihnen Probleme bei der Implementierung / Bedienung der Beer API auf?

Beer API

Verbesserte Version

- Im Ordner Meilenstein 04 finden Sie eine überarbeitete Version der Beer API die einige Probleme der ersten Version ausbessert
- Fällt Ihnen etwas auf?

Beer API

Herausforderungen

- Was für Herausforderungen sehen Sie wenn Sie eine Schnittstelle definieren?

REST Architektur

Selbst Erklärende Verteilte Systeme mit Webtechnologien

REST

Problemstellung

- Wie können Operationen in Ihrer Schnittstelle „selbsterklärend“ beschreiben?
 - Angenommen: Als Benutzer möchten Sie eine Maschine verändern
- Wie können Clients Informationen finden?
 - Angenommen: Sie möchten aus einer Liste ein einzelnes Element abfragen

REST

Definition

- REST = Representational State Transfer
- Software Architektur für das Beschreiben von Schnittstellen in einer Client – Server Architektur
 - Meistens für Maschine-zu-Maschine Kommunikation
- Kerngedanke: Wir verwenden Ansätze aus dem WWW für Schnittstellen

REST

Hintergrund

- REST Architektur von Roy Fielding 1994 vorgestellt in seiner Doktorarbeit
 - [Untitled Document \(uci.edu\)](#)
 - [Roy T. Fielding \(gbiv.com\)](#)
- Co-Founder von Apache, einem der bekanntesten Webserver

REST

Konzepte

- Geringe Kopplung zwischen Client und Server, um so viele Clients wie möglich zu erreichen
- Hohe Abstraktion zwischen Server Implementierung Details (z.B. Datenbank Zugriff) und auf was der Client zugreifen kann
- REST definiert **Ressourcen** auf die der Client zugreifen kann
 - Ressourcen beinhalten alles mögliche an Informationen (Bilder, Datenbank Abfragen, Services etc.)

REST

Konzepte

- Ein Client kann eine **Ressource nur über eine URI anfragen** und der Server antwortet mit einer **Repräsentation von dieser Ressource**.
- Die einheitliche Repräsentation von einer Ressource erlaubt, dass unterschiedliche Clients mit der selben Ressource arbeiten können
- Die Repräsentation einer Ressource wird in einem **Hypertext Format** übertragen

REST

Definition - RESTful

- Die REST Architektur definiert 6 Prinzipien die beachtet werden müssen
- Sind diese erfüllt wird eine Software auch als RESTful bezeichnet
 - In der Literatur unterscheiden sich die Definitionen wann eine Schnittstelle als RESTful bezeichnet wird

REST

Definition - RESTful

- Werden diese eingehalten erhält eine Software folgende nicht funktionale Eigenschaften:
 - Performance
 - Scalability
 - Simplicity
 - Modifiability
 - Visibility
 - Portability
 - Reliability

REST

Prinzipien – Client Server Model

- Dienst wird auf Server bereitgestellt, erst bei Anfrage des Clients wird dieser aktiviert
 - Separation of concerns: User Interface ist entlöst von der Daten Implementierung
 - Antwort kann in jedem beliebigen Format erfolgen, welches über HTTP übertragen werden kann (z.B. HTML, JSON, XML etc.)
-
- Begünstigt Skalierung
 - Begünstigt Portability (User Interface kann beliebig ausgetauscht werden)

REST

Prinzipien – Zustandslosigkeit

- Jede Nachricht enthält alle notwendigen Informationen und ist in sich geschlossen
- Nachrichten können unabhängig von anderen verarbeitet werden
- Erlaubt massive Skalierung der Dienste
- “Versteckt” Komplexität wie beispielsweise offene Datenbankverbindungen

REST

Prinzipien – Caching

- Ressourcen müssen nicht unabdingt ständig erneut angefragt werden
- Minimiert aufwendige Netzwerk Kommunikation zwischen Client und Server
- Jede Ressource kann als cachable definiert werden oder nicht
- Content Delivery Networks (CDN) können als Cache fungieren zwischen Client und Servern

REST

Prinzipien – Multi Layered

- Komplexität und Architektur wird hinter einer einzelnen Ressource “versteckt”
- Definierte Methoden (CRUD mit HTTP-Methoden) erlauben Transparenz
- Clients wissen nicht ob Sie mit dem “echten” Server sprechen oder einem Proxy / Load Balancer
- Server können mit weiteren Servern sprechen, um eine Antwort für den Client zu generieren
 - (Server wird zu Client und spricht mit anderem Server)
- Sicherheitsmechanismen können zwischen Application Logic eingezogen werden

REST

Prinzipien – Uniforme, standardisierte Schnittstelle

- Jede Ressource hat eine feste, eindeutige Adresse (URI) welche unveränderbar ist
- Server Daten Repräsentationen spiegeln nicht die Antwort für den Client wieder
 - Server kann Datenbank Daten als HTML, JSON oder z.B. XML übermitteln
 - Clients können Daten in verschiedenen Formaten anfordern
- Ressourcen Manipulation mittels Repräsentation der Daten
 - Client hat genügend Informationen für das verändern einer Ressource (Delete, Edit etc.)
- Selbst-beschreibende Nachrichten.
 - Jede Nachricht beinhaltet genügend Informationen für das verarbeiten einer Nachricht
 - Media-type Feld damit Client die Antwort richtig parsen kann

REST

Prinzipien – Uniforme, standardisierte Schnittstelle

- Hypermedia as the Engine of Application State (HATEOAS)
 - Ein Client mit einer initialen URI für eine REST Applikation kann alle verfügbaren Ressourcen entdecken
 - Navigation zwischen Ressourcen der Schnittstelle erfolgt nur über in der Ressource verfügbare Links
 - Wie ein Browser “navigiert” der Client durch die Dienste
 - “Self-Discovery” ist möglich d.h. mit einzelnen Einstiegspunkt können alle Ressourcen gefunden werden
 - Ermöglicht lose Bindung innerhalb der Schnittstelle. Dadurch Skalierung und weniger Abhängigkeiten
 - Bspw. [SWAPI - The Star Wars API](#)

REST

Details

- REST ist ein Architektur Pattern und hat keinen offiziellen Standard
- REST Schnittstellen definieren alle notwendigen Methoden für das bearbeiten von Ressourcen (CRUD)
 - **Create**
 - **Read**
 - **Update**
 - **Delete**

REST

Details

- Web Schnittstellen die REST anwenden verwenden die Semantischen HTTP Methoden für das bearbeiten von Ressourcen
 - GET – Liefert die Repräsentation der angefragten Ressource
 - POST – Verarbeiten von einer Repräsentation von einer Ressource definiert in der Anfrage
 - PUT – Anlegen oder updaten einer bestehenden Ressource
 - PATCH – Bedingtes Update von einer Ressource
 - DELETE – Löschen einer Ressource

REST

Details

- Sicherheitstechniken (z.B. Authentifizierung) sind nicht explizit vorgeschrieben.
 - In der Praxis werden die gängigen Muster aus der Web-Programmierung (Cookies, Tokens etc.) jedoch verwendet
- Reifegrad und Ausprägung der vorgestellten Prinzipien ist in der Praxis extrem unterschiedlich

REST

Maturity Model

- 2008 wurde von Leonard Richardson folgendes Maturity Model für Web APIs vorgestellt:
 - [Richardson Maturity Model \(martinfowler.com\)](http://martinfowler.com)
 - Level 0: Define one URI, and all operations are POST requests to this URI.
 - Level 1: Create separate URIs for individual resources
 - Level 2: Use HTTP methods to define operations on resources.
 - Level 3: Use hypermedia (HATEOAS)
- Erst ab Level 3 spricht man von einer RESTful API. In der Praxis scheitern aber viele Schnittstellen ab Level 2
 - [REST APIs must be hypertext-driven » Untangled \(gbiv.com\)](http://gbiv.com)
 - [</> htmx ~ How Did REST Come To Mean The Opposite of REST?](http://html5doctor.com)

REST

VOR- und Nachteile

- (+) Klar und “selbst-erklärende” Struktur der Ressourcen
- (+) Einfache Schnittstellen und Transparenz
- (+) Skalierbarkeit in der Performance und Komplexität
- (+) Nutzung der vorgestellten HTTP Grundlagen

REST

VOR- und Nachteile

- (-) Nicht alle REST-Schnittstellen besitzen selben Reifegrad
- (-) Unklares Vorgehen für Index-Ressourcen
- (-) Unklares Vorgehen für verschachtelte Ressourcen
- (-) Filterung, Sortierung und Aggregation sind unnötig komplex
- (-) Kann zu sehr vielen einzelnen HTTP-Anfragen führen

REST

Alternativen

- **SOAP** (Nachrichten-Basiert)
 - Austausch von Nachrichten im XML-Format über z.B. HTTP
 - Body enthält entweder Daten oder Anweisungen für den Empfänger
 - Nicht strikt Client-Server. Ebenso mit SMTP etc. möglich

REST

Alternativen

- **GraphQL** (“SQL fürs Web”)
 - Datenabfrage und Manipulationssprache. Entwickelt von Facebook
 - Client sendet “Query” an Webserver. Dieser verarbeitet Query und antwortet mit Daten
 - Definition der gewünschten Daten “ad-hoc” im Query
 - Reduziert Komplexität bei Filterung, Sortierung und angepassten Daten
 - Erhöht Skalierbarkeit da eine einzelne Anfrage alle Daten für eine Ansicht im User Interface erfragen kann
 - Komplexität in der Berechtigung und Verschachtelung
 - [GraphQL | A query language for your API](#)

REST

Übung

- Erkunden von REST APIs
 - [GitHub REST API - GitHub Docs](#)
 - [OMDb API - The Open Movie Database](#)
 - [SWAPI - The Star Wars API](#)
 - [Studio Ghibli API](#)
 - [PokéAPI \(pokeapi.co\)](#)
- Was fällt Ihnen auf?
 - Dokumentation der Schnittstelle?
 - Einhaltung aller REST-Paradigmen?
 - Auffindbarkeit der Ressourcen?
 - Filtern?

REST

Übung

- Weitere APIs
 - [Web API | Spotify for Developers](#)
 - [REST Admin API reference \(shopify.dev\)](#)
 - [Stack Exchange API](#)
 - [Mercedes-Benz /developers – The API platform by Mercedes-Benz](#)

REST

Zusammenfassung

- Ähnlich wie Webseiten können beliebige Daten zum Austausch zwischen Services übertragen werden. Dies erfolgt immer im Client-Server-Modell
- REST nutzt die Eigenschaften des Webs dafür, besonders Hyperlinks und die Zustandslosigkeit jeder Anfrage
- Vorteil ist die Skalierbarkeit und Einfachheit. Nachteil Komplexität bei Filterung und Reifegrade

REST

Fallbeispiel – Beer API

- Ist die zuvor implementierte Beer API aus Meilenstein 03 RESTful?
- Wir prüfen einmal die vorgestellten Prinzipien

REST

Fallbeispiel – Client Server Model

- Unsere Beer API wird über einen Server bereitgestellt und ist erst aktiv bei einer Client Anfrage
- Das User Interface kann komplett unabhängig implementiert werden

REST

Fallbeispiel – Zustandslosigkeit

- Anfragen können Unabhängig voneinander gestellt und verarbeitet werden
- Server nimmt ein Bier entgegen und speichert sich diese nicht

REST

Fallbeispiel – Caching

- Nicht implementiert kann aber über den HTTP Header Cache-Control gesteuert werden
 - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control>

REST

Fallbeispiel – Multi Layered

- Eigentliche Logik für das Verarbeiten von einem Bier ist abstrahiert in separaten Funktionen
- Bier Ressource kann über HTTP Anfragen modifiziert werden

REST

Fallbeispiel – Uniform Standardisierte Schnittstelle

- *Jede Ressource hat eine feste, eindeutige Adresse welche unveränderbar ist*
 - Verletzt – Wir beschreiben nicht die Ressourcen sondern die Funktionalität
 - Bspw. /beer/createBeer
- *Hypermedia as the Engine of Application State (HATEOAS)*
 - Unser Client kann mit einer URI nicht alle Ressourcen entdecken
 - Unser Server liefert kein HTML sondern JSON Daten

REST

Fallbeispiel – Beer API

- Die vorgestellte Beer API ist also nicht RESTful
- Nach Maturity Modell verletzten wir u.a. Level 3 (HATEOAS)

REST

Fallbeispiel – HATEOAS

- Was ist die Besonderheit wenn eine REST API Level 3 erreicht?

REST

Fallbeispiel – HATEOAS

```
1 <!DOCTYPE html>
2 <html lang="de">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Beispiel für REST mit HATEOAS</title>
7 </head>
8 <body>
9   <h1>Bierkatalog</h1>
10  <p>Wählen Sie ein Bier aus der Liste, um mehr Informationen zu sehen.</p>
11
12  <ul>
13    <li><a href="/beers/1">Pilsner</a></li>
14    <li><a href="/beers/2">Weizenbier</a></li>
15    <li><a href="/beers/3">Dunkelbier</a></li>
16  </ul>
17
18  <p><a href="/beers/new">Neues Bier hinzufügen</a></p>
19 </body>
20 </html>
```

```
1  {
2    "beer": {
3      "id": 1,
4      "name": "Pilsner",
5      "type": "Lager",
6      "brand": "Brauerei A",
7      "rating": 4.5
8    },
9    "_links": {
10      "self": {
11        "href": "/beers/1"
12      },
13      "next": {
14        "href": "/beers/2"
15      },
16      "add": {
17        "href": "/beers/new"
18      }
19    }
20  }
21 }
```

REST

Fallbeispiel – HATEOAS

- Für die Antwort einer Ressource als HTML Dokument benötigen Sie keine weitere Software oder implementierte Logik
 - Mit Ausnahme von einem Web Browser
- Das HTML Dokument ist selbst erklärend und als User kann ich einfach weitere Ressourcen finden

REST

Fallbeispiel – HATEOAS

- Die Antwort als JSON benötigt einen speziellen Client und Logik, um die Repräsentation von einer Ressource aufzubereiten
- Die JSON Antwort muss dokumentiert werden, was einzelne Properties bedeuten
 - In der Realität bspw. mittels Open API Standard

REST

RESTful in der Praxis

- In der modernen Softwarewelt gibt es aber beinahe nur noch APIs die JSON als Daten zurückgeben und alle das Maturity Level 3 verletzen
- Das ist aber auch nicht schlimm, denn in der modernen Softwarewelt sprechen primär Services mit anderen Services also eine Maschine zu Maschine Kommunikation
- Hierfür benötigen Sie nicht die Repräsentation von einer Ressource als HTML und damit Menschen Lesbar

REST

RESTful in der Praxis

- Aber wieso verwendet die Industrie den Begriff RESTful für APIs die in 99% der Fälle nicht RESTful sind?
- Als REST vorgestellt wurde (Jahr 2000) kam AJAX (Asynchronous JavaScript and XML) und andere RPC Patterns gerade auch erst auf den Markt
 - AJAX der Browser kann HTTP Requests ausführen
 - XML-RPC war Teil von SOAP
- Gleichzeitig um diese Zeit wurde JSON vorgestellt und in Web Technologien integriert
 - Und ungefähr 2008 hat sich JSON so langsam etabliert neben XML

REST

RESTful in der Praxis

- Mit der wachsenden Bedeutung von JSON wurden auch immer mehr APIs im JSON Format definiert
 - Und der Bedeutung von Service zu Service Kommunikation
- Und damit wurde nur Level 2 im Maturity Level von Martin Fowler umgesetzt und nicht mehr Level 3
 - Es gibt zwar APIs die versuchen die Konzepte aus Level 3 in die JSON Response zu integrieren (vgl. `_links` im Beispiel zuvor)
 - Aber das Problem bleibt: Sie benötigen eine Logik welche die JSON Antwort verarbeiten kann

REST

RESTful in der Praxis

- „*REST was the opposite of SOAP, JSON APIs weren't SOAP, therefore JSON APIs were REST.*“
 - [</> htmx ~ How Did REST Come To Mean The Opposite of REST?](#)
 - [REST APIs must be hypertext-driven » Untangled \(gbiv.com\)](#)
- 2010 begann dann der Trend von Single Page Applications (SPA)
 - Single Page Applications hatten deutlich komplexere Use Cases die mit einfaches HTML nicht mehr abgebildet werden konnten

REST

REST in der Vorlesung

- Wir werden in der Vorlesung nicht Level 3 erreichen
 - Im Gegenteil alle weiteren Beispiele sind nicht RESTful nach originaler Definition
- Wir verwenden aber trotzdem die Konzepte aus einer REST Architektur und versuchen mindestens Level 2 zu erreichen
- Wichtig ist vor allem, dass Sie saubere Schnittstellen definieren und damit arbeiten können. Die Konzepte aus REST bieten sich dafür hervorragend an

REST

Ausblick

- Und vielleicht ist es an der Zeit einen neuen moderneren „REST“ Standard speziell für JSON APIs zu definieren, der die Software Anforderungen von heute gerecht wird
- Technologien wandeln sich und auch Software Design Patterns
- Wichtig ist nur, dass die Person, mit der Sie sprechen denselben Kontext von einem Begriff wie Sie haben

Backend Development

Implementierung von einer vollständigen API

Backend Development

Hintergrund

- In einem neuen Meilenstein möchten wir unser Software Lösung der Beer API vollständig abrunden
- Fallen Ihnen Themen ein, die bei der Beer API noch nicht gelöst sind?

Backend Development

Meilenstein 05

- Wir möchten unsere Daten konsistent speichern und es soll für Clients möglich sein, nach Bieren zu suchen und filtern. Auch möchten wir die vorgestellte REST Architektur in unsere Lösung integrieren.
- Am Ende soll die Beer API selbstständig Biere verwalten und speichern können. Über eine REST Schnittstelle können Clients Daten austauschen.

Backend Development

Datenbank – Organisatorisches

- Die Konzepte hinter Datenbanken werden in einer anderen Vorlesung genauer betrachtet
- Die Web Programmierung eignet sich aber besonders gut, um ihr Wissen rund um Datenbanken einmal in der Praxis auszuprobieren

Backend Development

Datenbank – Integration

- NodeJS und Fastify sind Datenbank agnostisch d.h. es kann eine beliebige Datenbank Implementierung verwendet werden
- Node und auch Fastify unterstützen dabei verschiedene Datenbanktreiber
 - Die meisten bekannten Datenbanken gibt es als fertige Pakete auf NPM
- Für die Vorlesung nutzen wir SQLite

Backend Development

Datenbank – Hintergrund SQLite

- SQLite ist eine leichtgewichtige, serverlose relationale Datenbank.
- SQLite speichert alle Daten in einer einzigen Datei
- SQLite ist die am weitesten verbreitete Datenbank auf der Welt und Sie finden SQLite in jedem Mobil Telefon auf der Welt und in fast allen Computern
- Das SQLite Format ist cross-plattform Fähig und die Developers hinter SQLite versprechen einen Support bis 2050 (!)

Backend Development

Datenbank – Hintergrund SQLite

- Für SQLite müssen Sie keine extra Software auf Ihrem Computer installieren
- Anwendungsbeispiele für SQLite sind lokale oder kleinere Softwarelösungen, die keine komplexen Serverdatenbanken benötigen und geringe Ressourcenanforderungen haben
 - SQLite benötigt mit allen Features ~750KiB Speicher
- Besonders bei Mobilen Apps, Embedded Systemen und Desktop Anwendungen kommt SQLite zum Einsatz

Backend Development

Datenbank – Hintergrund SQLite

- Die einfache Installation von SQLite macht es zum perfekten Kandidaten für unsere Vorlesung
- Sie können alle gängigen SQL Statements definieren und verwenden
- Mit NodeJS Version 22 kam sogar eine erste SQLite Integration direkt in NodeJS
 - <https://nodejs.org/api/sqlite.html>
 - Aktuell müssen Sie SQLite noch über experimental Flags aktivieren

Backend Development

Datenbank – Installation

- Aus diesem Grund nutzen wir für die Vorlesung noch eine Third Party Integration von SQLite nämlich better-sqlite3 <https://github.com/WiseLibs/better-sqlite3/tree/master>

Backend Development

Datenbank – Fastify Integration

- Wir möchten unseren Fastify Server um eine Datenbank Integration erweitern
- Aus diesem Grund schreiben wir uns ein eigenes Fastify Plugin, welches die Datenbank Verbindung aufbaut und das Datenbank Objekt an die Fastify Insatz hängt

Backend Development

Datenbank – Fastify Integration



[Github](#)

- Im ersten Schritt installieren wir *better-sqlite3* und *fastify-plugin* über NPM
- Als nächstes erstellen wir eine JavaScript Datei, welche sich um die Datenbank Verbindung kümmert und das Fastify Plugin definiert
- In dieser Datei definieren wir den Dateipfad für unsere Datenbank und die erste Tabelle

```
1 import fp from "fastify-plugin";
2 import Database from "better-sqlite3";
3
4 // Defines where sqlite will store the database on the given path on the filesystem
5 const filePath = "./database/beer.db";
6
7 // Defines the SQL statement to create the beer table
8 const createTableStatement = `
9   CREATE TABLE IF NOT EXISTS beers (
10     id INTEGER PRIMARY KEY AUTOINCREMENT,
11     name VARCHAR(50) NOT NULL,
12     taste VARCHAR(50) NOT NULL,
13     brand VARCHAR(50) NOT NULL,
14     amount INTEGER NOT NULL,
15     rating INTEGER NOT NULL
16   );
17 `;
18
```

Backend Development

Datenbank – Fastify Integration

- Anschließend schreiben wir uns eine neue Funktion, welche die Datenbank als Fastify Plugin exportiert
- Wenn das Plugin geladen wird, führen wir auch das createTableStatement aus



[Github](#)

```
1 /**
2  * Adds a plugin to the fastify instance and creates the database instance.
3  * You only need to call this once in the main register flow (server.js)
4  * @param {object} fastify - The fastify instance
5  * @param {object} options - The fastify plugin options
6  * @param {function} next - The fastify next callback
7 */
8 function dbConnector(fastify, options, next) {
9   // Creates a new database instance
10  const db = new Database(filePath);
11
12  // Creates the beer table if it does not exist
13  db.exec(createTableStatement);
14
15  // Adds the database instance to the fastify instance
16  fastify.decorate("db", db);
17
18  // Closes the database connection when the server is closed
19  fastify.addHook("onClose", (fastify, done) => {
20    db.close();
21    done();
22  });
23
24  next();
25}
26
27 /**
28  * Exposes the plugin to fastify using the `fp` plugin helper
29  */
30 export default fp(dbConnector);
```

Backend Development

Datenbank – Fastify Integration



[Github](#)

- Jetzt registrieren wir unsere Datenbank als Plugin in der server.js Datei

```
1 import dbConnector from "./database/database.js";
2 fastify.register(dbConnector);
```

- Jetzt können wir überall wo wir Zugriff auf die Fastify Instance haben auch auf die Datenbank zugreifen

Backend Development

Datenbank – Beer Core Umbau

- Wir können jetzt unsere Beer API Core Implementierung umbauen und alle Biere in der Datenbank speichern
- Aus diesem Grund brauchen wir auch nicht mehr die beers.js Implementierung
- Stattdessen verändern wir die beer.js Core Implementierung und nutzen die Datenbank für das Speichern und Lesen der Daten

Backend Development

Datenbank – Beer Core Umbau

- Hinweis: Auf den Folien ist nur der Umbau für die Funktion ein neues Bier anzulegen abgebildet.
- Die vollständige Lösung finden Sie aber auf Github speziell im Meilenstein 04

Backend Development

Datenbank – Beer Core Umbau



[Github](#)

- Die Funktion **createBeer** wird um einen neuen Parameter *fastify* erweitert.
 - An diesen Parameter übergeben wir die aktuelle Fastify Instanz, um Zugang zum Datenbank Objekt zu bekommen

```
1 function createBeer(fastify, beerProps)
```

- Anschließend definieren wir zwei Prepared SQL Statements, um ein neues Bier anzulegen und das angelegte Bier anschließend abzufragen

```
1 const insertIntoStatement = fastify.db.prepare(
2   `INSERT INTO beers (name, taste, brand, amount, rating) VALUES (?, ?, ?, ?, ?)`
3 );
4 const selectStatement = fastify.db.prepare(
5   `SELECT * FROM beers WHERE id = ?`
6 );
```

Backend Development

Datenbank – Beer Core Umbau



[Github](#)

- Prepared Statements sind vorab kompilierte SQL Anweisungen die das Ziel haben SQL Abfragen effizienter und sicherer zu machen.
- Die eigentlichen Parameter werden erst zu einem späteren Zeitpunkt gesetzt, wodurch SQL Injections deutlich komplizierter werden
- Mehr in anderen Vorlesungen

Backend Development

Datenbank – Beer Core Umbau



[Github](#)

- Als nächstes erstellen wir wie gewohnt unser Beer Objekt

```
1 const beerToCreate = {  
2   name: beerProps.name,  
3   taste: beerProps.taste,  
4   brand: beerProps.brand,  
5   amount: 0,  
6   rating: 1,  
7 };
```

Backend Development

Datenbank – Beer Core Umbau



[Github](#)

- Mit dem Prepared Statement und dem Beer Objekt versuchen wir jetzt ein neues Bier anzulegen
- Die Information über die zuletzt hinzugefügte Row speichern wir uns, diese Information nutzen wir zum Abfragen des eben angelegten Bieres

```
1 try {
2   const { name, taste, brand, amount, rating } = beerToCreate;
3   const info = insertIntoStatement.run(name, taste, brand, amount, rating);
4
5   const createdBeer = selectStatement.get(info.lastInsertRowid);
6   return createdBeer;
7 } catch (error) {
8   fastify.log.error(error);
9   return null;
10 }
```

Backend Development

Datenbank – Beer Core Umbau



[Github](#)

- Fällt Ihnen etwas auf bei der Syntax in Zeile 2?

```
1 try {
2   const { name, taste, brand, amount, rating } = beerToCreate;
3   const info = insertIntoStatement.run(name, taste, brand, amount, rating);
4
5   const createdBeer = selectStatement.get(info.lastInsertRowid);
6   return createdBeer;
7 } catch (error) {
8   fastify.log.error(error);
9   return null;
10 }
```

Backend Development

JavaScript – Destructuring Assignment



[Github](#)

- Diese Art Variablen zu definieren, wird als **Destructuring Assingment** bezeichnet und ermöglicht es uns, Objekte oder Arrays in ihre Bestandteile zu zerlegen und direkt Variablen daraus zu erstellen
- Der Vorteil ist u.a. wir müssen nicht direkt das Beer Objekt referenzieren, um Zugriff auf die Werte zu bekommen, sondern können direkt die Werte in eigene Variablen speichern

Backend Development

Datenbank – Beer Core Umbau



[Github](#)

- Zusammenfassend können wir also die neue **createBeer** Funktion so beschreiben:
- Wir bereiten zwei SQL Statements vor, einmal um das neue Bier in die Tabelle beers zu schreiben und das andere, um das neu angelegte Bier direkt abzufragen
- Beide Statements werden ausgeführt und im Erfolgsfall wird das neu angelegte Bier an den Client zurückgegeben
- Im Fehlerfall loggen wir den Fehler und geben explizit null zurück

Backend Development

Datenbank – Beer Routes Umbau



[Github](#)

- Auf der Routen Definition zum Anlegen von einem Bier müssen wir nicht viel ändern
- Wir übergeben die aktuelle Fastify Instanz an die Funktion und prüfen ob ein Bier erfolgreich angelegt wurde – Für den Fall, dass der Server einen Fehler macht geben wir explizit 500 zurück

```
1 fastify.post("/", createBeerOptions, async (request, reply) => {
2   const beerProps = request.body;
3
4   const beer = createBeer(fastify, beerProps);
5
6   if (!beer) {
7     reply.code(500);
8     return { error: "Could not create beer" };
9   }
10
11  reply.code(201);
12  return { beer: beer };
13});
```

Backend Development

Datenbank – Testen der neuen Anlege Funktion

- Anschließend können wir über Bruno ein neues Bier anlegen

The screenshot shows the Postman interface with a successful API call. The request method is POST, directed to <http://localhost:8080/beer>. The response status is 201 Created, with a response time of 19ms and a size of 107B. The request body is a JSON object:

```
1 {  
2   "name": "Goldochsen Kellerbier",  
3   "taste": "Super",  
4   "brand": "Goldochsen"  
5 }
```

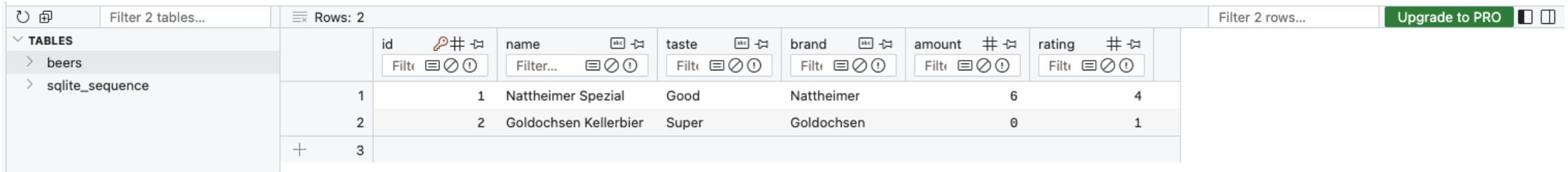
The response body is also a JSON object:

```
1 {  
2   "beer": {  
3     "id": 2,  
4     "name": "Goldochsen Kellerbier",  
5     "taste": "Super",  
6     "brand": "Goldochsen",  
7     "amount": 0,  
8     "rating": 1  
9   }  
10 }
```

Backend Development

Datenbank – Testen der neuen Anlege Funktion

- Mit der VSCode Extension SQLite Viewer können wir uns auch die Datenbank direkt anschauen, indem wir einfach auf die beer.db Datei klicken
 - Link zur Extension <https://marketplace.visualstudio.com/items?itemName=qwtel.sqlite-viewer>



The screenshot shows the SQLite Viewer extension interface in VSCode. On the left, there's a sidebar with a 'TABLES' section containing 'beers' and 'sqlite_sequence'. The main area displays the 'beers' table with the following data:

	id	name	taste	brand	amount	rating	
	1	Nattheimer Spezial	Good	Nattheimer	6	4	
	2	Goldochsen Kellerbier	Super	Goldochsen	0	1	
+	3						

Backend Development

Datenbank – Fazit

- Wenn wir jetzt unseren Server neu starten, werden alle Biere in der Datenbank gespeichert und sind anschließend wieder verfügbar
- Den Vollständigen Umbau aller Funktionen mit der Datenbank zu arbeiten finden Sie hier <https://github.com/wasdJens/dhw-webprogrammierung-wwi/blob/master/10-Beer-Solution/05-milestone/core/beer.js>

Backend Development

Datenbanken – Hinweis Prüfungsleistung

- Für die Prüfungsleistung müssen Sie keine Datenbank Einsetzen
- Für die Note „sehr gut“ erwarten wir aber eine Datenbank Implementierung
- Sie können SQLite für die Prüfungsleistung nutzen
- Alternative Datenbanken dürfen Sie auch benutzen, müssen aber sicherstellen, dass diese als Docker Container in der Abgabe hinterlegt sind
 - Wir installieren keine Datenbank selbst

Backend Development

Meilenstein 05 – Filterung

- Gegeben sind zwei neue Anwendungsfälle für unsere Bier Schnittstelle
 - Sie möchten genau ein bestimmte Bier abfragen
 - Sie möchten die Liste an Bieren filtern nach bestimmten Eigenschaften

Backend Development

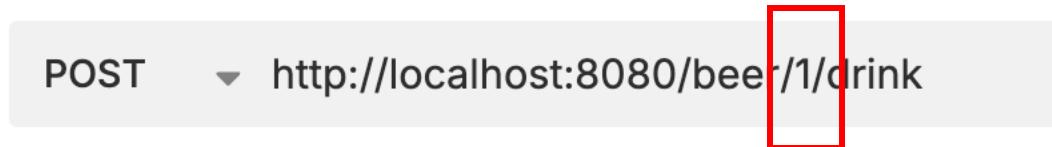
Filterung – Routen Parameter

- Im Zuge von Meilenstein 04 und 05 haben wir ein neue ID Property für unser Bier hinzugefügt
- Mit dieser ID kann ich ein Bier genau zuordnen
- Als Client kann ich diese ID als Teil der URL angeben, um Informationen über genau dieses Bier zu bekommen
 - Oder um genau dieses Bier zu trinken

Backend Development

Filterung – Routen Parameter

- Die ID ist dabei Teil der URL und wird vom Client übermittelt



- Woher weiß der Server aber jetzt welches Bier wir genau trinken möchten?

Backend Development

Filterung – Routen Parameter



[Github](#)

- In Fastify (Und anderen Frameworks) können wir eine Route mit einem oder mehreren Parametern definieren
- Der Parameter wird dabei mit : definiert gefolgt vom Namen

```
1 fastify.get("/:id", getBeerOptions, async (request, reply) => {
```

Backend Development

Filterung – Routen Parameter



[Github](#)

- Fastify schreibt alle Parameter automatisch in das **.params** Objekt auf dem **request** Objekt
- Wichtig: Alle params Property werden als String geparsed – In unserem Beispiel brauchen wir aber die ID als Zahl deshalb parsen wir diese in einen integer

```
1 const id = parseInt(request.params.id, 10);
```

Backend Development

Filterung – Routen Parameter

- Routen können eine beliebige Anzahl an Routen Parametern beinhalten
 - Bspw. `/example/:orgId/:userId/deactivate`
- Mehrere Request Parameter können dann auch mit der Destructuring Assignment Syntax in Variablen umgewandelt werden

Backend Development

Filterung – Query Parameter

- Wenn wir jetzt alle Biere nach Eigenschaften filtern möchten als Client könnten wir entweder Routen Parameter einsetzen oder eine Art Filter Objekt im Body als Payload mitschicken
- In der Web Entwicklung hat sich aber das Konzept von Query Strings durchgesetzt
- Als Client kann ich an die URL eine Query anhängen, die vom Server interpretiert werden kann

Backend Development

Filterung – Query Parameter

- An den normalen Endpunkt wird an das Ende ein **?** gesetzt gefolgt von einem key und einem Wert nach dem wir Filtern möchten

```
GET      ▾  http://localhost:8080/beer?taste=good
```

Backend Development

Filterung – Query Parameter

- Wenn wir jetzt den Request absenden bekommen wir folgendes zurück

The screenshot shows a REST client interface with the following details:

- Request URL:** GET http://localhost:8080/beer?taste=good
- Params:** taste: good
- Response Headers:** Headers, Timeline, Tests
- Response Status:** 200 OK, 18ms, 195B
- Response Body:**

```
[{"name": "Nattheimer Spezial", "taste": "Good", "brand": "Nattheimer", "amount": 6, "rating": 4, "id": 1}, {"name": "Goldochsen Kellerbier", "taste": "Super", "brand": "Goldochsen", "amount": 0, "rating": 1, "id": 2}]
```

Backend Development

Filterung – Query Parameter

- Query Strings sind optionale Teile, die an eine URL gehängt werden können
- Ob der Server aber diesen Query String interpretieren kann, liegt ganz an der Implementierung
- In unserem Beispiel haben wir kein Handling für Query Parameter implementiert, dementsprechend bekommen wir die normale **getBeers** Antwort zurück

Backend Development

Filterung – Query Parameter

- Das Parsen von Query Strings unterscheidet sich in unterschiedlichen Frameworks / Implementierungen
- Fastify nutzt das Node Interne querystring Module im Hintergrund.
 - <https://nodejs.org/api/querystring.html>
 - Fastify bietet aber auch die Möglichkeit einen eigenen Parser zu implementieren
- Im Browser gibt es bspw. URLSearchParams als Interface
<https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams>

Backend Development

Filterung – Query Parameter



[Github](#)

- Fastify liefert uns alle Query Parameter über das **query** Objekt auf dem **request** Objekt
- Wenn wir nach dem Geschmack filtern möchten, können wir dann die Liste mit allen Bieren durchgehen und danach filtern

```
1 fastify.get("/beer", async function handler(request, reply) {
2   const { taste } = request.query;
3   if (taste) {
4     const filteredBeers = beers.filter((beer) => beer.taste.toLowerCase().includes(taste.toLowerCase()));
5     return { beers: filteredBeers };
6   }
7   return { beers };
8});
```

Backend Development

Filterung – Query Parameter

- Query Parameter können auch verkettet werden mit einem & Symbol in der URL

```
GET      ▾ http://localhost:8080/beer?taste=good&rating=4
```

- Für diesen Filter müssen wir aber auch unsere Implementierung entsprechend updaten, um alle Query Parameter abzubilden
- Wir können aber über ein JSON Schema in Fastify auch den erwarteten Query String definieren und alle Anfragen mit falschen Query String automatisch ablehnen

Backend Development

Fazit Meilenstein 05

- Mit dem Einsatz von SQLite können wir Daten konsistent auf dem Dateisystem speichern.
- Auch die zuvor verletzte REST Guideline Uniforme Standardisierte Schnittstelle ist jetzt gelöst und die Ressource hat eine feste, eindeutige Adresse
 - Vgl. /beer mit HTTP Methoden vs. /beer/createBeer
- Nur Maturity Level 3 erreichen wir auch weiterhin nicht (Hypermedia as the Engine of Application State)

Backend Development

Fazit

- Und mit Meilenstein 05 haben wir jetzt eine vollwertige API Schnittstelle für unsere Beer App
- Biere können über klar definierte HTTP Anfragen ausgetauscht und angelegt werden
- Was uns jetzt noch Fehlt ist eine Webseite, über die User unsere Beer App nutzen können

Frontend Development

Webseiten mit Clientseitigen JavaScript entwickeln

Frontend Development

Meilenstein 06

- Wir möchten unsere Beer App jetzt als Webseite auf den Markt bringen. Zukünftige User können über einen Webbrowser die Seite besuchen und mit unserer App interagieren. Alle Funktionalitäten sollen auch über die Webseite funktionieren und es soll eine abgerundete User Experience bereitgestellt werden
- In Meilenstein 06 möchten wir uns jetzt eine Webseite bauen die mit unserer Beer API kommuniziert und alle Informationen als HTML darstellt. Die Webseite selbst möchten wir dabei mit einem JavaScript Framework entwickeln und implementieren

Frontend Development

Hintergrund

- Moderne Webseiten werden immer komplexer in ihren Anwendungsfällen
 - Migration von Desktop Anwendungen zu Web Anwendungen
- Webseiten können für verschiedene Endgeräte entwickelt werden mit unterschiedlichen Herausforderungen
 - Offline Funktionalität
 - Natives Erlebnis
- Immer mehr Dynamische Inhalte werden auf Webseiten benötigt was bei klassischen HTML zu einem Neu laden der Seite führt
 - JavaScript kann „live“ HTML Elemente modifizieren und User Interaktionen abbilden

Frontend Development

Hintergrund

- Webseiten mit JavaScript zu programmieren ist in Bezug auf den Implementierungsaufwand aber sehr aufwendig und langsam
- Webseiten speziell HTML werden im Hintergrund über ein Document Object Model (DOM) abgebildet
- Um Webseiten zu entwickeln, gibt es deshalb eine spezielle Programmierschnittstelle, um mit dem DOM zu arbeiten
 - Auf diese Schnittstelle greift auch JavaScript im Browser zu

Frontend Development

Hintergrund

- Der DOM ist aber nicht Teil von JavaScript sondern eine spezielle Web API um Webseiten zu bauen
 - Diese Schnittstelle gibt es auch nicht in NodeJS
- Aus diesem Grund können auch andere Programmiersprachen mit dem DOM interagieren und dynamisch Webseiten erzeugen
 - Bspw. Python

Frontend Development

Document Object Model – Beispiel



```
1 <body>
2   <h1>DOM Einführung</h1>
3
4   <label for="newText">Text hinzufügen:</label>
5   <input type="text" id="newText" placeholder="Gib einen Text ein" />
6   <button id="addElement">Neues Element hinzufügen</button>
7
8   <div id="newElements"></div>
9
10 <script>
11   const introParagraph = document.getElementById("intro");
12   const addElementButton = document.getElementById("addElement");
13   const newTextInput = document.getElementById("newText");
14   const newElementsDiv = document.getElementById("newElements");
15
16   addElementButton.addEventListener("click", () => {
17     const userInput = newTextInput.value.trim();
18     if (userInput) {
19       const newParagraph = document.createElement("p");
20       newParagraph.textContent = userInput;
21       newElementsDiv.appendChild(newParagraph);
22       newTextInput.value = ""; // Leere das Eingabefeld
23     }
24   });
25 </script>
26 </body>
```

Frontend Development

Document Object Model – Beispiel

- Mit JavaScript und dem DOM können wir also eine HTML Seite Dynamisch modifizieren und bspw. neue HTML Elemente erstellen
- Ohne den DOM wüsste JavaScript nicht wie eine Webseite aussieht
- Alle Elemente in einem Dokument sind auch Teil vom DOM
 - Und diese Elemente können mittels DOM und JavaScript manipuliert werden

Frontend Development

Document Object Model – Beispiel



```
1 <body>
2   <h1>DOM Manipulation</h1>
3
4   <div class="container">
5     <textarea class="story"></textarea>
6     <button id="set-text" type="button">Set text content</button>
7     <button id="clear-text" type="button">Clear text content</button>
8   </div>
9
10 <script>
11   const story = document.body.querySelector(".story");
12
13   const setText = document.body.querySelector("#set-text");
14   setText.addEventListener("click", () => {
15     story.textContent = "It was a dark and stormy night...";
16   });
17
18   const clearText = document.body.querySelector("#clear-text");
19   clearText.addEventListener("click", () => {
20     story.textContent = "";
21   });
22 </script>
23 </body>
```

Frontend Development

DOM in der Praxis

- Fällt Ihnen etwas auf in den vorherigen Beispielen in Bezug auf das Programmieren gegen die DOM Schnittstelle?
- Angenommen wir möchten eine Liste an Bieren auf einer Webseite anzeigen

```
1  {
2    name: "Nattheimer Spezial",
3    taste: "Good",
4    brand: "Nattheimer",
5    amount: 6,
6    rating: 4,
7    id: 1,
8  },
```

Frontend Development

DOM in der Praxis



```
1  renderBeerList();
2
3  function buildBeerCard(props) {
4    const beerListElement = document.createElement("div");
5    beerListElement.classList.add("beer-list-element");
6
7    const beerName = createBeerProperty("h2", `Name: ${props.name}`);
8    beerListElement.appendChild(beerName);
9
10   const beerTaste = createBeerProperty("p", `Taste: ${props.taste}`);
11   beerListElement.appendChild(beerTaste);
12
13   const beerBrand = createBeerProperty("p", `Brand: ${props.brand}`);
14   beerListElement.appendChild(beerBrand);
15
16   const beerAmount = createBeerProperty("p", `Amount: ${props.amount}`);
17   beerListElement.appendChild(beerAmount);
18
19   const beerRating = createBeerProperty("p", `Rating: ${props.rating}`);
20   beerListElement.appendChild(beerRating);
21
22   return beerListElement;
23 }
24
25 function createBeerProperty(element, value) {
26   const beerProperty = document.createElement(element);
27   beerProperty.textContent = value;
28   return beerProperty;
29 }
30
31 function renderBeerList() {
32   const beerList = document.createElement("ul");
33   beerList.classList.add("beer-list");
34
35   beers.forEach((beer) => {
36     const beerListElement = document.createElement("li");
37     const beerCardElement = buildBeerCard(beer);
38
39     beerListElement.appendChild(beerCardElement);
40     beerList.appendChild(beerListElement);
41   });
42
43   document.body.appendChild(beerList);
44 }
```

Frontend Development

DOM in der Praxis

- Nur für das Rendern der einzelnen Elemente benötigen wir alleine 44 Zeilen JavaScript Code
 - Selbst mit Optimierungen werden es nicht viel weniger
- Aber wir haben noch keine Business Logik eingebaut
- Wir haben auch noch keine Styles oder ähnliches – Der Code dient nur zum Erstellen von einfachen HTML Elementen

Frontend Development

DOM in der Praxis

- Arbeiten mit dem DOM benötigen unglaublich viel Wissen über die einzelnen DOM Schnittstellen
 - Wie erstellen wir Elemente, wie können wir Attribute modifizieren, wie können wir Element ineinander verschachteln etc.
- Kurz gesagt: Arbeiten mit dem DOM ist sehr verbose und die Schnittstelle bietet keine gute Developer Experience bzw. Abstraktion der unterliegenden Funktionalitäten

Frontend Development

Frameworks für Client Seitige Applikationen

- Aus diesem Grund wurden sehr früh erste JavaScript Frameworks* entwickelt, die genau diese Komplexität abstrahieren
- Anstatt direkt mit dem DOM zu arbeiten, schreiben wir unsere Programme mit einem JavaScript Framework* und NodeJS
- Am Ende kommt eine fertige HTML Seite heraus die das komplette JavaScript beinhaltet, um unsere Webseite darzustellen
- Frameworks* inkludiert hier Libraries, Tools, Scripte etc. - Nicht jedes Package auf NPM ist direkt ein Framework

Frontend Development

Frameworks für Client Seitige Applikationen



- Das Anzeigen von Bieren würde bspw. mit dem Angular Framework ungefähr so aussehen:
- **Wir beschreiben User Interfaces auf Deklarative Art**

```
1  @Component({
2    selector: 'app-beer-list',
3    template: `
4      <h1>Beer List</h1>
5      <ul class="beer-list">
6        @for (let beer of beers) {
7          <li class="beer-list-element">
8            <div>
9              <h2>{{ beer.name }}</h2>
10             <p>Taste: {{ beer.taste }}</p>
11             <p>Brand: {{ beer.brand }}</p>
12             <p>Amount: {{ beer.amount }}</p>
13             <p>Rating: {{ beer.rating }}</p>
14           </div>
15         </li>
16       }
17     </ul>
18   `,
19   standalone: true
20 })
```

Frontend Development

Web Applikationen



- (Moderne) Web Applikationen unterscheiden sich heute sehr stark von den ursprünglichen Ansätzen von Webseiten
- Fallen Ihnen spontan Bausteine ein, die zu Web Applikationen gehören?

Frontend Development

Bausteine einer Web Applikation

- **User Interface** – Wie Benutzer mit einer Anwendung interagieren
- **Routing** – Wie Benutzer zwischen einzelnen Bereichen navigieren
- **Data Fetching** – Wo Daten „leben“ und wie man diese bekommt
- **Rendering** – Wann und Wo Statischer Content, Dynamischer Content angezeigt wird
- **Integrations** – Third Party Services (CMS, Auth, Payments)

Frontend Development

Bausteine einer Web Applikation

- **Infrastructure** – Deployment, Storage, Ausführung Applikation (CDN, Edge etc.)
- **Performance** – Optimierung für End-Benutzer (Anwendungsfälle)
- **Scalability** – Wie skaliert die Anwendung mit mehr Traffic / Daten / Usern
- **Developer Experience** – Wie schnell können Anwendungen gebaut werden und können diese erweitert werden (Maintenance*)

Frontend Development

Wahl der Bausteine

- Für all diese Bausteine muss man sich die folgenden Fragen stellen:
- Bauen wir die Softwarelösung selbst oder nutzen wir etwas, was bereits auf dem Markt existiert?
- Speziell in der Web Programmierung gibt es eine riesige Anzahl an fertigen Tools / Libraries / Frameworks die uns die Umsetzung leichter gestalten

Frontend Development

Design Patterns

- Die Umsetzung von Web Applikationen hat sich in den letzten 25 Jahren stark verändert
 - Von reinen Webseiten hin zu Web Anwendungen mit neuen Herausforderungen
- Für die Umsetzung haben sich unterschiedliche Architektur Patterns bzw. Design Patterns etabliert
- Heute (2024) ist das wohl bekannteste Design Pattern Single Page Apps (SPAs)

Frontend Development

Single Page Applications

- Single Page Applications (SPAs) sind eine besondere Art von Webseiten
- Ein Webserver liefert nur eine einzige HTML Datei aus die ein Element als „Einstiegspunkt“ definiert und alles weitere wird über JavaScript dynamisch erstellt
- Das bedeutet aber auch, dass alles auf dem Client Computer direkt läuft
 - Und man u.a. Zugang zum Source Code hat

Frontend Development

Single Page Applications

- Eine React Anwendung kann bspw. über folgendes HTML Dokument von einem Webserver ausgeliefert werden

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Vite + React</title>
8      <script type="module" crossorigin src="/assets/index-a7e7d0e4.js"></script>
9      <link rel="stylesheet" href="/assets/index-00bf253d.css" />
10    </head>
11    <body>
12      <div id="root"></div>
13    </body>
14  </html>
15
```

Frontend Development

Single Page Applications

- Webseiten können heute viel mehr Funktionen abbilden, die früher bspw. nur in Desktop Applikationen möglich waren
- Viele Desktop Applikationen gibt es mittlerweile als Web Applikation
 - Office 365, Google Docs, Teams, SAP
- SPAs eignen sich besonders für dynamische Inhalte und aktive User Interaktionen

Frontend Development

Alternativen

- Content Management Systeme
 - Inhalte anlegen ohne direkt Code zu entwickeln. Speziell geeignet für Content Writers (e.g. Zeitungen etc.)
 - Bspw: Wordpress, Joomla und Drupal
- Server Side Rendering (SSR)
 - Anwendungen werden auf Server Seite gerendert und der Client bekommt nur die fertigen HTML files
 - Für alle in der Vorlesung betrachteten Frameworks gibt es auch SSR Optionen

Frontend Development

Alternativen

- Static Site Generator
 - Basierend auf einem Template werden statische HTML-Seiten erstellt, welche von einem Webserver ausgeliefert werden können
 - Static Site Generators können beide Welten kombinieren (Sie können auf einer einzelnen Seite z.B. Clientseitiges Javascript ausführen (bspw. React))

Frontend Development

Design Patterns

- Der Trend aktuell geht aber wieder mehr in die Richtung Server Side Rendering und Business Logik Server seitig auszuführen
- Die Hintergründe sind zum einen, dass Web Applikationen immer mehr Business Logik benötigen und Server Rechenleistung deutlich günstiger ist, als Client Rechen Leistung
- Gleichzeitig können mit solchen Frameworks Webseiten die als CMS, Static Site etc. fungieren viele einfacher abgebildet werden

Frontend Development

Design Patterns

- Was für ein Design Pattern Sie am Ende einsetzen ist am Ende abhängig was für ein Problem Sie lösen möchten
- Die vorgestellten Konzepte in der Vorlesung sind aber auf SPAs und SSR Konzepte universell anwendbar
- Gerade im Enterprise Umfeld und der Transformation zu Cloud Anwendungen werden sie noch sehr viele SPAs finden während größere Content getriebene Seiten eher zu SSR tendieren

Frontend Development

Eingrenzung

- In der Vorlesung beschäftigen wir uns primär mit den Konzepten hinter Single Page Applications
- Als Frameworks* nutzen wir React / Angular
- Speziell für React werden wir aber ein weiteres Framework nutzen (Gleich mehr dazu), was viele Konzepte von SSR / MPA nutzt bzw. parallel anbietet

Frontend Development

Eingrenzung

- React – Eine JavaScript Bibliothek für das Erstellen von User Interfaces
 - Benötigt weitere Packages für vollständige Apps
 - Entwickelt von Meta (Facebook) und als Open Source Technologie verfügbar
- Angular – Ein vollwertiges Framework, um Anwendungen zu programmieren
 - Verwendet TypeScript* anstelle von JavaScript
 - Wird von Google und einer Reihe Freier Maintainer entwickelt und ist auch Open Source

Frontend Development

Besonderheiten React vs. Angular

- React ist eine reine Bibliothek mit einer speziellen Markup Syntax: JSX
 - Themen wie State Management, HTTP Kommunikation müssen Sie entweder separat installieren oder selbst implementieren
- Angular ist ein Framework und kommt mit einer Auswahl an fertigen Funktionen und man programmiert mit TypeScript*
 - Routing, State Management, eigener HTTP Client, CLI Tools etc. ist alles inkludiert

Frontend Development

Eingrenzung

- Aus diesem Grund werden wir für die Entwicklung mit React auch auf ein Framework (NextJS) zurückgreifen

Frontend Development

Hinweise

- Aufgrund der Komplexität der einzelnen Frameworks und ihren Funktionalitäten begrenzen wir uns in der Vorlesung auf die Grundlegenden Konzepte wie man moderne Web Applikationen programmiert
- Tiefere Verständnis für ein Framework kann im Selbststudium erarbeitet werden

Frontend Development

Imperative vs. Deklarative Programmierung

- Wenn wir uns nochmal das DOM Beispiel anschauen, um User Interfaces zu programmieren, haben wir einen imperativen Programmierstil verwendet.
- Imperativer Programmierstil:
 - Wie wird es gemacht?
 - Bedeutet: **Wie** ein Problem gelöst wird, wird detailliert beschrieben
- Deklarativer Programmierstil
 - Was soll erreicht werden?
 - Bedeutet: **Was** soll erreicht werden, ohne die Details der Implementierung zu spezifizieren

Frontend Development

Imperative vs. Deklarative Programmierung

- Beispiel Imperativer Stil:

```
1 for (let i = 1; i <= 5; i++) {  
2   console.log(i);  
3 }
```

- Hier wird explizit angegeben, wie die Schleife funktioniert und welche Schritte ausgeführt werden

```
1 [1, 2, 3, 4, 5].forEach((num) => console.log(num));  
2
```

Frontend Development

Imperative vs. Deklarative Programmierung

- Beispiel Deklarativer Stil:

```
1 [1, 2, 3, 4, 5].forEach((num) => console.log(num));
```

- Hier wird angegeben, wie jede Zahl ausgegeben werden soll, aber es wird nicht erklärt, wie die Iteration intern funktioniert

Frontend Development

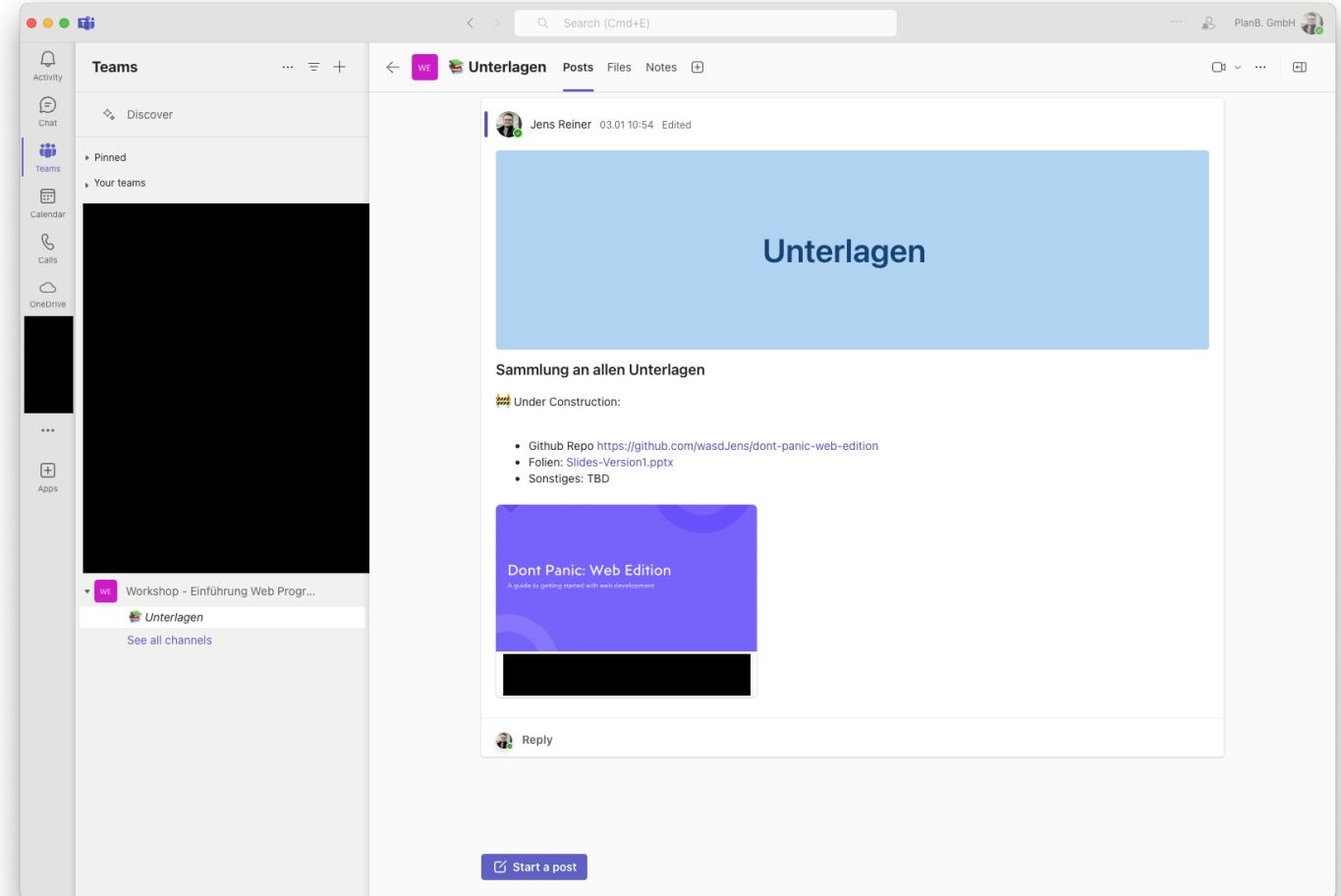
User Interfaces

- Für unseren Baustein User Interfaces einer Web Applikationen verwenden wir speziell den Deklarativen Programmierstil Ansatz
- Wir beschreiben als Entwickler, **was** passieren soll innerhalb der UI und ein Framework übernimmt die Aufgabe, **wie** der DOM manipuliert werden muss für das Gewünschte Ergebnis

Frontend Development

User Interface – Fallbeispiel

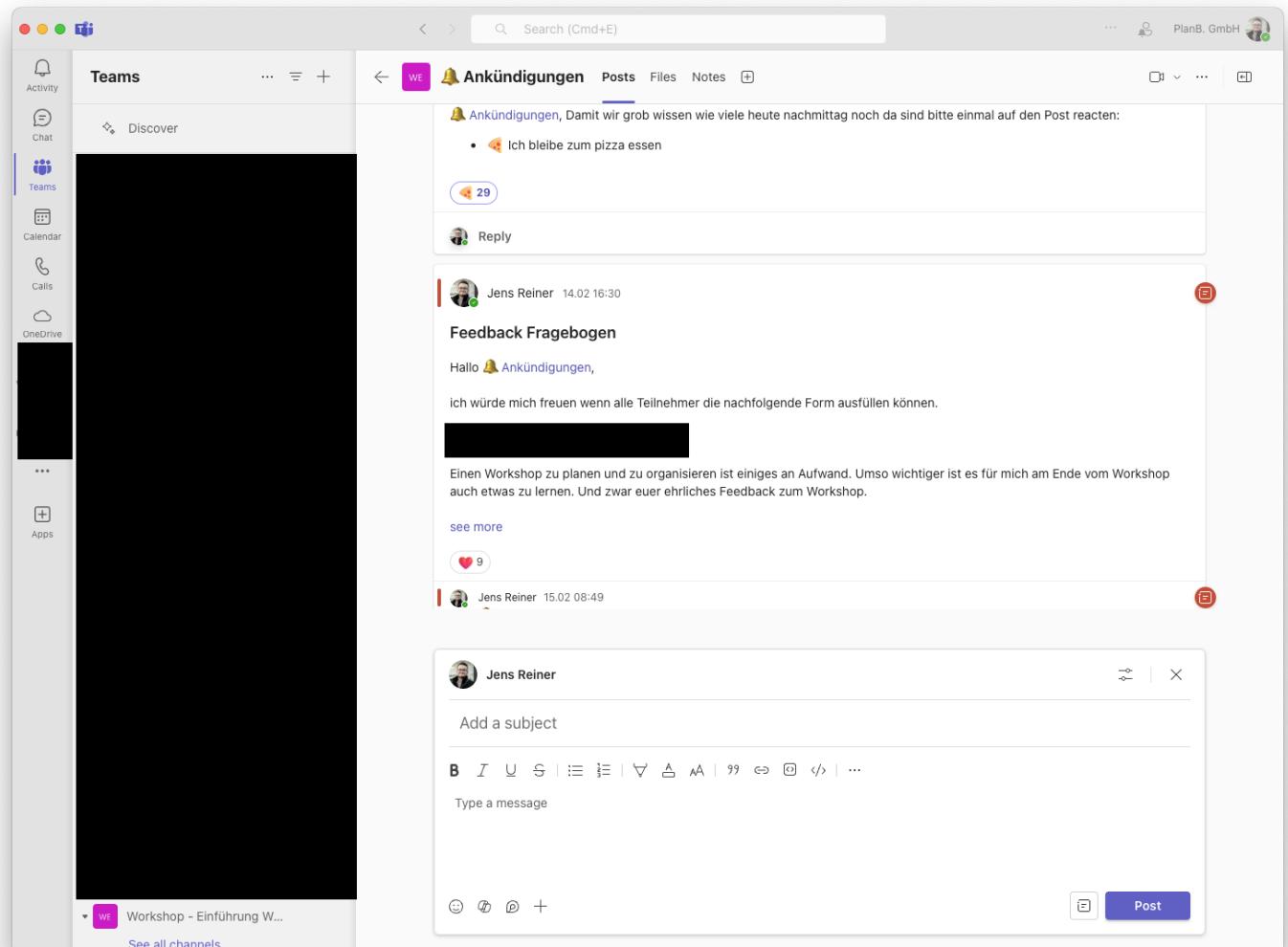
- Angenommen Sie müssten die Teams Oberfläche als User Interface realisieren
- Wie würden Sie vorgehen?



Frontend Development

User Interface – Fallbeispiel

- Teams ist eher eine Anwendung im Vergleich zu einer klassischen Webseite
- Teams hat sehr viele Elemente, die eine User Interaktion anbieten
- Viele UI Elemente werden wiederverwendet



Frontend Development

User Interface – Fallbeispiel

- Ein Problem in der Umsetzung von User Interfaces, ist der Ansatz, die gesamte Oberfläche in einem Stück zu entwickeln
- In den meisten Fällen bekommt man einen Entwurf für die vollständige Anwendung und versucht diesen darzustellen
- Wenn man jetzt ein neues Feature hinzufügt, ist der Aufwand deutlich höher, weil man in den meisten Fällen die Oberfläche anpassen muss

Frontend Development

Component Driven Development

- User Interfaces sollten deshalb mit einem **Bottom Up Konzept** programmiert werden
- Zuerst identifiziert man sich die Elemente, die immer wieder vorkommen
- Diese Elemente deklarieren wir als Eigenständige Komponenten, die man wiederverwenden kann
- Aus diesen kleinen Bausteinen definieren wir uns dann neue Bausteine, die größere Bereiche abbilden (Bottom Up)

Frontend Development

Component Driven Development

- Anstatt ein User Interface als Ganzes zu deklarieren, nutzen wir ganz viele kleinere Components die einander “gesteckt“ werden, um weitere Components zu definieren
- Am Ende nutzen wir diese Components, um die gesamte Seite darzustellen
- Diese Components können dann auch beliebig getauscht werden

Frontend Development

Component Driven Development

- Was eine Component ist, kann man anhand der folgenden Drei Konzepten festhalten
- Eine Component sollte genau eine Aufgabe übernehmen
- Eine Component hat einen internen State
- Eine Component kann über Props / Inputs von außen beeinflusst werden

Frontend Development

Component Driven Development

- Aus der Sicht von Programmiersprachen ist eine Component also nichts anderes als eine Funktion
- Eine Funktion löst genau eine Aufgabe
- Eine Funktion kann sich intern Variablen und Werte zwischen speichern
- Eine Funktion kann Parameter entgegennehmen die den Ablauf der Funktion beeinflussen

Frontend Development

Component Driven Development

- Angenommen wir möchten eine Icon Component programmieren
- Ein Icon besteht dabei aus einem Text und einem SVG Icon
- Daraus lässt sich (Pseudocode) folgende Funktionen definieren:

Frontend Development

Component Driven Development

```
1 function SvgIcon(props: { icon: string, color: string, size: number }) {
2   return `<svg fill="${props.color}" width="${props.size}" height="${props.size}" viewBox="0 0 24 24">
3     <path d="${props.icon}"></path>
4   </svg>`;
5 }
6
7 function IconText(props: { text: string, color: string }) {
8   return `<span style="color: ${props.color}">${props.text}</span>`;
9 }
10
11 function Icon(props: { icon: string, text: string, active: boolean }) {
12   return `<div style="background-color: ${active ? 'blue' : 'white'}">
13   ${SvgIcon({ icon: props.icon, color: 'black', size: 24 })}
14   ${IconText({ text: props.text, color: 'black' })}
15 </div>`;
16 }
```

Frontend Development

Definition Component

- Components sind standardisierte, wiederverwendbare Bausteine, die ein Element von einer Oberfläche abbilden.
- Die Komponente beinhaltet dabei die Darstellung und Logik des Elements
- Komponenten können in anderen Komponenten zusammengefasst werden, um eine vollständige Oberfläche zu erstellen

Frontend Development

Component Design

- Woher weiß man jetzt, was eine „gute“ Component ist?
- Wie identifiziert man Components in einem Design Entwurf?

Frontend Development

Component Design – Kriterien

- Wie viel Komplexität sollte die Component beinhalten?
- Welche „API“ nach außen benötigt meine Component, damit andere diese wiederverwenden können?
- Gibt es Components dir nur anzeigen und welche die Daten verarbeiten?
- Welchen State benötigt meine Component intern?
- Hat meine Component Abhängigkeiten zu anderen und wie sehen diese aus?

Frontend Development

Component Design – Kriterien

- Die Komplexität sollte man immer mit dem Single Responsibility Principle entscheiden
 - Vgl. Bottom Up Konzept
- Eine API nach außen entscheidet sich nach dem Use Case der Component
 - Ein Button wird Verhältnis mäßig viele Optionen nach außen anbieten
 - Eine Layout Component wenig bis gar keine Optionen nach außen
- Welche Aufgabe übernimmt die Component?
 - Reine Anzeige?
 - Business Logik?

Frontend Development

Component Design – Smart and Dumb Component

- Es bietet sich an Components in zwei Kriterien zu unterteilen:
 - Smart Component
 - Dumb Component
- Eine Smart Component ist zuständig für die Logik und Datenverarbeitung. Hat aber keine UI-Implementierung sondern „ruft“ andere Components auf
- Eine Dumb Component hat neben ihrem eigenem State keine Logik und nimmt Daten einfach entgegen, diese werden visuell dargestellt

Frontend Development

Component Driven Principles

1. Beginne mit den kleinsten wiederkehrenden Elementen
2. Fasse viele kleine Komponenten zu größeren Komponenten zusammen
3. Baue Seiten aus Komponenten
4. Baue die komplette Oberfläche indem Komponenten integriert werden

Frontend Development

Component Driven Development

- Eine ausführliche Erklärung zu diesem Thema finden Sie auch hier
<https://github.com/wasdJens/dont-panic-web-edition/blob/main/explanation/component-driven-development/component-driven-user-interfaces.md>

Frontend Development

Umsetzung

- Die gesamte Theorie hinter deklarativen User Interfaces und Component basierten Frameworks möchten wir uns jetzt einmal in der Praxis anschauen
- Als nächsten setzen wir die Client Anwendung für unsere Beer App mit Angular und NextJS um

Frontend Development

Web Applikationen mit Angular und NextJS

Frontend Development

Hinweise

- Frameworks wirken auf den ersten Moment sehr komplex
- Frameworks setzen sehr viele Konzepte, Features etc. direkt um, die wir uns in der Vorlesung noch nicht angeschaut haben
- Aber die wichtigsten Konzepte sollten Sie mit ihrem JavaScript Wissen verstehen.
- Gleichzeitig schauen wir uns auch die wichtigste „Magie“ im Detail genauer an

Frontend Development

Hinweise

- Aufgrund der Komplexität von Angular und NextJS beschäftigen wir uns in der Vorlesung sehr oberflächlich mit beiden Frameworks
- Für einen speziellen Deep Dive eignet sich die offizielle Dokumentation oder fragen sie explizit nach, falls etwas unklar ist

Frontend Development

Hinweise

- NextJS hat auch am 21.10.2024 im Rahmen der Next Conf eine neue Major Version bekommen und einige Funktionalitäten funktionieren noch nicht reibungslos mit beliebten Third Party Paketen



Frontend Development

NextJS – Installation

- NextJS hat ein NPM Utility für die Installation und dem Setup von einem Projekt:
 - Command **npx create-next-app@latest**

```
~/Code/DHBW/dhbw-webprogrammierung-wwi/04-frontend/01-nextjs on ✘ master 15:45:30
$ npx create-next-app@latest
Need to install the following packages:
create-next-app@15.0.1
Ok to proceed? (y) y
```

- Bestätigung der Installation mit y

Frontend Development

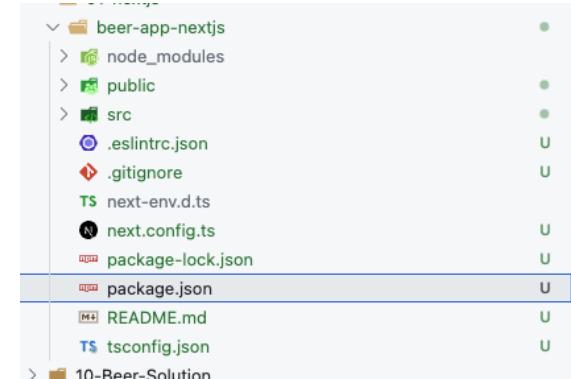
NextJS – Installation

- Sie können Folgende Einstellungen übernehmen:

```
OK to proceed? (y) y
✓ What is your project named? ... beer-app-nextjs
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like your code inside a 'src/' directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to use Turbopack for next dev? ... No / Yes
✓ Would you like to customize the import alias (@/* by default)? ... No / Yes
✓ What import alias would you like configured? ... @/*
Creating a new Next.js app in /Users/jens/Code/DHBW/dhbw-webprogrammierung-wwi/04-frontend/01-nextjs/beer-app-nextjs.

Hilfe: npm
```

- Anschließend sollte ein ähnlicher Ordner erstellt worden sein:



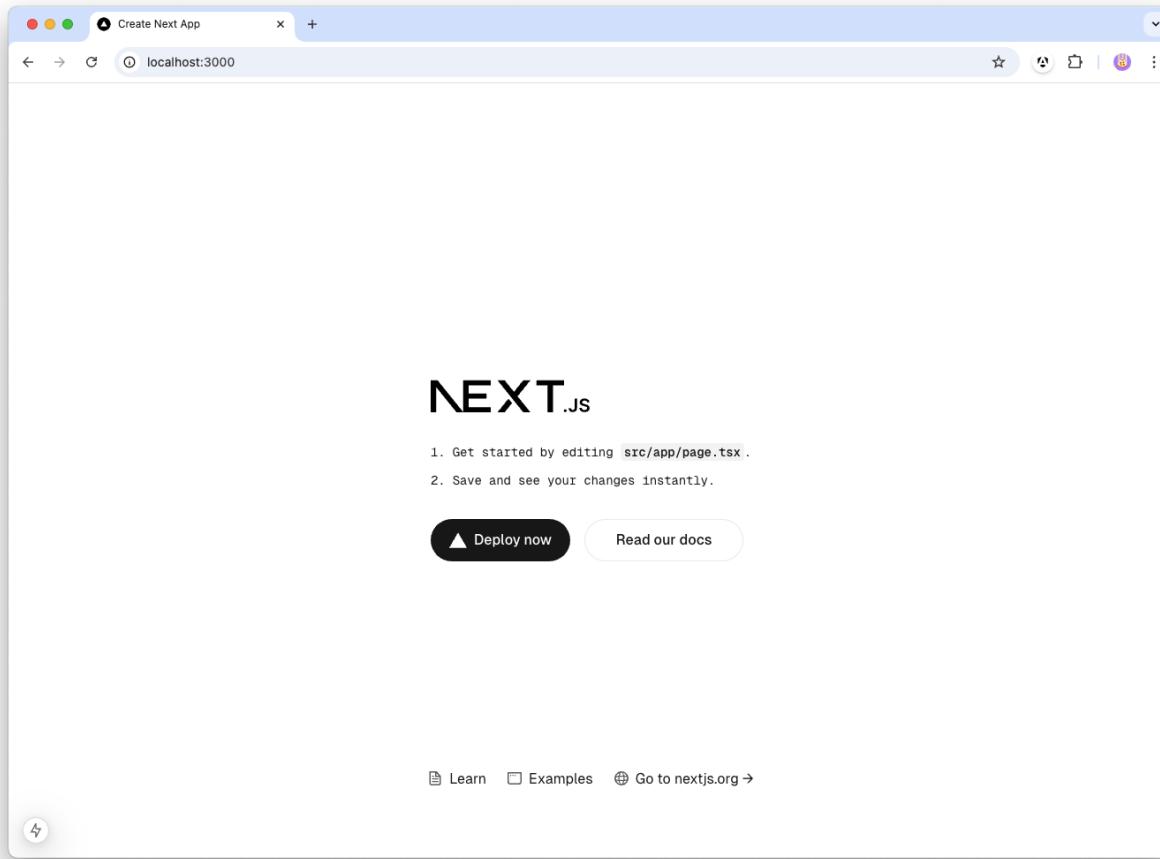
Frontend Development

NextJS – Installation

- Nach der Installation können Sie mit **npm run dev** den Development Server von NextJS starten
- Anschließend öffnen Sie im Browser [**http://localhost:3000**](http://localhost:3000) und sollten eine NextJS Willkommens Seite bekommen
- Hinweis: Wenn Sie Änderungen vornehmen, müssen Sie den Server nicht neu starten. NextJS aktualisiert sich selbstständig im Hintergrund

Frontend Development

NextJS – Installation



Frontend Development

NextJS – Ordner Struktur

- `/app/` - Beinhaltet alle Routen, Components und die Logik der Applikation
 - In diesem Ordner werden wir primär arbeiten und programmieren
- `/app/lib` – Beinhaltet Funktionen, welche über die Anwendung hinweg wiederverwendet werden können
- `/app/ui` – Beinhaltet alle UI Components (Base Components)
- `/public` – Beinhaltet Static Assets (u.a. Bilder etc.)
- Den Ordner `/lib` und `/ui` müssen Sie ggf. erst selbst anlegen

Frontend Development

Hinweis Typescript

- Alle Dateien haben nicht die Endung .js sondern entweder .ts oder .tsx
- In Angular und React nutzen wir Typescript für die Implementierung.
- Genaueres zum Thema Typescript schauen wir uns erst später an
- Für den Moment, können Sie wie gewohnt JavaScript programmieren

Frontend Development

NextJS – Grundlagen

- Im Ordner /app befindet sich eine **globals.css** Datei. In dieser Datei können Sie globale CSS-Klassen definieren
- Wenn Sie diese Klassen nutzen möchten, können Sie die Globale CSS Definition einfach in einer **.tsx** Datei importieren
- Bspw. finden Sie in der **layout.tsx** einen Import auf die Globale CSS Datei
- In React übergeben Sie CSS Klassen über das **className** Attribute

Frontend Development

NextJS – Grundlagen

```
git: master / 0 files / 0 dirs / 000 app / 000 app / 000 layouts / 000
  1 import type { Metadata } from "next";
  2 import localFont from "next/font/local";
  3 import "./globals.css";
  4
  5 > const geistSans = localFont({
  6   weight: "normal",
  7   style: "normal",
  8   variable: "sans-serif",
  9 });
10 > const geistMono = localFont({
11   weight: "normal",
12   style: "normal",
13   variable: "monospace",
14 });
15
16 > export const metadata: Metadata = {
17   title: "Next.js Layout Example"
18 };
19
20
21  export default function RootLayout({
22   children,
23 }: Readonly<{
24   children: React.ReactNode;
25 }>) {
26   return (
27     <html lang="en">
28       <body className={`${geistSans.variable} ${geistMono.variable}`}>
29         {children}
30       </body>
31     </html>
32   );
33 }
34 }
```

Frontend Development

NextJS – Grundlagen

- Tipp – Falls Sie in React Klassen über eine Bedingung hinzufügen möchten, können Sie sich das Package clsx installieren
 - <https://github.com/lukeed/clsx>

Frontend Development

Angular – Installation

- Angular bietet eine CLI für die Installation und Verwaltung von Angular Anwendungen an
- Die CLI kann global über NPM installiert werden **npm install -g @angular/cli**
 - Hinweis: Unter Windows sind Globale Packages deaktiviert hier finden Sie die Option, die execution Policy zu ändern <https://angular.dev/tools/cli/setup-local#powershell-execution-policy>
- Anschließend können Sie mit dem Befehl **ng new <project-name>** ein neues Angular Projekt anlegen

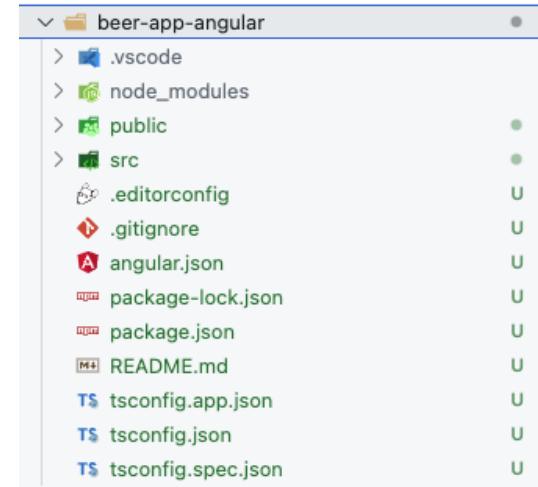
Frontend Development

Angular – Installation

- Sie können hier folgende Einstellungen nutzen

```
~/Code/DHBW/dhbw-webprogrammierung-wi/04-frontend/02-angular on y master! 16:32:34
$ ng new beer-app-angular
? Which stylesheet format would you like to use? CSS [https://developer.mozilla.org/docs/Web/CSS]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)?
no
CREATE beer-app-angular/.README.md (1076 bytes)
```

- Anschließend sollte folgende Ordner Struktur angelegt worden sein



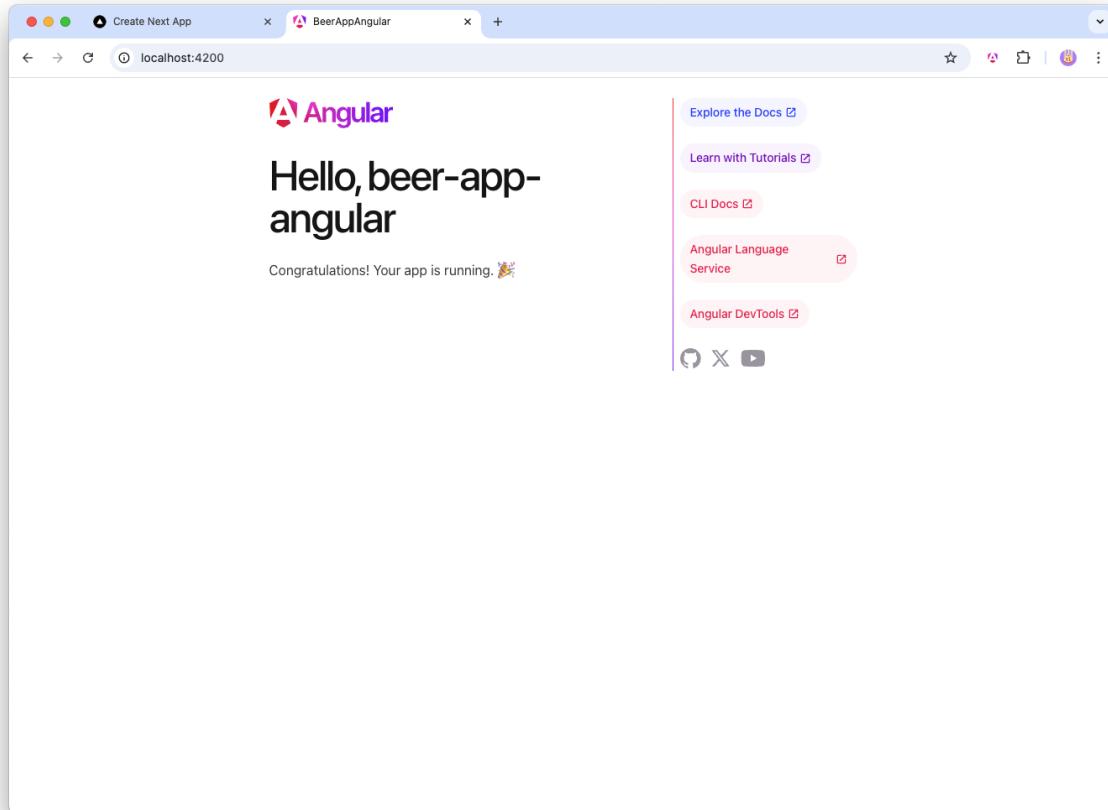
Frontend Development

Angular – Installation

- Nach der Installation können Sie mit **npm run start** den Development Server von Angular starten
- Anschließend öffnen Sie im Browser [**http://localhost:4200**](http://localhost:4200) und sollten eine Angular Willkommens Seite bekommen
- Hinweis: Wenn Sie Änderungen vornehmen, müssen Sie den Server nicht neu starten. Auch Angular aktualisiert die Webseite automatisch

Frontend Development

Angular – Installation



Frontend Development

Angular – Ordner Struktur

- /app/ - Beinhaltet alle Routen, Components und die Logik der Applikation
 - In diesem Ordner werden wir primär arbeiten und programmieren
- /app/shared – Beinhaltet Funktionen, welche über die Anwendung hinweg wiederverwendet werden können
- /app/ui – Beinhaltet alle UI Components (Base Components)
- /public – Beinhaltet Static Assets
- Den Ordner /shared und /ui müssen Sie ggf. erst selbst anlegen

Frontend Development

Hinweis Typescript / Klassen

- Auch Angular wird mit Typescript programmiert aus diesem Grund sind auch hier alle JavaScript Dateien mit **.ts** als Dateiendung
- Wenn Sie eine Component öffnen werden Sie sehen, dass Angular Klassen nutzt – Auch dieses Konzept schauen wir uns später genauer an

Frontend Development

Angular – Grundlagen

- Im Ordner /src befindet sich eine **styles.css** Datei. In dieser Datei können Sie globale CSS Klassen definieren
- In Angular müssen Sie die globale CSS Datei nicht importieren, wenn Sie eine Globale Klasse nutzen möchten können Sie diese einfach über das **class** Attribute angeben

Frontend Development

Components Implementieren

- Wir möchten jetzt einmal die Grundlagen von Components in den jeweiligen Frameworks genauer betrachten
- Wir schauen uns dafür vereinzelte Beispiele an, um die wichtigsten Patterns darzustellen
- Die Folien werden jetzt immer ein Side By Side Layout haben

Frontend Development

Components – Grundlagen

React / NextJS

- In React werden Components in einer speziellen Syntax geschrieben: JSX
- JSX kombiniert JavaScript und HTML
- Eine Component in React ist einfach eine JS Funktion, die ein JSX String zurück gibt

Angular

- In Angular werden Components in einer Klasse definiert
- HTML und JavaScript sind in getrennten Dateien
- Eine Component in Angular ist eine Decorator Funktion, die als JavaScript Klasse definiert wird

Frontend Development

Components – Grundlagen

React / NextJS

```
1 "use client";
2
3 import styles from "./beer-footer.module.css";
4
5 export default function BeerFooter() {
6   return (
7     <div className={styles.footer}>
8       <div>
9         <p>
10          Diese Webseite dient nur für Unterhaltungszwecke und wurde im Rahmen
11          der Web Programmierung Vorlesung 2024 entwickelt
12        </p>
13        <p>Made with ❤ by Jens Reiner</p>
14      </div>
15      <div className={styles.links}>
16        <a
17          href="https://www.bundesdrogenbeauftragter.de/service/beratungsangebote/"
18          target="_blank"
19        >
20          ⚡ Beratungsangebote
21        </a>
22        <a
23          href="https://www.dhs.de/service/suchthilfeverzeichnis"
24          target="_blank"
25        >
26          ⚡ Deutsche Hauptstelle für Suchtfragen
27        </a>
28      </div>
29    </div>
30  );
31}
```

Angular

```
1 <div class="footer">
2   <div>
3     <p>
4       Diese Webseite dient nur für Unterhaltungszwecke und wurde im Rahmen
5       der Web Programmierung Vorlesung 2024 entwickelt
6     </p>
7     <p>Made with ❤ by Jens Reiner</p>
8   </div>
9   <div class="links">
10     <a
11       href="https://www.bundesdrogenbeauftragter.de/service/beratungsangebote/"
12       target="_blank"
13     >
14       ⚡ Beratungsangebote
15     </a>
16     <a
17       href="https://www.dhs.de/service/suchthilfeverzeichnis"
18       target="_blank"
19     >
20       ⚡ Deutsche Hauptstelle für Suchtfragen
21     </a>
22   </div>
23 </div>
24
```

Frontend Development

Components – Grundlagen

React / NextJS

- Der Name der Funktion für eine Component muss in React mit einem Großbuchstaben beginnen
- Eine React Component hat immer einen Return Wert, den JSX String
- Innerhalb der Funktion können ganz normal Variablen etc. definiert werden

Angular

- Der @Decorator beschreibt die Component in Angular und bindet Style und HTML Dateien ein
- Innerhalb der Klasse kann alles an JavaScript definiert werden

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-beer-footer',
5   standalone: true,
6   imports: [],
7   templateUrl: './beer-footer.component.html',
8   styleUrls: ['./beer-footer.component.css']
9 })
10 export class BeerFooterComponent {}
```

Frontend Development

Components – Besonderheiten

React / NextJS

- CSS Klassen müssen über ein spezielles **className** Attribute gesetzt werden
- CSS Dateien werden als CSS Module in der tsx Datei importiert
 - Scoped auf die Component
- **use client** gibt NextJS die Information, dass diese Component beim Client gerendert wird (Später mehr)

Angular

- CSS Klassen können einfach in der Component CSS Datei definiert werden
- Alle Dateien und Selektoren können automatisch in Angular erstellt werden
- Mit dem Befehl **ng g component <pfad/name>** können über die Angular CLI Components angelegt werden

Frontend Development

Components – Grundlagen

React / NextJS

- NextJS hat eine „Root“ Component in dieser können wir unsere Footer Component wie ein HTML Element einbinden
- Dafür importieren wir einfach die Function (Also die Component)

Angular

- In Angular gibt es Standalone und Module Components
 - Der Default seit einigen Versionen ist Standalone
- In der Angular Root Component können wir über das **imports** Array einfach Standalone Components importieren und dann im HTML mit ihrem Selektor aufrufen

Frontend Development

Components – Grundlagen

React / NextJS

```
1 export default function RootLayout({  
2   children,  
3 }: Readonly<{  
4   children: React.ReactNode;  
5 >>) {  
6   return (  
7     <html lang="en">  
8       <body className={`${geistSans.variable} ${geistMono.variable} ${styles.layout}`}>  
9         <header className={styles.header}>  
10           <BeerHeader>/<BeerHeader>  
11         </header>  
12         <main className={styles.main}>{children}</main>  
13         <footer className={styles.footer}>  
14           <BeerFooter>/<BeerFooter>  
15         </footer>  
16       </body>  
17     </html>  
18   );  
19 }  
20
```

Angular

```
1 <div class="layout">  
2   <header class="header">  
3     <app-beer-header>/<app-beer-header>  
4   </header>  
5   <main class="main"></main>  
6   <footer class="footer">  
7     <app-beer-footer>/<app-beer-footer>  
8   </footer>  
9 </div>  
10  
11  
12 import { Component } from '@angular/core';  
13 import { RouterOutlet } from '@angular/router';  
14 import { BeerHeaderComponent } from './ui/beer-header/beer-header.component';  
15 import { BeerFooterComponent } from './ui/beer-footer/beer-footer.component';  
16  
17 @Component({  
18   selector: 'app-root',  
19   standalone: true,  
20   imports: [RouterOutlet, BeerHeaderComponent, BeerFooterComponent],  
21   templateUrl: './app.component.html',  
22   styleUrls: ['./app.component.css']  
23 })  
24 export class AppComponent {  
25   title = 'beer-app-angular';  
26 }
```

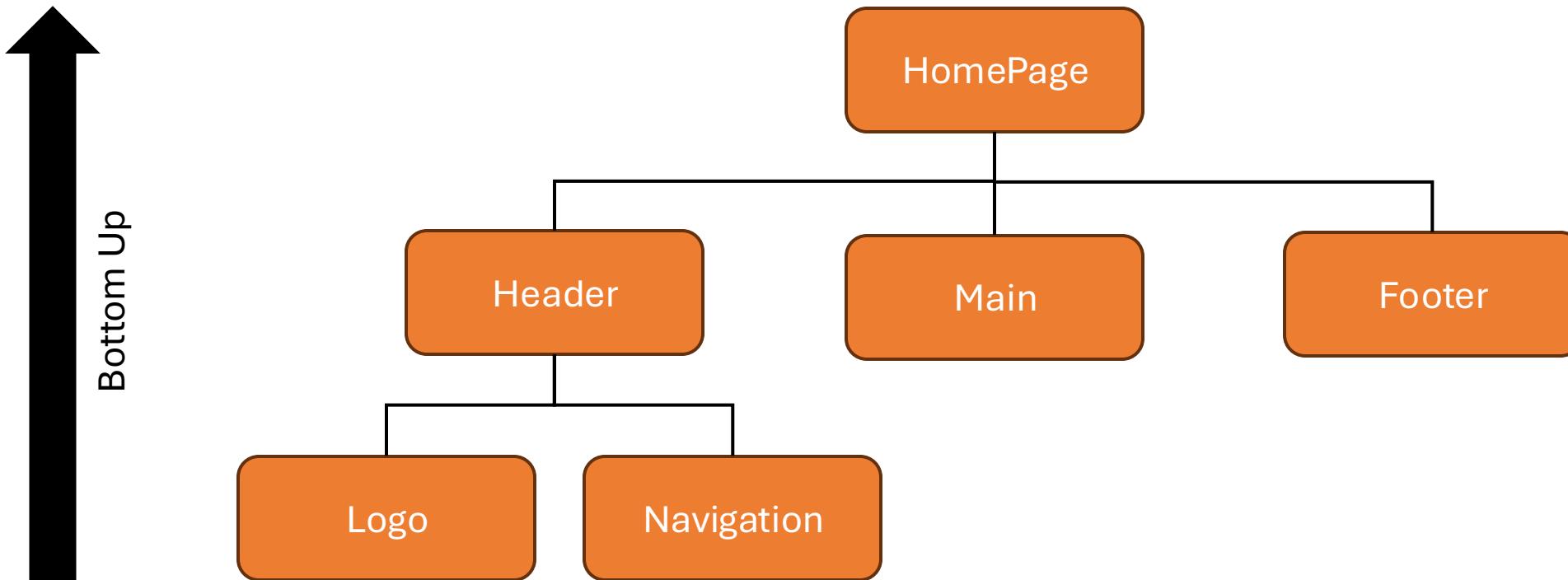
Frontend Development

Components – Grundlagen

- In beiden Frameworks definieren wir uns also Components als eigenständige Bausteine und können diese in anderen Components einfach als HTML Elemente verwenden
- Auf diese Art und Weise können wir uns Component Trees zusammenstellen und Components beliebig verschachteln / nutzen

Frontend Development

Components – Grundlagen



Frontend Development

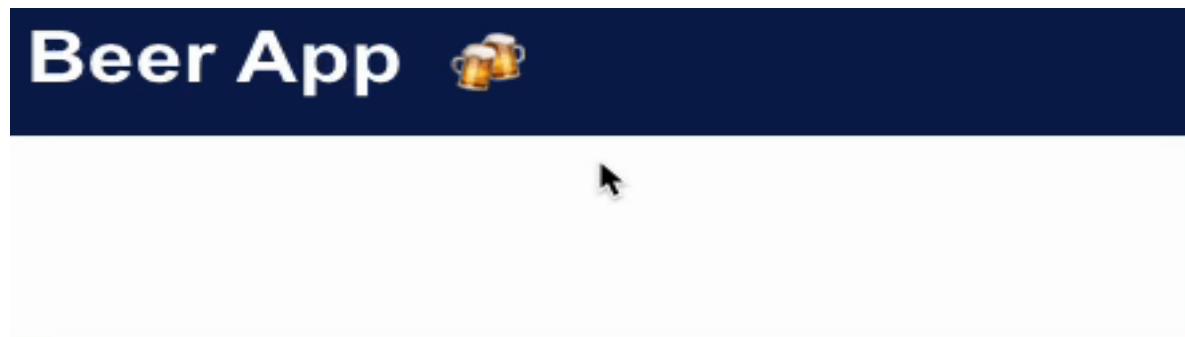
Components – State & Events

- Components können neben HTML Elementen auch einen internen State haben
- Mit diesem State können wir Variablen oder ähnliches über JavaScript einfach abbilden und im Template verwenden / anzeigen
- Auch können wir auf jedes HTML Element reagieren und anschließend den State verändern

Frontend Development

Components – State & Events

- Nachfolgend möchten wir wenn man über die Biere im Header hovered ein Prost! anzeigen



Frontend Development

Components – State & Events

React / NextJS

- React nutzt eine spezielle **useState** Funktion, um einen State innerhalb der Component abzubilden
- Die **useState** Funktion nimmt einen initialen Zustand und eine Funktion zum Verändern entgegen
- **useState** ist dabei ein sogenannter React Hook
 - Der Prefix **use** deutet immer auf einen Hook hin

Angular

- In Angular können wir primitiven State einfach als Variablen innerhalb der Klasse definieren
- Primitive Werte können dann einfach modifiziert werden
 - Vgl. mit normalen JavaScript
- Für nicht primitive Werte können wir eine Methode schreiben, die ein Objekt bspw. verändert

Frontend Development

Components – State & Events

React / NextJS

- State in React nutzt die Array Destructuring Syntax für die Definition
- Der eigentliche Wert vom State und die Update Funktion werden als Array Dabei gespeichert
- State Veränderungen sollten immer mit der Update Funktion ausgeführt werden!

Angular

- Sie können primitive Werte auch direkt im Template modifizieren
 - Es empfiehlt sich aber für solche Manipulationen immer Funktionen zu definieren
 - Dann kann Angular auch Automatisch die UI updaten

Frontend Development

Components – State & Events

React / NextJS

```
1  export default function BeerHeader() {
2    const [isHovered, setIsHovered] = useState(false);
3
4    return (
5      <div className={styles.header}>
6        <div className={styles.title}>
7          <h1>Beer App</h1>
8          <h1
9            onMouseEnter={() => setIsHovered(true)}
10           onMouseLeave={() => setIsHovered(false)}
11           className={isHovered ? styles.shake : ""}>
12        >
13          {isHovered && <span className={styles.prost}>Prost!</span>}
14        </h1>
15      </div>
16      <nav className={styles.navigation}>
17        <Link href="/" >🍺 Home</Link>
18        <a
19          href="https://www.bierbewusstgeniessen.de/drink-responsibly/"
20          target="_blank"
21        >
22          △ Drink Responsible
23        </a>
24      </nav>
25    </div>
26  );
27}
28 }
```

Angular

```
1  <div class="header">
2    <div class="title">
3      <h1>Beer App</h1>
4      <h1
5        (mouseenter)="isHovered = true"
6        (mouseleave)="isHovered = false"
7        [class]="isHovered ? 'shake' : ''"
8      >
9        @if (isHovered) {
10          <span class="prost">Prost!</span>
11        }
12      </h1>
13    </div>
14    <nav class="navigation">
15      <a routerLink="/" >🍺 Home</a>
16      <a
17        href="https://www.bierbewusstgeniessen.de/drink-responsibly/"
18        target="_blank"
19      >
20        △ Drink Responsible
21      </a>
22    </nav>
23  </div>
24
25  export class Beer HeaderComponent {
26    public isHovered: boolean = false;
27  }
```

Frontend Development

Components – State & Events

React / NextJS

- HTML Events haben in React (Speziell JSX) immer einen **on** Prefix und mit Attribute Definition kann auf Events reagiert werden
- Sie können dann direkt eine Funktion an das Eintretende Event übergeben
- Oder eine Anonyme Funktion aufrufen

Angular

- In Angular kann man auf Events mit () hören
- Angular exposed alle bekannten HTML Events von HTML Elementen sowie die Möglichkeit eigene Events zu definieren
- Beim eintreten von einem Event kann eine Methode aufgerufen werden

Frontend Development

Components – State & Events

React / NextJS

- Auf den Wert von einem State können wir dank JSX einfach über die definierte Variable zugreifen
- Wichtig wenn Sie JavaScript Variablen verwenden möchten müssen Sie in JSX diese in ein {} wrappen

Angular

- Auf Variablen kann in Angular über die {{}} Syntax im Template zugegriffen werden
- Wenn man Variablen an ein Attribute Binden möchte, muss man das Attribut mit [] definieren
- Anschließend können die public Variablen einfach übergeben werden
 - Als Teil vom Attribute String

Frontend Development

Components – Props

- Wenn wir Components wiederverwenden möchten bietet es sich an, Daten von außerhalb an diese Components zu übergeben
- Definiert man sich „Dumb“ Components die einfach nur Daten anzeigen möchten wir den eigentlichen Inhalt über eine andere Component übergeben

Frontend Development

Components – Props / Inputs

- Nachfolgend möchten wir eine **Card** Component als Dumb Component erstellen
- Die Card Component soll später unsere Biere als HTML Elemente darstellen
- Die Card Component besteht dabei aus einem Header, Content und Footer Bereich
- Diese werden auch als einzelne Components abgebildet und den Content und Footer kann man beliebig von außerhalb definieren

Frontend Development

Components – Props / Inputs

React / NextJS

- Props werden einfach als Parameter an die Component Funktion übergeben
- Man kann einzelne Props als Variable definieren oder die Destructuring Assignment Syntax nutzen wenn man beliebige Props entgegen nimmt
- Als Props kann man jeden beliebigen JavaScript Datentyp übergeben

Angular

- Props in Angular heißen **Inputs** und werden als Decorator innerhalb der Klasse definiert
- Inputs müssen immer explizit angegeben werden
 - Es gibt aber auch den Vorteil, Inputs als Required zu setzen
- Inputs können jeden beliebigen JavaScript Datentyp annehmen

Frontend Development

Components – Props / Inputs

React / NextJS

```
1 import styles from './card.module.css';
2
3 export function CardHeader({...props}) {
4     return (
5         <div className={styles.header}>
6             <h3>{props.card.title}</h3>
7             <p>{props.card.subtitle}</p>
8         </div>
9     );
10 }
```

Angular

```
1 <div>
2   <h3>{{card.title}}</h3>
3   <p>{{card.subtitle}}</p>
4 </div>
5
```

...

```
1 export class CardHeaderComponent {
2   @Input({required: true}) card!: { title: string, subtitle: string };
3 }
```

Frontend Development

Components – Props / Inputs

React / NextJS

- Man kann aber auch andere Components als Props übergeben
- Mit diesem Konzept können wir unsere Card Content und Footer aufbauen die einfach eine fremde Component als Prop entgegen nehmen
- React exposed dafür ein besonderes **children** Prop

Angular

- Components können nicht als Inputs übergeben werden in Angular
- Stattdessen gibt es die Content Projection und ein spezielles Angular Element **ng-content**
 - <https://angular.dev/guide/components/content-projection>
- Über das Attribute **ngProjectAs** können dann gezielt Components eingesetzt werden

Frontend Development

Components – Props / Inputs

React / NextJS

```
1 import styles from './card.module.css';
2
3 export function CardContent({...props}) {
4   return (
5     <div>
6       {props.children}
7     </div>
8   );
9 }
```

Angular

```
1 <div>
2   <ng-content></ng-content>
3 </div>
4
```

Frontend Development

Components – Props / Inputs

React / NextJS

- An der Stelle wo man **props.children** aufruft wird die übergebene Component gerendert
- Man kann auch verschiedene Components übergeben und diese explizit an andere Components übergeben

Angular

- An der Stelle wo **ng-content** definiert ist, wird die übergebene Component gerendert
- Man kann auch verschiedene Components übergeben und einzelnen **ng-content** Elementen Selektoren definieren

Frontend Development

Components – Props / Inputs

React / NextJS

```
1  export function Card({ children, ...props }) {  
2    const [content, footer] = React.Children.toArray(children);  
3  
4    return (  
5      <div className={styles.card}>  
6        <CardHeader {...props}></CardHeader>  
7        <CardContent>{content}</CardContent>  
8        <CardFooter>{footer}</CardFooter>  
9      </div>  
10    );  
11  }  
12
```

Angular

```
1  <div class="card">  
2    <app-card-header [card]="card"></app-card-header>  
3    <app-card-content>  
4      <ng-content select="card-content"></ng-content>  
5    </app-card-content>  
6    <app-card-footer>  
7      <ng-content select="card-footer"></ng-content>  
8    </app-card-footer>  
9  </div>  
10
```

Frontend Development

Components – Props / Inputs

- Mit der Card Base Component können wir uns anschließend eine Beer Component erstellen
- Die Beer Component nutzt die Card Component, um ein Bier darzustellen also eine Parent Component
- Die Parent Component implementiert dann die Logik ein Bier zu trinken oder zu bewerten

Frontend Development

Components – Props / Inputs

React / NextJS

```
1 return (
2   <Card card={{ title: beer.name, subtitle: beer.brand }}>
3     <div className={styles.content}>
4       <BeerMug rating={beer.rating} onClick={handleDrink}></BeerMug>
5     </div>
6     <div className={styles.footer}>
7       <h4>{beer.amount}</h4>
8       <BeerRatingSlider
9         currentRating={beer.rating}
10        onRatingSelect={handleRatingSelect}
11      ></BeerRatingSlider>
12    </div>
13  </Card>
14 );
```

Angular

```
1 <app-card [card]="{ title: beer.name, subtitle: beer.brand }">
2   <div class="content" ngProjectAs="card-content">
3     <app-beer-mug
4       [rating]="beer.rating"
5       (clickEvent)="handleDrink()"
6     ></app-beer-mug>
7   </div>
8   <div class="footer" ngProjectAs="card-footer">
9     <app-beer-rating-slider
10       [currentRating]="beer.rating"
11       (ratingSelect)="handleRatingSelect($event)"
12     ></app-beer-rating-slider>
13     <h4>{{ beer.amount }}</h4>
14   </div>
15 </app-card>
16
```

Frontend Development

Components – Props / Inputs

React / NextJS

- In React definieren wir die Logik einfach als Anonyme Funktion innerhalb der Component
- Die Funktion selbst wird dann als prop an das Children gegeben, welches den Event Listener hat
- Die Child Component ruft anschließend die Funktion einfach auf

Angular

- In Angular definieren wir die Logik einfach als Methoden innerhalb der Klasse
- Anders als in React wird die Methode nicht übergeben, sondern Children Components können über Events mit der Parent Component kommunizieren
- Das Children emittet ein Event und das Parent ruft seine Methode auf

Frontend Development

Components – Props / Inputs

React / NextJS

```
1 export function BeerRatingSlider({ currentRating, onRatingSelect }) {
2   const [hoveredRating, setHoveredRating] = useState(0);
3
4   const handleMouseEnter = (rating) => {
5     setHoveredRating(rating);
6   };
7
8   const handleMouseLeave = () => {
9     setHoveredRating(0);
10  };
11
12  const handleClick = (rating) => {
13    onRatingSelect(rating);
14  };
15
16  const StarIcon = ({ filled }) => (
17    <svg
18      xmlns="http://www.w3.org/2000/svg"
19      viewBox="0 0 24 24"
20      className={`${styles.star} ${filled ? styles.filled : ""}`}
21    >
22      <path d="M12 17.27L18.18 21l-1.64-7.03L22 9.24l-7.19-.61L12 2 9.19 8.63 2 9.24L5.46 4.73L5.82 21z" />
23    </svg>
24  );
25
26  return (
27    <div className={styles.ratingContainer}>
28      {[1, 2, 3, 4, 5].map((star) => (
29        <span
30          key={star}
31          onMouseEnter={() => handleMouseEnter(star)}
32          onMouseLeave={handleMouseLeave}
33          onClick={() => handleClick(star)}
34          role="button"
35          aria-label={`Rate ${star} stars`}
36        >
37          <StarIcon filled={star <= (hoveredRating || currentRating)} />
38        </span>
39      )));
40    </div>
41  );
42}
```

Angular

```
1 <div class="ratingContainer">
2   @for(star of ratings; track star) {
3     <span
4       (mouseenter)="handleMouseEnter(star)"
5       (mouseleave)="handleMouseLeave()"
6       (click)="handleClick(star)"
7       role="button"
8       [attr.aria-label]="'Rate {{star}} stars'"
9     >
10    <svg
11      xmlns="http://www.w3.org/2000/svg"
12      viewBox="0 0 24 24"
13      [class]={`${star: true, filled: isFilled(star)}`}
14    >
15      <path
16        d="M12 17.27L18.18 21l-1.64-7.03L22 9.24l-7.19-.61L12 2 9.19 8.63 2 9.24L5.46 4.73L5.82 21z"
17      />
18    </svg>
19  </span>
20 }
21 </div>
22
23 export class BeerRatingSliderComponent {
24   @Input({required: true}) currentRating!: number;
25
26   @Output() ratingSelect: EventEmitter<number> = new EventEmitter<number>();
27
28   public hoveredRating = 0;
29   public ratings = [1, 2, 3, 4, 5];
30
31   public handleClick(rating: number): void {
32     this.ratingSelect.emit(rating);
33   }
34
35   public handleMouseEnter(rating: number): void {
36     this.hoveredRating = rating;
37   }
38
39   public handleMouseLeave(): void {
40     this.hoveredRating = 0;
41   }
42
43   public isFilled(star: number): boolean {
44     return star <= (this.hoveredRating || this.currentRating);
45   }
46 }
```

Frontend Development

Components – Props / Inputs

React / NextJS

- Die Child Component kommuniziert also nicht mit dem Parent sondern ruft einfach Funktionen, welche als Prop übergeben werden auf
- Eine weitere Besonderheit in der Beer Rating Slider Implementierung, wir können auch Components innerhalb einer Component definieren
 - Siehe **StarIcon**

Angular

- Die Child Component kann über ein **Output** Event eine „Nachricht“ an den Parent zurück geben
- Ein Output Event kann dabei auch Daten nach oben schicken oder einfach nur ein Event auslösen
- Wenn der User auf einen Stern klickt, wird ein **ratingSelect** Event ausgelöst
 - Auf dieses reagiert der Parent

Frontend Development

NextJS – Server & Client Components

- NextJS unterstützt zwei Arten von Components: Client Side Components und Server Side Components
- Client Side Components werden auf dem Client (Browser) ausgeführt und erlauben Interaktive Components
- Server Side Components werden auf dem Server vorgerendert und das fertige Ergebnis an den Client geschickt
 - So wie das Web bis vor Single Page Apps funktioniert hat

Frontend Development

NextJS – Server & Client Components

- Auf Server Side Components können wir **nicht** mit allen Browser APIs und React Funktionalitäten arbeiten
 - NextJS wirft beim Ausführen einen genauen Fehler, welche API nicht funktioniert
- Per Default sind alle Components in NextJS Server Side Components
- Wenn wir also in der Parent Component Beer Card die Logik zum Bewerten und Trinken implementieren, müssen wir die Beer Card als Client Seitige Component deklarieren

Frontend Development

NextJS – Server & Client Components

- Hierfür schreiben wir in die alle erste Zeile ein “**use client**“ String
 - NextJS erkennt dann beim compilieren automatisch, dass es sich um eine Client Component handelt und führt diese anders aus
- Server Side Components haben dagegen den Vorteil, dass wir Business Logik wie Datenbank Abfragen direkt in der Component implementieren können, und der Client diese Abfragen nicht sieht
- Der Client bekommt das fertige Ergebnis direkt als HTML angezeigt

Frontend Development

NextJS – Server & Client Components

- In unserem Beispiel macht es also Sinn die Beer Card Component als Client Component zu deklarieren, weil der User direkt mit dieser interagiert und ein Bier bewerten bzw. trinken kann
- Auf der anderen Seite können wir die Liste mit allen Bieren direkt als Server Component vorab rendern, dadurch spart sich der Client einen HTTP Request, um alle Biere abzufragen

Frontend Development

Angular – Server & Client Components

- Der Trend wieder mehr Rendering auf Server Seite auszuführen, wird mittlerweile in einer ersten Version auch von Angular unterstützt
- Aufgrund der noch Instabilen API und verfügbaren Funktionen, betrachten wir in der Vorlesung nur Client Side Rendering in Angular
 - Also eine klassische Single Page Application
- Das Konzept wird aber unter folgenden Link einmal vorgestellt
<https://angular.dev/guide/hydration>

Frontend Development

Asynchrone Operationen

- In der Beer Card Component gibt es die Operationen **handleDrink** und **handleRatingSelect**
- Wenn wir uns das React Beispiel einmal anschauen, erkennen wir einige neue Konzepte, welche wir noch nicht betrachtet haben

```
1 const handleDrink = async () => {
2   try {
3     const apiResponse = await drinkBeer(beer);
4     setBeer(() => ({ ...apiResponse.beer }));
5   } catch (error) {
6     console.error("Error drinking beer:", error);
7   }
8};
```

Frontend Development

Asynchrone Operationen

- In Programmiersprachen gibt es Operationen, welche nicht sofort (also Synchron) ausgeführt werden können
- Bspw. wenn Sie eine Datei lesen möchten, müssen Sie einen Moment warten bis das Betriebssystem Ihnen die Datei zur Verfügung stellt.
- Bspw. wenn Sie mit einer Datenbank sprechen möchten, müssen Sie warten bis die Datenbank ihnen die gewünschten Daten zurückgibt

Frontend Development

Asynchrone Operationen

- Und wenn wir mit einer Schnittstelle sprechen möchten also einen HTTP Request ausführen wollen
- Eine HTTP Anfrage benötigt ein Netzwerk für die Anfrage und muss warten bis der Server eine Antwort geschickt hat
- In der Zwischenzeit kann die Anwendung aber bspw. andere Operationen ausführen

Frontend Development

Asynchrone Operationen

- JavaScript bietet zwei Konzepte für die Asynchrone Programmierung:
 - Async Callbacks
 - Promises

Frontend Development

Asynchrone Operationen – Async Callbacks

- Eine Callback Funktion, welche an eine andere Funktion übergeben wird, ist asynchrone Programmierung, weil die übergebene Funktion erst zu einem späteren Zeitpunkt ausgeführt wird
- Aber der JavaScript Prozess wartet nicht wirklich auf die Operation, sondern macht einfach mit dem Programm weiter

Frontend Development

Asynchrone Operationen – Async Callbacks

- Async Callbacks sind in modernen Web APIs immer weniger zu finden, weil Sie dazu tendieren Programme deutlich Komplexer zu gestalten, als Notwendig ist
- Ein häufiges Problem sind viele Async Callbacks die mehrfach im Code aufgerufen werden
 - Auch bekannt als Callback Hell
- Auch ist es deutlich schwieriger sauberes Fehler Handling einzubauen, weil man nicht im Vorfeld sagen kann, an welcher Callback Funktion ein Fehler geworfen wird

Frontend Development

Asynchrone Operationen – Promises

- Deshalb gibt es in JavaScript ein spezielles Objekt **Promise** für Asynchrone Programmierung
- Eine **Promise** ist dabei ein Objekt, welches von einer asynchronen Funktion zurückgegeben wird
- Das **Promise** Objekt beinhaltet Methoden, welche ausgeführt werden können wenn die Operation erfolgreich oder fehlerhaft war

Frontend Development

Fetch API

- Wir möchten eine JavaScript Funktion schreiben, welche alle Biere von unserer Beer API über einen HTTP Request abfragt
- Für die Interaktion mit dem HTTP Protokoll bietet JavaScript die Fetch API an
 - Seit Node 18 gibt es die Node API auch in NodeJS
- **Fetch** ist dabei eine asynchrone Funktion, welche uns eine Promise zurück gibt

Frontend Development

Fetch API



```
1 const fetchPromise = fetch('http://localhost:8080/beer');
2
3 console.log(fetchPromise);
4
5 fetchPromise.then((response) => {
6   console.log(`Received response: ${response.status}`)
7 })
8
9 console.log('Start getting all beers');
```

```
$ node 01-fetch.js
Promise { <pending> }
Start getting all beers
Received response: 200
```

Frontend Development

Fetch API

- Das Promise Objekt speichern wir uns in einer Variable und rufen unseren HTTP Endpunkt über eine URL auf
- Das Promise Objekt geben wir anschließend aus und bekommen einen **Pending** State als aktuellen Wert
- Das Promise Objekt exposed eine spezielle **then** Funktion, wenn die Promise erfolgreich abgeschlossen ist, wird die **then** Funktion automatisch aufgerufen
- Die **then** Funktion bekommt dann die HTTP Antwort als **Response** Parameter

Frontend Development

Fetch API

- Mit demselben Konzept können wir anschließend die JSON Antwort ausgeben
- Das **response** Objekt beinhaltet eine **.json()** Methode, welche auch wieder asynchron ist und eine Promise zurückgibt
- Auf die JSON Promise können wir dann auch wieder einen **.then** Handler hängen

Frontend Development

Fetch API



```
1 const fetchPromise = fetch("http://localhost:8080/beer");
2
3 fetchPromise
4   .then((response) => response.json())
5   .then((beers) => {
6     console.log(beers);
7   });
8
9 console.log("Start getting all beers");
10
```

Frontend Development

Promises – Grundlagen

- Eine Promise kann sich in drei Zuständen befinden:
 - **Pending** – Die Promise wurde erstellt und ist noch nicht abgeschlossen. Die Operation läuft also noch
 - **Fulfilled** – Die Asynchrone Operation war erfolgreich und der **.then** Handler wird aufgerufen
 - **Rejected** – Die Asynchrone Operation ist fehlgeschlagen und der **.catch** Handler wird aufgerufen
- Eine Rejection kann bspw. ausgelöst werden, wenn die HTTP Antwort nicht erfolgreich war (Status Code != 2xx) und anschließend eine Exception geworfen wird

Frontend Development

Async / Await Syntax

- Weil JavaScript keinen Rückgabewert für eine Funktion definieren kann können Promises über das **async** Keyword deklariert werden
- Eine Funktion, welche **async** ist, gibt automatisch eine Promise zurück
- Wenn wir jetzt warten möchte, bis die Promise ausgeführt ist, können wir mit dem **await** Keyword direkt auf die Antwort warten

Frontend Development

Async / Await Syntax



```
1  export async function drinkBeer(beer) {
2    const response = await fetch(`#${BASE_BEER_URL}/${beer.id}/drink`, {
3      method: "POST",
4      headers: {
5        "Content-Type": "application/json",
6      },
7      body: JSON.stringify(beer),
8    );
9
10   if (!response.ok) {
11     throw new Error("Failed to drink beer");
12   }
13
14   const data = await response.json();
15   return data;
16 }
17
```

Frontend Development

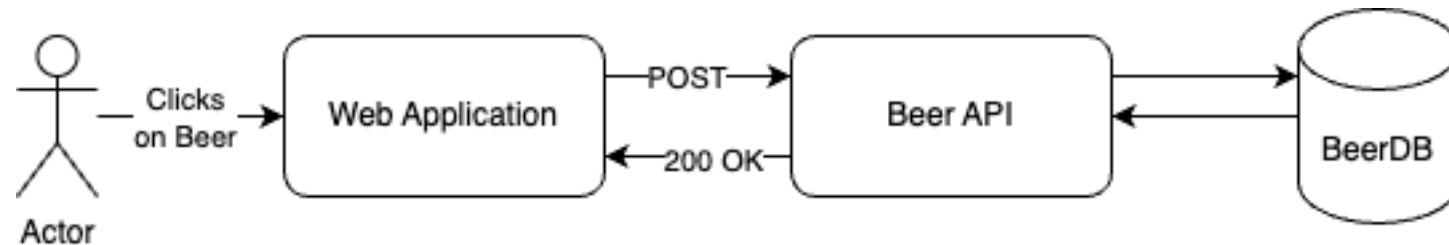
Promises – Fazit

- Promises sind das Fundament für die Asynchrone Programmierung in JavaScript
- Mit Promises können wir Sequenzen von Asynchronen Operationen einfach abbilden, ohne tiefe Callbacks
- Mit **async** und **await** können wir darüber hinaus noch die Promise Chains (`.then().then`) uns sparen und Code schreiben, der wie Synchroner Code aussieht
- Promises werden von allen modernen Browsern unterstützt und auch NodeJS bietet mittlerweile viele APIs als Promise Based Functions an

Frontend Development

HTTP Kommunikation

- Unsere BeerCard Component hat zwei Asynchrone Operationen:
 - Ein Bier trinken
 - Ein Bier bewerten
- Beide Operationen sollen mit unserer Beer API aus Meilenstein 05 sprechen



Frontend Development

HTTP Kommunikation

React / NextJS

- In React definieren wir uns zwei Event Handler die ausgeführt werden, wenn ein Bier getrunken oder bewertet wird
- Der Event Handler ruft eine definierte API Funktion auf, welche eine Promise zurück gibt
- Anschließend wird das aktuelle Bier im State aktualisiert, mit der Response vom Server

Angular

- In Angular definieren wir uns zwei Methoden auf unserer Klasse, die ausgeführt werden, wenn ein Bier getrunken oder bewertet wird
- Die Methoden rufen dann einen von uns definierten Service Funktion auf, welche ein Observable zurück gibt
- Sobald das Observable ein Event zurück gibt, wird das aktuelle Bier überschrieben

Frontend Development

HTTP Kommunikation

React / NextJS

- Den eigentlichen API Aufruf lagern wir in eine eigene Funktion aus, die unabhängig von React / NextJS verwendet werden kann
- In diesen Funktionen nutzen wir die Fetch API für die HTTP Aufrufe gegen unsere Endpunkte
- Mit diesem Ansatz können wir API Aufrufe in Client und Server Components wiederverwenden

Angular

- Die eigentliche Logik lagern wir in einen Angular Service aus.
- Services können wir als Dependency in eine Klasse injecten
 - Dependency Injection
- Mit diesem Ansatz können wir Business Logik ausgelagert in einem Service definieren aber Service Informationen in jede Component injecten

Frontend Development

HTTP Kommunikation

React / NextJS

```
1 const handleDrink = async () => {
2   try {
3     const apiResponse = await drinkBeer(beer);
4     setBeer(() => ({ ...apiResponse.beer }));
5   } catch (error) {
6     console.error("Error drinking beer:", error);
7   }
8 };

1 export async function drinkBeer(beer) {
2   const response = await fetch(`.${BASE_BEER_URL}/${beer.id}/drink`, {
3     method: "POST",
4     headers: {
5       "Content-Type": "application/json",
6     },
7     body: JSON.stringify(beer),
8   );
9
10  if (!response.ok) {
11    throw new Error("Failed to drink beer");
12  }
13
14  const data = await response.json();
15  return data;
16 }
```

Angular

```
1 constructor(private beer ApiService: Beer ApiService) {}
2
3 public handleDrink() {
4   this.beer ApiService.drinkBeer(this.beer).subscribe((response) => {
5     this.beer = response.beer;
6   });
7 }

1 @Injectable({
2   providedIn: 'root'
3 })
4 export class Beer ApiService {
5   private readonly BASE_BEER_URL = 'http://localhost:8080/beer';
6
7   constructor(private http: HttpClient) { }
8
9   public drinkBeer(beer: any): Observable<any> {
10     return this.http.post(`.${this.BASE_BEER_URL}/${beer.id}/drink`, beer);
11   }
12 }
```

Frontend Development

Reaktive Programmierung

- Angular verwendet (anders als vielleicht gewohnt) sehr stark Paradigmen aus der Reaktiven Programmierung
- Reaktive Programmierung unterscheidet sich, dass wir nicht explizit auf Änderungen prüfen müssen, sondern reaktiv auf eine Änderung reagieren
- Reaktive Programmierung stellen im Normalfall einen Mechanismus bereit, dass wir uns auf Änderungen „abonnieren“ können
 - Wenn die Änderung eintritt, werden wir darüber informiert und können reagieren

Frontend Development

Reaktive Programmierung

- Mit Reaktiver Programmierung müssen wir also nicht prüfen ob ein Benutzer eine bestimmte Interaktion ausgelöst hat oder eine Daten Operation vollständig abgeschlossen ist
- Stattdessen definieren wir uns ein Observer / Subscriber Pattern
 - Ein Observer (Beobachter) subscribed (abonniert) sich auf ein Observable (Beobachtbaren) Objekt
- Löst das Observable ein Event aus, reagiert der Observer selbständig als Teil seiner subscribe Funktion

Frontend Development

Reaktive Programmierung

- Diese Art der Programmierung finden Sie immer mehr gerade in UI-Applikationen aber auch wenn Sie Event getriebene Systeme entwerfen
 - Event Driven Architecture
- Und speziell Angular nutzt Reactive Programmierung für fast alle internen Mechanismen wie wir gleich sehen werden

Frontend Development

Angular – Reaktive Programmierung

- Für HTTP Anfragen in Angular nutzen wir nicht die bekannte Fetch API sondern ein spezielles Angular HTTP Module
- Dieses Module können wir in unserem Service injecten
 - Dependency Injection immer über die Constructor Funktion
- Das HTTP Module übernimmt dabei die Aufgaben den Body als JSON zu parsen und wir können bequem auf die bekannten HTTP Methoden zugreifen

Frontend Development

Angular – Reaktive Programmierung

- Aber anstelle von einer Promise bekommen wir vom HTTP Module ein Observable für unsere Anfrage zurück
- Ein Observable ist dabei ein spezielles Objekt, welches uns eine Datenquelle (In Form von Events) zur Verfügung stellt, auf die wir asynchron reagieren können
- Ein Observable gibt dabei so lange Events zurück, wie es Subscriber gibt
 - **Wichtig:** Wenn Sie auf Observables nicht subscriben sind diese in Angular „cold“ also führen nichts aus

Frontend Development

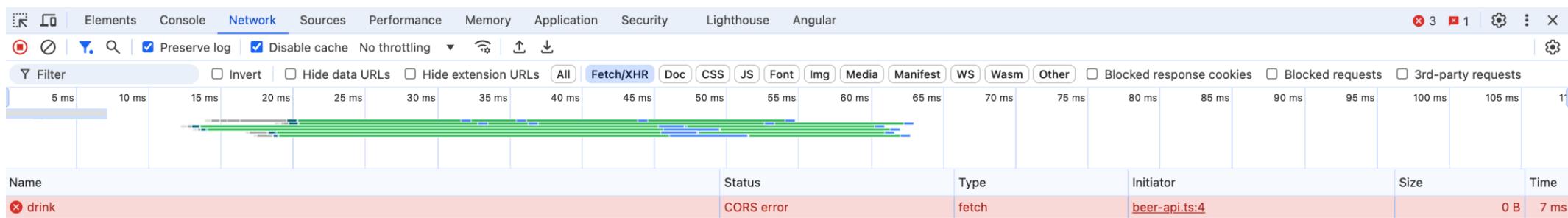
Angular – Reaktive Programmierung

- Damit unser HTTP Request also ausgeführt wird subscriben wir uns auf die **drinkBeer** Funktion
- Sobald die HTTP Anfrage abgeschlossen ist, erhalten wir die Antwort automatisch im **.subscribe** Teil unserer **handleDrink** Methode
- Ein Observable gibt dabei auch drei Verschiedene Zustände in Form von Objekten zurück (**Next**, **Error**, **Complete**)
 - Betrachten wir hier noch nicht

Frontend Development

CORS

- Wenn wir aber jetzt unsere Anwendung ausführen und ein Bier trinken wollen werden wir einen Fehler bekommen
- Mit F12 und ein Blick in das Network können wir sehen, dass unser **drink** request wegen einem CORS Error abgelehnt wurde



Frontend Development

CORS

- CORS (Cross Origin Resource Sharing) ist ein Sicherheitsmechanismus, der den Zugriff auf Ressourcen von einer Domain auf eine andere steuert, um unerwünschte Zugriffe zu verhindern
- CORS erlaubt Webanwendungen, bestimmte Anfragen an eine andere Domain zu senden, die nicht zur ursprünglichen Domain gehört, indem der Server spezifische Header zurückgibt, die die Zugriffsrechte definieren
- CORS wird vom Browser als Schutz verwendet vor potenziellen schädlichen Anfragen und ermöglicht gleichzeitig sicherer Datenaustausch zwischen verschiedenen Ursprüngen

Frontend Development

CORS

- Aus diesem Grund installieren wir uns das Fastify CORS Plugin und erlauben explizit die Quellen localhost:3000 und localhost:4200 auf unserem Server
 - Siehe <https://github.com/fastify/fastify-cors>

```
1 // CORS integration for Frontend Development
2 fastify.register(cors, {
3   origin: (origin, callback) => {
4     const allowedOrigins = ['http://localhost:3000', 'http://localhost:4200'];
5     if (!origin || allowedOrigins.includes(origin)) {
6       callback(null, true);
7     } else {
8       callback(new Error('Not allowed by CORS'));
9     }
10   }
11 });


```

Frontend Development

CORS

- Anschließend können wir Anfragen an unsere Beer API schicken ohne Fehler im Netzwerk

Frontend Development

HTTP Kommunikation

React / NextJS

- In Server Side Components können wir einfach die Fetch Funktion direkt aufrufen und das Ergebnis anzeigen
- Die Liste mit allen Bieren können wir so auf Server Side Rendern und direkt ausgeben
- Wir brauchen also keine Lifecycle Methoden

Angular

- Für GET Anfragen können wir den Prozess rund um das Subscriben vereinfachen indem wir die **Async** Pipe in Angular nutzen
- Wir speichern das Observable in einer Variable und nutzen die **Async** Pipe
- Angular macht dann selbstständig die Subscription und die Daten werden zurückgegeben

Frontend Development

HTTP Kommunikation

React / NextJS

```
1 export default async function BeerList() {  
2   const beers = await fetchBeers();  
3  
4   return (  
5     <div className={styles.list}>  
6       {beers.map((beer) => (  
7         <BeerCard key={beer.id} providedBeer={beer}></BeerCard>  
8       ))}  
9       <BeerCreateCard></BeerCreateCard>  
10    </div>  
11  )  
12}
```

Angular

```
1 export class BeerListComponent {  
2   public beers$;  
3  
4   constructor(private beer ApiService: Beer ApiService) {  
5     this.beers$ = this.beer ApiService.fetchBeers();  
6   }  
7 }  
  
1 <div class="list">  
2   @for(beer of beers$ | async; track beer.id) {  
3     <app-beer-card [beer]="beer"></app-beer-card>  
4   }  
5   <app-beer-create-card></app-beer-create-card>  
6 </div>  
7
```

Frontend Development

HTTP Kommunikation – Rule of Thumb

React / NextJS

- Wir definieren uns alle HTTP Anfragen als Asynchrone JavaScript Funktionen, welche die Fetch API nutzen
- Diese Funktionen importieren wir dann in den jeweiligen Components
- Alle GET Anfragen können wir direkt über Server Side Components handhaben

Angular

- Alle HTTP Interaktionen definieren wir in einem Service als Observables
- Die API Services Injecten wir uns dann in den jeweiligen Components
- GET Anfragen können wir mit der Async Pipe unkompliziert abbilden

Frontend Development

HTTP Kommunikation – Rule of Thumb

React / NextJS

- User Interaktionen (POST / PUT) können wir über Event Handler integrieren und anschließend den Component State wieder updaten

Angular

- User Interaktionen (POST / PUT) müssen wir subscriben, damit das Observable ausgelöst wird und wir eine Antwort erhalten

Frontend Development

Bier anlegen

- User sollen auch selbständig ihre Lieblings Bier Sorte über unsere Web Applikation anlegen können. Dafür können User einen speziellen Link öffnen und neue Biere anlegen. Den Link möchten wir später als QR Code auf Sticker drucken und auf Partys verteilen.

Frontend Development

Routing

- Web Applikationen (bzw. Allgemein Webseiten) bestehen normalerweise aus vielen unterschiedlichen Seiten
- Über unseren Browser und einer URL können wir als User solche Seiten besuchen
 - Alternativ über einen Anchor auf der Webseite selbst
- In Clientseitigen Web Applikationen fällt diese Aufgabe immer unter das Thema Routing

Frontend Development

Routing

React / NextJS

- NextJS verwendet einen File System basierten Router und wir können über Ordner Routen definieren
- Dabei definiert jeder Ordner ein Routen Segment, welches später Teil der URL ist
- Um eine Route sichtbar zu machen, gibt es eine spezielle **page.ts** Datei

Angular

- Angular hat ein spezielles Routing Module und Routen müssen explizit definiert werden
- Routen können in einer **.routes** Datei definiert werden und über die **app.config** als Provider deklariert werden
- Routen haben immer einen Pfad und eine Component

Frontend Development

Routing

React / NextJS

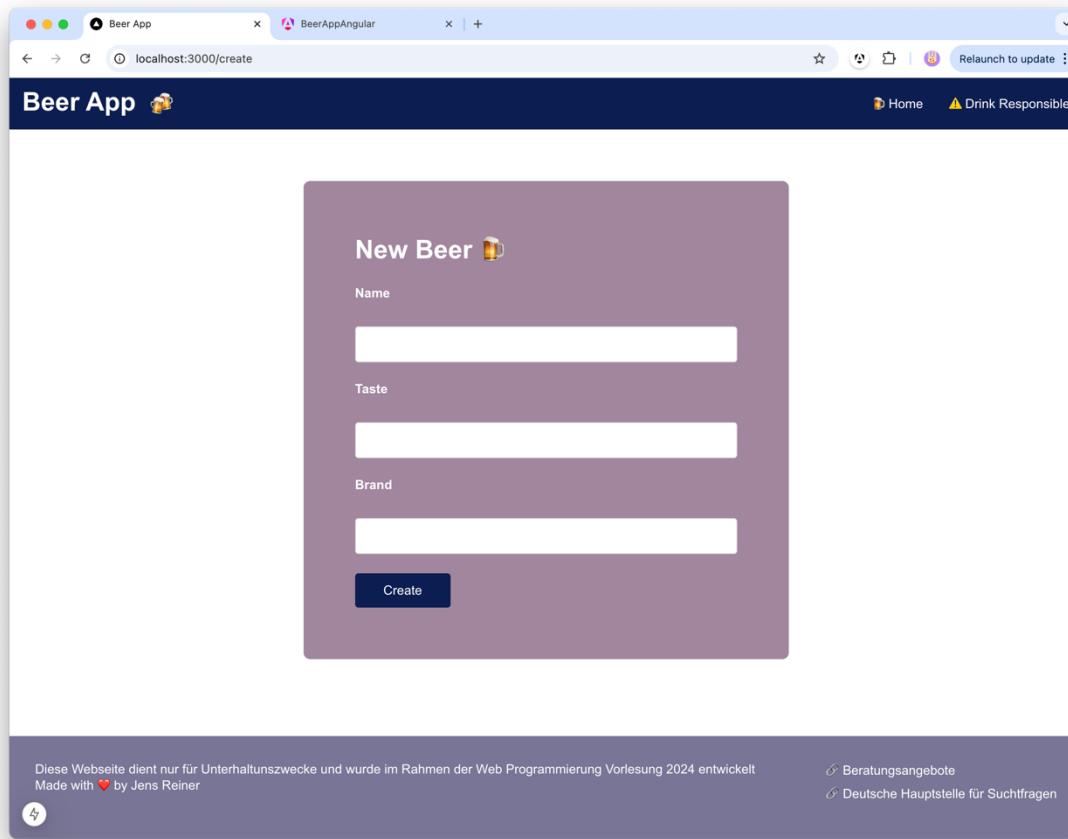
- Wir legen einen Ordner **create** im **app** Verzeichnis an und erstellen uns eine **page.ts** Datei
- In dieser **page** Component können wir jetzt unsere UI definieren, welche speziell für diese Route ist
- Tipp: Layouts die immer sichtbar sind können über eine spezielle **layout** Datei definiert werden

Angular

- Wir fügen eine neue Route **create** im **app.routes.ts** an und definieren die **BeerCreateComponent** darauf
- In der **app.component.html** nutzen wir jetzt für den **main** Teil ein spezielles **router-outlet** Element

Frontend Development

Routing



Frontend Development

Routing

React / NextJS

- Für die Navigation zwischen einzelnen Seiten bietet NextJS eine spezielle **Link** Component an
- **Link** erweitert dabei den nativen **a** Tag vom Web Browser
- Für Server Components gibt es noch die **redirect** Funktion

Angular

- Angular bietet eine Directive **routerLink** an, welche einfach auf einem anchor tag als attribute definiert werden kann
- Für Code Level Routing gibt es das **Router** und **ActivatedRoute** Module, welches über Dependency Injection einer Component übergeben werden kann

Frontend Development

Routing

React / NextJS

- Für dynamische Routen (bspw. ein Bier mit einer bestimmten ID) können Ordner mit square Brackets [] definiert werden
- Bspw: app/beers/[id]/page.ts
- Alle Dynamischen Segmente werden automatisch als Params Prop von NextJS übergeben für die Dateien **layout**, **page** und **route**

Angular

- Dynamische Routen werden auch mit der : Syntax in der routing Datei definiert
 - Vgl. Fasitfy Routen
- Über die Option **withComponentInputBinding** werden Routen Parameter automatisch als Inputs übergeben
 - <https://angular.dev/guide/routing/common-router-tasks#add-withcomponentinputbinding>

Frontend Development

Forms

- Web Forms sind Strukturen, die es Usern ermöglicht, Daten in eine Webseite einzugeben und an einen Server zu senden.
- Forms bestehen aus unterschiedlichen Form Controls (Textfield, Dropdown, Checkbox etc.) welche alle innerhalb einer **<form>** gruppiert sind.
- Die Daten aus einer Form können anschließend über das **action** Attribute als HTTP Anfrage an einen Server übermittelt werden
- Darüber hinaus können wir über Forms auch Validierungen implementieren

Frontend Development

Forms

React / NextJS

- Wir definieren uns eine spezielle Server Action **createBeer**
- Diese Action bekommt alle Daten aus der Form übergeben und macht einen HTTP Request an unsere Beer Schnittstelle
- NextJS exposed hierfür einen speziellen Hook **useActionState**
 - NextJS v15 notwendig bzw. React v19

Angular

- Wir definieren uns eine FormGroup mit unterschiedlichen Form Controls
- Angular bietet eine umfangreiche Forms API an für das Arbeiten mit Forms
 - Reactive Forms
- Die FormGroup können wir als Attribute auf die Form binden und beim Submit unseren API Service aufrufen

Frontend Development

Forms

React / NextJS

```
1 function SubmitButton() {
2   const { pending } = useFormStatus();
3
4   return (
5     <button type="submit" aria-disabled={pending}>
6       Create
7     </button>
8   );
9 }
10
11 export default function BeerCreate() {
12   const [state, formAction] = useActionState(createBeer, initialState);
13
14   return (
15     <div className={styles.layout}>
16       <form action={formAction}>
17         <h1>New Beer 🍺</h1>
18         <label htmlFor="name">Name</label>
19         <input type="text" id="name" name="name" required />
20         <label htmlFor="taste">Taste</label>
21         <input type="text" id="taste" name="taste" required />
22         <label htmlFor="brand">Brand</label>
23         <input type="text" id="brand" name="brand" required />
24         <SubmitButton />
25       </form>
26     </div>
27   );
28 }
```

Angular

```
1 constructor(private beer ApiService: Beer ApiService, private router: Router) {}
2
3 public createBeerForm = new FormGroup({
4   name: new FormControl(null, [Validators.required]),
5   taste: new FormControl(null, [Validators.required]),
6   brand: new FormControl(null, [Validators.required])
7 });
8
9 public handleFormSubmit() {
10   this.beer ApiService.createBeer(this.createBeerForm).subscribe(() => {
11     this.router.navigate(['/']);
12   });
13 }
14
15 <div class="layout">
16   <form [FormGroup]="createBeerForm" (ngSubmit)="handleFormSubmit()">
17     <h1>New Beer 🍺</h1>
18     <label for="name">Name</label>
19     <input type="text" formControlName="name" id="name" required />
20     <label for="taste">Taste</label>
21     <input type="text" formControlName="taste" id="taste" required />
22     <label for="brand">Brand</label>
23     <input type="text" formControlName="brand" id="brand" required />
24     <button type="submit" [disabled]="createBeerForm.invalid">Create</button>
25   </form>
26 </div>
27
```

Frontend Development

Forms

React / NextJS

```
1 "use server";
2
3 import { redirect } from 'next/navigation'
4 import { BASE_BEER_URL } from "../lib/beer-api-url";
5
6 export async function createBeer(prevState: any, formData: FormData) {
7   console.log(formData);
8
9   const response = await fetch(`#${BASE_BEER_URL}`, {
10     method: "POST",
11     headers: {
12       "Content-Type": "application/json",
13     },
14     body: JSON.stringify({
15       name: formData.get("name"),
16       taste: formData.get("taste"),
17       brand: formData.get("brand"),
18     }),
19   });
20
21   if (!response.ok) {
22     throw new Error("Error creating beer");
23   }
24
25   redirect('/');
26 }
27
```

Angular

```
1 public createBeer(formData: FormGroup): Observable<any> {
2   return this.http.post(this.BASE_BEER_URL, {
3     name: formData.get('name')?.value,
4     taste: formData.get('taste')?.value,
5     brand: formData.get('brand')?.value,
6   });
7 }
```

Frontend Development

Ausblick

- Und damit hätten wir die Grundlagen wie Sie moderne Web Applikationen mit entweder NextJS (React) oder Angular programmieren können
- Das war jetzt sehr viel auf einmal und vermutlich auch komplex wenn Sie vorher noch nie mit Client seitigen Frameworks gearbeitet haben
- Sie werden die vorgestellten Konzepte auch nicht von den Folien lernen – Sie müssen einmal mit den jeweiligen Frameworks eine Web Applikation umsetzen

Frontend Development

Ausblick

- Am Ende von diesem Kapitel gibt es aus diesem Grund nochmal eine Liste an Links, welche für das jeweilige Framework die Konzepte erklärt
- Abschließend wollen wir uns jetzt noch einmal anschauen, wie uns diese Frameworks helfen, Web Applikationen umzusetzen

Frontend Development

Fazit – Bausteine einer Web Applikation

- **User Interface** – Über Components definieren Sie UI Elemente die Informationen darstellen und User Interaktionen anbieten. In NextJS und Angular schieben Sie sich Components die sie beliebig verschachteln um vollständige Seiten zu programmieren
- **Routing** – NextJS bietet eine Datei System basiertes Routing System und spezielle Components für das Navigieren. Angular hat eine Router Implementierung und Routen müssen als Liste mit ihren Components definiert werden. Ein Directive in Angular ermöglicht Routing

Frontend Development

Fazit – Bausteine einer Web Applikation

- **Data Fetching** – In NextJS können Sie mit der Fetch API Daten auf Server Seite vorab laden oder über Event Handler User Interaktionen abbilden. In Angular nutzen Sie ein spezielles HTTP Module und Daten werden über Services bereit gestellt
- **Rendering** – Wie Components später im Browser dargestellt werden übernimmt das jeweilige Framework für Sie. NextJS bietet zudem Server Side Rendering automatisch mit an.
- **Integrations** – Beide Frameworks können beliebig erweitert werden und speziell in Angular können Integrationen über Services bereit gestellt werden. In NextJS können Sie sich Hooks definieren für Integrationen

Frontend Development

Fazit – Bausteine einer Web Applikation

- **Infrastructure** – Betrachten wir nicht in der Vorlesung. Sie können aber beide Lösungen ohne Probleme im Internet betreiben (NextJS 1 Click Deployments bei Vercel bspw)
- **Performance** – Ist abhängig von den Anwendungsfällen aber gerade in NextJS können Sie mit Server Side Rendering einiges an Performance optimieren. Die meisten Performance Aufgaben übernimmt aber das Framework für sie automatisch
- **Scalability** – Beide Frameworks eignen sich auch für tausende von Usern gleichzeitig. Bei Angular ist es u.a. wichtig, dass alles auf Client Seite läuft d.h ein Web Server muss nur das initiale HTML liefern während NextJS ggf. größere Server Ressourcen benötigt

Frontend Development

Fazit – Bausteine einer Web Applikation

- **Developer Experience** – Beide Frameworks beschleunigen die Entwicklung von Web Applikationen um ein vielfaches. Welches Framework am Ende eingesetzt wird ist u.a. auch Abhängig von den Anwendungsfällen und persönlichen Präferenzen

Frontend Development

Weiterführende Links

Component Design

- Keeping Components Pure (Beispiel in React die Argumentation lohnt sich aber)
<https://react.dev/learn/keeping-components-pure>
- Understanding Your UI as a Tree (Beispiel React die Argumentation lohnt sich aber)
<https://react.dev/learn/understanding-your-ui-as-a-tree>
- Component Driven User Interfaces <https://github.com/wasdJens/dont-panic-web-edition/blob/main/explanation/component-driven-development/component-driven-user-interfaces.md>

Frontend Development

Weiterführende Links

React

- Describing the UI using React <https://react.dev/learn/describing-the-ui>
- Importing and Exporting Components <https://react.dev/learn/importing-and-exporting-components>
- Writing Markup with JSX <https://react.dev/learn/writing-markup-with-jsx>
 - The rules of JSX <https://react.dev/learn/writing-markup-with-jsx#the-rules-of-jsx>
- JavaScript in JSX <https://react.dev/learn/javascript-in-jsx-with-curly-braces>
- Passing Props to a Component <https://react.dev/learn/passing-props-to-a-component>

Frontend Development

Weiterführende Links

React

- Adding Interactivity <https://react.dev/learn/adding-interactivity>
- Managing State <https://react.dev/learn/managing-state>
- Passing Data Deeply with Context <https://react.dev/learn/passing-data-deeply-with-context>
- Scaling Up with Reducer and Context <https://react.dev/learn/scaling-up-with-reducer-and-context>

Frontend Development

Weiterführende Links

NextJS

- Routing Fundamentals <https://nextjs.org/docs/app/building-your-application/routing>
- Data Fetching <https://nextjs.org/docs/app/building-your-application/data-fetching>
- Rendering Fundamentals <https://nextjs.org/docs/app/building-your-application/rendering>
- CSS <https://nextjs.org/docs/app/building-your-application/styling>

Frontend Development

Weiterführende Links

Angular

- Angular Essentials <https://angular.dev/essentials>
- HTTP Client <https://angular.dev/guide/http>
- Forms <https://angular.dev/guide/forms>
 - Nutzen Sie bitte immer Reactive Forms
- Dependency Injection <https://angular.dev/guide/di>
- Learn RxJS <https://www.learnrxjs.io>

Frontend Development

Übung

- Erweitern Sie eine der vorgestellten Web Applikationen (Angular ODER NextJS) um eine Detailseite für ein bestimmtes Bier.
- Ein einzelnes Bier soll als Seite über eine URL sichtbar sein und alle Informationen zum jeweiligen Bier darstellen. Hier können Sie auch wieder das Bier bewerten oder trinken.

Erweiterte Konzepte

Bonus Kapitel

Erweiterte Konzepte

Hinweis

- Die nachfolgenden Kapitel können alle im Eigenstudium bearbeitet werden und hatten leider keinen Platz in der Vorlesung direkt
- Sie finden hier u.a. spezielle Vertiefungen zu einzelnen Themen – Wenn Sie bspw. mit React arbeiten wollen finden Sie hier noch zusätzliche Bibliotheken und Tipps
- Sie können aber jederzeit Fragen zu jedem Thema stellen, falls etwas unklar erklärt ist

Erweiterte Konzepte

Inhalte

- How to Win – Wie man Webanwendungen richtig baut
- Debugging – Fehlersuche in JavaScript
- Datenaustausch mit HTTP und Fetch API
- Fastify Fluent Schema
- State Management in Client Seitigen Frameworks
- Typescript
- Websockets

- Eine Empfehlung für Third Party Packages
- Eine Sammlung mit nützlichen Links

Referenzen

Eine Sammlung von praktischen Links

Referenzen

Allgemeines

FreeAPI ist eine Lernplattform die eine Umfangreiche API Implementation bereit stellt und sehr viele Beispiele für unterschiedliche API Interaktionen hat

- <https://freeapi.app>
- Wenn Sie Docker installiert haben, können Sie die Beispiele auch Lokal bei sich ausführen

Node Best Practices eine umfangreiche Liste über Node Best Practices aber auch allgemeinen Software Design Patterns

- <https://github.com/goldbergryoni/nodebestpractices>

Referenzen

React

- React Hooks for Data Fetching
 - <https://swr.vercel.app>
- React Hook Form für das Arbeiten mit Forms in React
 - <https://www.react-hook-form.com>
- Formik für das bauen von Forms in React (Ohne Tears)
 - <https://formik.org>
- TanStack Query für State Management und HTTP Interaktionen
 - <https://tanstack.com/query/latest>

Referenzen

React

- Eine Referenz für React Libraries in 2024
 - <https://www.robinwieruch.de/react-libraries/>
- Laden von Daten in React mit Performance in Mind
 - <https://www.developerway.com/posts/how-to-fetch-data-in-react>
- ShadnUI für eine umfangreiche Components Bibliothek
 - <https://ui.shadcn.com>

Referenzen

Angular

- Die offizielle Angular Dokumentation sollte ihre erste Anlaufstelle sein
 - <https://angular.dev/overview>
- Angular University ist eine Seite, die sich speziell auf praktische Problemstellungen spezialisiert und Lösungen erläutert
 - <https://angular-university.io>

Referenzen

VSCode Plugins

- Total TypeScript für das Interaktive Lernen von TypeScript
 - <https://marketplace.visualstudio.com/items?itemName=mattcock.ts-error-translator>
- Live Preview HTML Seiten direkt in VSCode anzeigen
 - <https://marketplace.visualstudio.com/items?itemName=ms-vscode.live-server>
- Live Share zusammen im eigenen Editor am selben Projekt arbeiten
 - <https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare>

Referenzen

UI Component Libraries

- Bootstrap
 - React <https://react-bootstrap.netlify.app>
 - Angular <https://ng-bootstrap.github.io/#/home>
- Material
 - React <https://mui.com/material-ui/>
 - Angular <https://material.angular.io>

How To Win

Wie man Webanwendungen richtig baut

How To Win

Bevor man anfängt

- Wir haben uns in der Vorlesung sehr viele Konzepte rund um die moderne Web Entwicklung angeschaut, bevor man also mit einer Softwarelösung startet sollte man sich mit den Grundlagen vertraut machen und die Konzepte verstehen
- Anbei ein Überblick über die wichtigsten Konzepte mit Referenzen
- Auf der ersten Folie finden Sie Fragestellungen, wenn Sie diese alle beantworten können, probieren Sie auf der darauffolgenden Folie den Wissenstest einmal umsetzen

How To Win

Grundlagen – HTML / CSS

- Wie kann man Inhalte mit HTML beschreiben?
- Wie kann man mit CSS Inhalte gestalten?
- Referenzen:
 - Folien aus dem ersten Semester (Siehe Email)
 - <https://github.com/wasdJens/dont-panic-web-edition/blob/main/reference/html/01-html.md>
 - MDN https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started

How To Win

Wissenstest – HTML / CSS

- Sie sind in der Lage Informationen mit semantisch richtigen HTML auf einer Statischen Seite darzustellen. Mit Modernen CSS Techniken können Sie zudem das Layout beeinflussen und besondere Bereiche hervorheben.

How To Win

Grundlagen – JavaScript

- Was sind die wichtigsten Datentypen in JavaScript?
- Was sind Objekte in JavaScript?
- Wie können Sie mit Funktionen in JavaScript arbeiten?
- Wie kann man Funktionen als Parameter verwenden?
- Referenzen:
 - [Kapitel JavaScript eine Einführung in die Programmiersprache](#)

How To Win

Wissenstest – JavaScript

- Sie sind in der Lage Informationen in JavaScript Objekten abzubilden und Funktionen zu definieren, welche die Informationen bearbeiten können. Sie können sich ein kleines Programm schreiben, das Daten verwalten und modifizieren kann.

How To Win

Grundlagen – JavaScript Module

- Was sind Module in JavaScript?
- Was ist CJS und ESM in JavaScript?
- Was ist die Package.json in JavaScript?
- Wie können Sie externe Pakete installieren und verwenden?
- Referenzen:
 - [Kapitel Module in JavaScript](#)
 - [Kapitel NPM der Package Manager für NodeJS](#)

How To Win

Wissenstest – JavaScript Module

- Sie sind in der Lage ein neues Node Projekt aufzusetzen und das ESM Format zu verwenden. Mit NPM können Sie externe Abhängigkeiten installieren und in ihrem Quellcode verwenden. Einzelne Funktionalitäten lagern Sie in verschiedene Bereiche aus.

How To Win

Grundlagen – HTTP

- Was ist HTTP und wieso ist es so wichtig für das Internet?
- Was sind HTTP Methoden und Status Codes?
- Was sind HTTP Header?
- Was sind HTTP Anfragen und Responses?
- Was ist der HTTP Body?
- Was ist eine URL?
- Referenzen:
 - [Kapitel HTTP Definition und Überblick](#)

How To Win

Wissenstest – HTTP

- Sie sind in der Lage mit einem Tool eigenständig HTTP Anfragen an einen HTTP Server zu übermitteln und können über Request Header die Anfragen steuern. Mit der Server Response und Status Codes können Sie weitere Anfragen ableiten.

How to Win

Grundlagen – Backend Development

- Was ist eine Route?
- Was ist eine API Schnittstelle?
- Wie können Sie API Schnittstellen aufbauen?
- Was sind Besonderheiten von API Schnittstellen?
- Was ist das JSON Format und welche Vorteile bietet es?
- Referenzen:
 - [Kapitel Backend Development ein erster Entwurf für die Beer API](#)
 - [Kapitel Fastify Backend Development mit Fastify](#)
 - [Kapitel Rest Architektur](#)

How to Win

Wissenstest – Backend Development

- Sie sind in der Lage mit NodeJS und Fastify eine API Schnittstelle zu entwickeln, welche CRUD Operationen für eine Ressource anbieten und mit dem JSON Format Daten austauschen.

How to Win

Grundlagen – Frontend Development

- Was sind Components?
- Was ist das Bottom Up Konzept?
- Welche Aufgaben übernehmen Frameworks?
- Was ist die Fetch API?
- Was ist deklarative UI Entwicklung?
- Referenzen:
 - [Kapitel Frontend Development Webseiten mit Clientseitigen JavaScript entwickeln](#)
 - [Kapitel Frontend Development Web Applikationen mit Angular und NextJS](#)

How to Win

Wissenstest – Frontend Development

- Sie sind in der Lage mit einem gewählten Frontend Framework eine Client Applikation als Webseite zu gestalten und das User Interface mittels Components darzustellen. Über das HTTP Protokoll können Sie Daten mit einer Server Schnittstelle austauschen.

How To Win

Vorbereiten

- Anbei ein paar Informationen wie Sie sich optimal mit den vorgestellten Materialien vorbereiten und üben können

How To Win

Vorbereitung – Übungen

- Bearbeiten Sie einmal selbstständig und in Ruhe alle Übungen der Vorlesungen. Nehmen Sie sich für jede Übung ausreichend Zeit und versuchen Sie die Aufgabenstellung zu lösen
- Nutzen Sie auch die Beer Solution Meilensteine als Grundlage für ihre eigenen Lösungen bzw. erweitern Sie die Meilensteine um eigene Ideen um ein Gefühl für das Entwickeln von Web Applikationen zu bekommen
- Bitte fangen Sie **NICHT** an die Folien ohne parallel Code zu entwickeln durchzugehen. Zusammenfassen oder nur Lesen der meisten Folien wird Ihnen weniger bringen als Hand On Erfahrungen

How To Win

Vorbereitungen – Rubberducking

- Rubberducking ist ein Prozess aus der Softwareentwicklung ein Problem einer anderen Person zu erklären.
- Sie beschreiben das Problem Schritt für Schritt der „Gummiente“
- Während der Erklärung stellen Sie oft fest, wo das Problem liegt oder erhalten neue Erkenntnisse
- Ersetzen Sie bspw. die Gummiente mit einem AI Tool ihrer Wahl

How To Win

Vorbereitung – Übungsaufgaben von AI Tools

- Lassen Sie sich Aufgabenstellungen von AI Tools im Stile der Vorlesung generieren und versuchen Sie diese zu lösen
- Falls Sie vor einem Problem stehen, fragen Sie die AI nach einer Teillösung (Aber nicht nach der kompletten) und die Hintergründe

How To Win

Skizzieren von einer Softwarelösung

- Wenn Sie sich mit den Übungen sicher fühlen können Sie anfangen ihre Softwarelösung zu planen
- Bevor Sie aber direkt mit Code starten sollten Sie einmal probieren die Problemstellung zu skizzieren und wie mögliche Lösungen aussehen könnten
- Anbei ein paar Tipps

How To Win

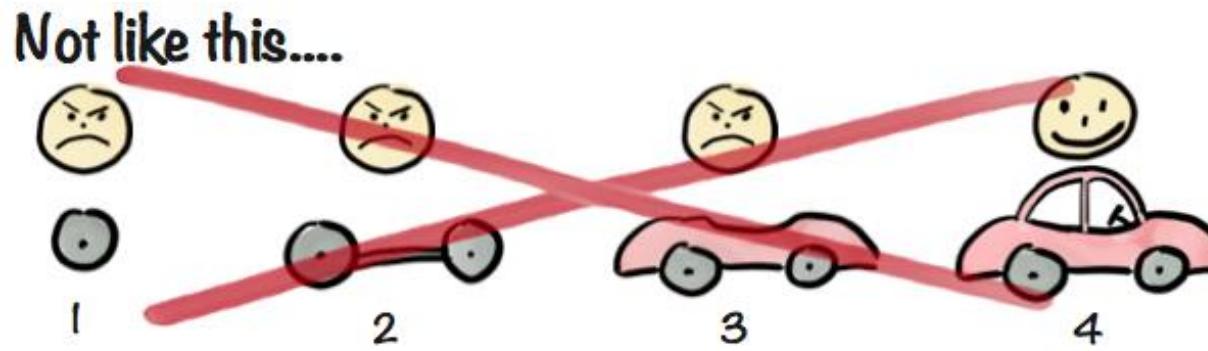
Aufgabenstellung

Für ein mittelständiges Unternehmen in der Region möchten wir eine Softwarelösung für die Inventur vom Lager gestalten. Im Backoffice soll unkompliziert über eine Webseite die aktuellen Lagerbestände verfolgt werden und informiert werden, wenn ein Artikel einen gewissen Lagerbestand unterschritten hat. Unsere Lagerarbeiter haben einen Handscanner mit entsprechender Software und scannen alle Artikel die ein und ausgehen automatisch.

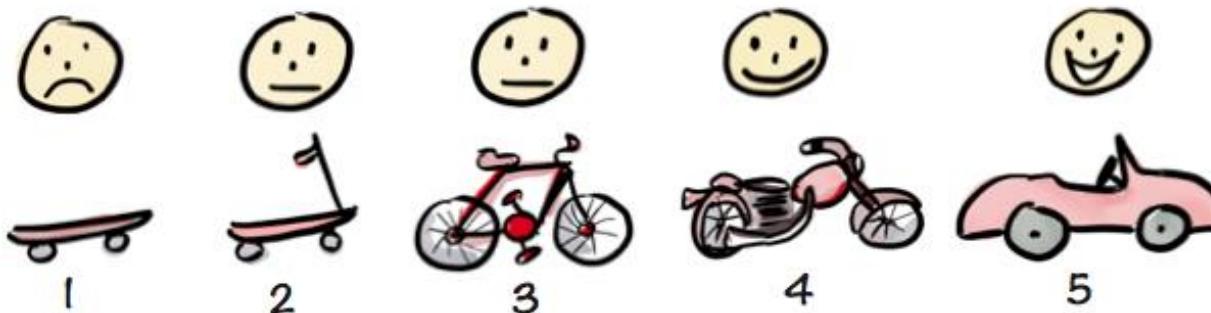
Auf jedem Artikel in unserem Lager befindet sich aus diesem Grund ein QR Code mit den wichtigsten Informationen über den Artikel. Wichtig Artikel können auch in einem bestimmten Regal liegen diese Informationen können auch gescannt werden

How To Win

Klein starten



Like this!



How To Win

Problemstellung Definieren

- Versetzen Sie sich in ihren Anwendungsfall und versuchen diesen einmal praktisch durchzugehen. Sie können sich dafür bspw. eine Anwendung auf Papier zeichnen und „durch klicken“
 - Alternativ schnappen Sie sich ein paar Kommilitonen und sprechen einmal gemeinsam über die Anwendung
- Versuchen Sie die einzelnen Problemstellungen herauszuarbeiten und welche Anforderungen Sie lösen müssen.
- Ziel: Ein Verständnis für die Anwendung bekommen und eine ungefähre Vorstellung wie diese später Funktioniert

How to Win

Datenmodell

- Als erstes überlegen Sie sich ein grobes Datenmodell für die Daten / Informationen mit denen Sie arbeiten müssen
 - Was für Informationen hat ein Datenobjekt?
 - Was für Felder können Sie ableiten?
 - Was müssen Sie speichern?
 - Gibt es Abhängigkeiten zu anderen Daten?
 - Denken Sie auch an ER Diagramme aus den Datenbank Konzepten
- Am einfachsten definieren Sie sich ein fertiges JSON und probieren damit die einzelnen Problemstellungen einmal Gedanklich durchzugehen

How To Win

Schnittstellen

- Ausgehend der Anwendungsfälle überlegen Sie sich einmal alle API Schnittstellen die Sie implementieren müssen und wie diese ungefähr aussehen
 - Welche HTTP Methoden und welche Status Codes?
 - Wie könnte eine passende URL aussehen?
 - CRUD (Create, Read, Update, Delete) Operationen für die einzelnen Ressourcen?
 - Was für Operationen kann ich ausführen?
 - Was muss mein System unterstützen?
 - Was muss ich von einem Client erhalten?
- Definieren Sie sich eine URL und eine HTTP Methode für jeden Anwendungsfall den Sie durchspielen – Achten Sie dabei darauf, was für Daten sie von A nach B schicken müssen

How To Win

Schnittstellen

- Mit einfachen Schnittstellen für ihre Ressourcen können Sie sich auch einmal überlegen ob es Anwendungsfälle gibt, wo man Filtern bzw. nach Daten suchen möchte, und überlegen Sie sich passende Query und Routen Parameter
 - Was für Query Parameter sind möglich?
 - Welche Filter Optionen bieten Sie an?
 - Welche Routen Parameter haben Sie?
 - Wie könnte eine mögliche Suche aussehen?
- Überlegen Sie sich am Anfang einfache Möglichkeiten der Filterung ohne direkt eine Google Suche zu implementieren

How To Win

User Interface

- Skizzieren Sie sich einmal, wie die Anwendung aussehen könnte. Am einfachsten ist es mit einer Homepage zu beginnen und alle CRUD Operationen abzubilden
 - Welche Ansichten gibt es?
 - Wie viele Ansichten gibt es?
 - Gibt es Elemente die immer sichtbar sind?
 - Wie stellen Sie die wichtigsten Informationen da?
 - Wie sieht das Layout aus?
- Versuchen Sie nicht direkt in Frontend Konzepte direkt zu denken sondern überlegen Sie sich einfach mal das User Interface als „Bild“

How To Win

User Interface

- Versuchen Sie anhand ihrer Skizze die ersten Components zu identifizieren, welche die Basis ihrer Anwendung bilden. Probieren Sie auch direkt alle Layouts als mögliche Components darzustellen. Programmieren Sie diese aber noch nicht aus
 - Was für Components gibt es?
 - Welche Components können wiederverwendet werden?
 - Was sind Basis Elemente für ihre Elemente?
 - Was für Layouts haben Sie?
 - Was für eine Aufgabe hat die Component?
- Hier möchten wir ein Gefühl bekommen aus wie vielen Einzelteilen ihre Anwendung nachher besteht und was für Aufgaben die Component übernimmt. Schreiben Sie sich einfach Component Name und Aufgabe als Text Beschreibung auf

How To Win

User Interfaces

- Überlegen Sie sich jetzt, woher Sie ihre Daten bekommen und welche Component was für Daten benötigt.
 - Können Sie Daten via Props / Inputs an Components verteilen?
 - Was liefert u.a. ihre Schnittstelle und was sind statische Daten?
 - Was sind Smart Components und was sind Dumb Components?
 - Was für einen State hat die einzelne Component?
- Bauen Sie ihre Components in diesem Schritt weiter auf und definieren Sie sich State und Props / Inputs sowie ob die Component Smart oder Dumb ist

How To Win

User Interface

- Anhand der User Interaktionen versuchen Sie die möglichen Routen abzuleiten, die ihre Webseite später unterstützt
 - Am einfachsten beginnen Sie mit Homepage und Detailseiten für CRUD Operationen
 - Gibt es Layout Components die auf jeder Route zum Einsatz kommen?
 - Auf welchen Components liegen ggf. Anchors zum Navigieren?
- Versetzen Sie sich in die Lage von einem Anwender und überlegen Sie sich wie Sie auf der Webseite navigieren würden

How To Win

User Interface

- Nutzen Sie Design Tools um einen ersten Webseiten Entwurf anzulegen
 - Bspw. Figma
- Denken Sie an die vorgestellten Component Prinzipien (u.a. Single Responsibility Principle)

How To Win

Projekt vorbereiten

- Setzen Sie für ihre Client und Server Anwendung die Projekte auf d.h. erstellen Sie sich jeweils Ordner für die Anwendung und installieren Sie die benötigten Abhängigkeiten
 - Bei Angular / NextJS nutzen Sie die vorgestellten Tools
 - Für Fastify setzen Sie einen einfachen Fastify Server als Boilerplate auf (siehe Vorlesung)
- Bereiten Sie eine rudimentäre Dokumentation vor wie Sie beide Anwendungen starten können und diese installieren
 - README.md
- Stellen Sie sicher, dass beide Projekte ohne eigentliche Logik starten

How To Win

Projekt vorbereiten

- Überlegen Sie sich ob Third Party Packages Ihnen einen Anwendungsfall einfacher umsetzen lassen
 - Bspw. UI Frameworks im Frontend oder spezielle Server Frameworks für spezielle Logik
- Planen Sie diese bereits vorab in Ihre Umsetzung mit ein

How To Win

Software Boilerplate

- Bereiten Sie für ihre Server Schnittstelle das Boilerplate vor d.h. definieren Sie sich ihre zuvor überlegten Router einmal ohne echte Daten zurückzugeben.
- Lagern Sie Routen in unterschiedliche Plugins (Bei Fastify) aus und bereiten sie bei Bedarf die Datenbank Anbindung vor
 - Tabellen Anlegen, Datenbank Objekt exposen als Plugin etc.
- Für Ihre Client Applikation können Sie sich erste Layout Components anlegen und das rudimentäre Routing mit Platzhalter Components abbilden

How To Win

Software Umsetzung

- Definieren Sie sich ein Statisches Datenmodell, welches ihr Server in der ersten Entwicklung zurück gibt und vom Frontend bereits genutzt werden kann
 - Statische Daten bedeutet einfach das JSON Format, welches Sie am Ende über HTTP verschicken
 - Dadurch kann die Server Schnittstelle bereits Daten liefern ohne die eigentliche Logik implementiert zu haben
- Definieren Sie sich in ihrer Client Applikation ihre API Kommunikation d.h. bauen Sie die ersten Funktionen für die HTTP Anfragen und geben bei Bedarf die Statischen Mock Daten zurück
 - Auf diesen Weg können ihre Components bereits Daten verarbeiten ohne mit der Schnittstelle zu sprechen

How To Win

Integration von Server und Client

- Bauen Sie Schritt für Schritt die Endpunkte auf und ersetzen die Statischen Daten mit den echten Aufrufen.
 - Wenn Sie vorab bereits das passende Datenmodell genutzt haben sollte die echte Kommunikation sofort funktionieren
 - Beginnen Sie mit den einfachen Endpunkten d.h. CRUD Operationen
- Prüfen Sie ob die Kommunikation zwischen Client und Server wie erwartet funktioniert
 - Haben Sie CORS enabled?
 - Passt das Datenmodell?
 - Müssen Sie am Datenmodell nochmal etwas ändern (Neue Erkenntnis?)
 - Interagiert ihre Webseite richtig mit User Interaktionen und dem Server?

How To Win

Zusammenarbeit im Team

- Verwenden Sie für die Quellcode Verwaltung GIT am besten mit einem Cloud Provider
 - Github ist für Studenten kostenlos und Repos können Sie auch privat erstellen
 - Überlegen Sie sich ob Sie mit einem Branch arbeiten oder parallel Branches pflegen
- Nutzen Sie Live Share Extension von VSCode um zeitgleich am selben Projekt zu arbeiten
 - Sie können sogar Server und Webseite über Live Share teilen, ohne das Projekt direkt zu hosten
- Arbeiten Sie iterativ – Beginnen Sie mit kleinen Meilensteinen ohne direkt alles zu lösen
 - CRUD Operationen sind meistens schnell erledigt

How To Win

Hinweise

- Arbeiten Sie gemeinsam und tauschen Sie sich untereinander aus
 - Bei Bedarf auch mal andere Gruppen Fragen wie die etwas lösen
- Beginnen Sie zuerst ohne AI-Werkzeuge und aktivieren Sie diese erst später wenn Sie ein gewisses Verständnis von ihrer Software / Anforderung haben
 - Mittlerweile sind viele Tools „dumm“ und produzieren Code der zwar nach was aussieht aber etwas komplett anderes macht
- Nutzen Sie Debugging Tools um nachzuvollziehen was ihre Anwendung in den einzelnen Use Cases genau tut für ein besseres Verständnis

How To Win

Schwierigkeiten

- State Management in Client Applikationen
- Fehlendes Verständnis wie Client und Server zusammenarbeiten
- Ungenaue API-Definitionen
- Unbekannte JavaScript Fehler
 - Tipp: Fehler lesen hilft meistens
 - Ansonsten: Fehler in ChatGPT werfen
- Seiteneffekte beim Arbeiten mit JavaScript Daten
 - Denken Sie daran, Objekte werden als Referenz übergeben und können modifiziert werden

How To Win

Hinweise

- Und falls Sie mal gar nicht weiterkommen können Sie jederzeit Fragen per Email stellen

Debugging

Fehlersuche in JavaScript / Browser

Debugging

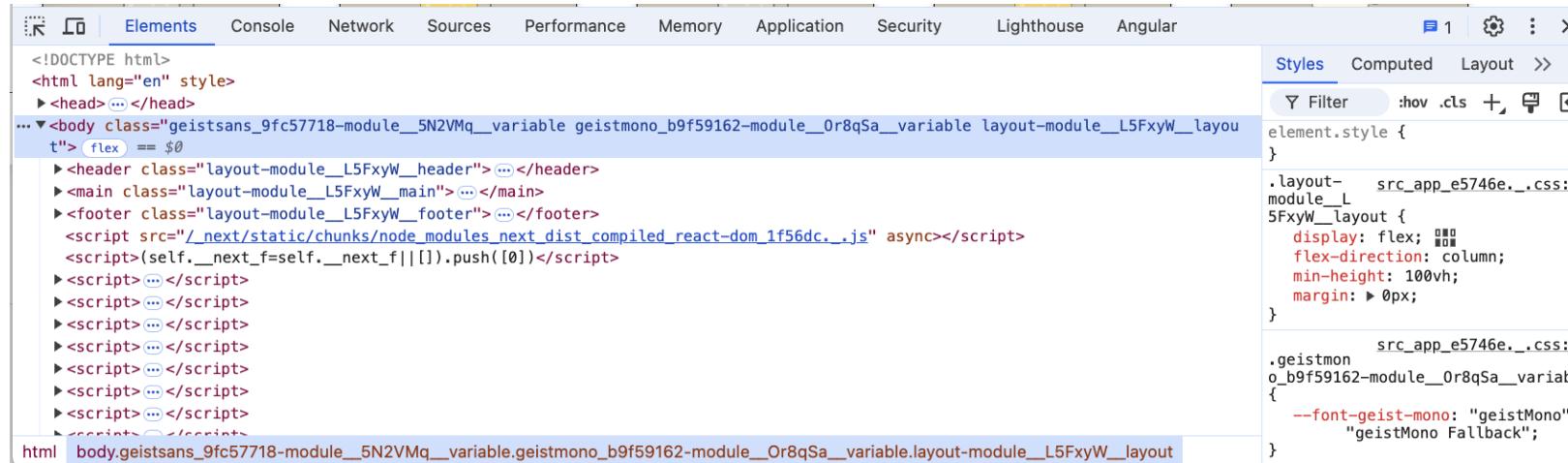
Hintergrund

- Verstehen wie Sie Entwicklerwerkzeuge Praktisch einsetzen können, ist ein fundamentaler Aspekt der Software Entwicklung
- Gerade in JavaScript ist es sinnvoll, zur Laufzeit nachzuvollziehen was ein JavaScript Programm macht
 - Denken Sie an die dynamische Typisierung von JavaScript
- Ihr Browser aber auch VSCode bietet umfangreiche Tools für das Debuggen von Web Applikationen

Debugging

Browser Developer Tools

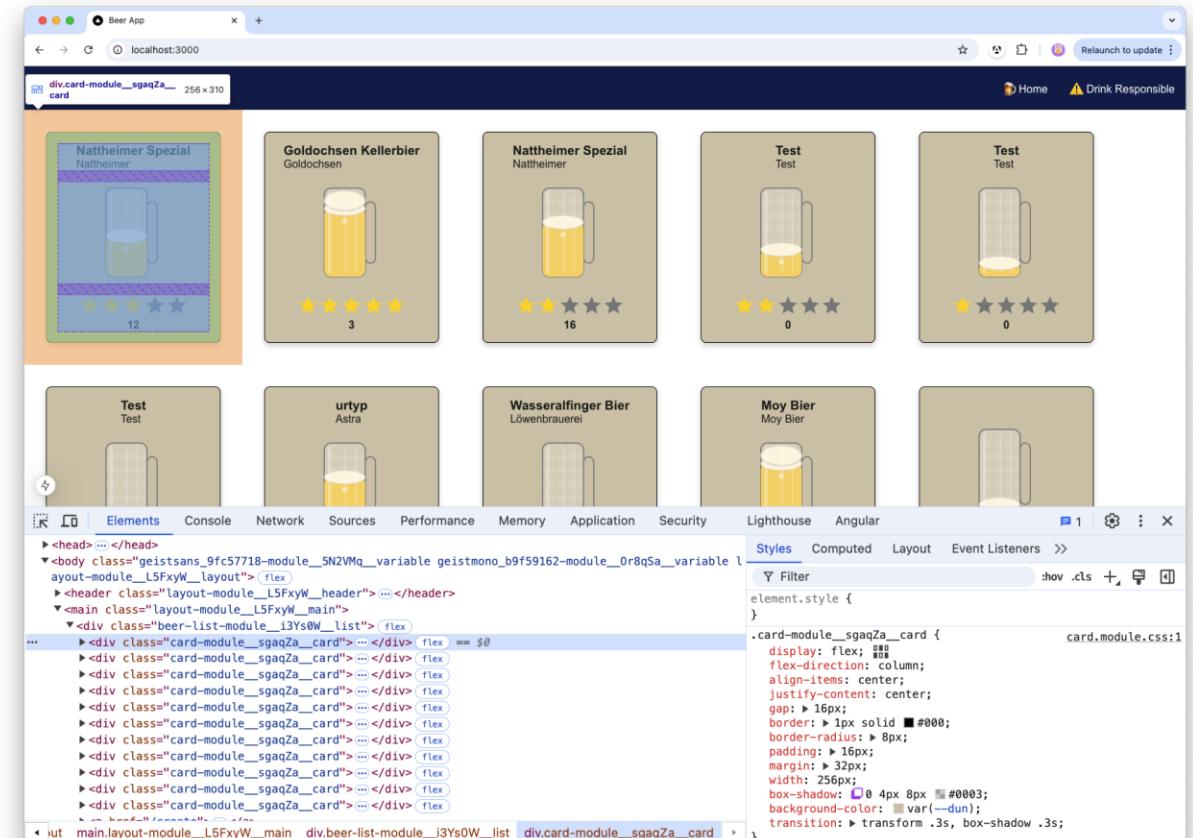
- Über F12 können Sie im Chrome die Developer Tools öffnen
- Über die Toolbar können Sie unterschiedliche Tools auswählen
 - Nachfolgend schauen wir uns die wichtigsten an



Debugging

Browser Elements Toolbar

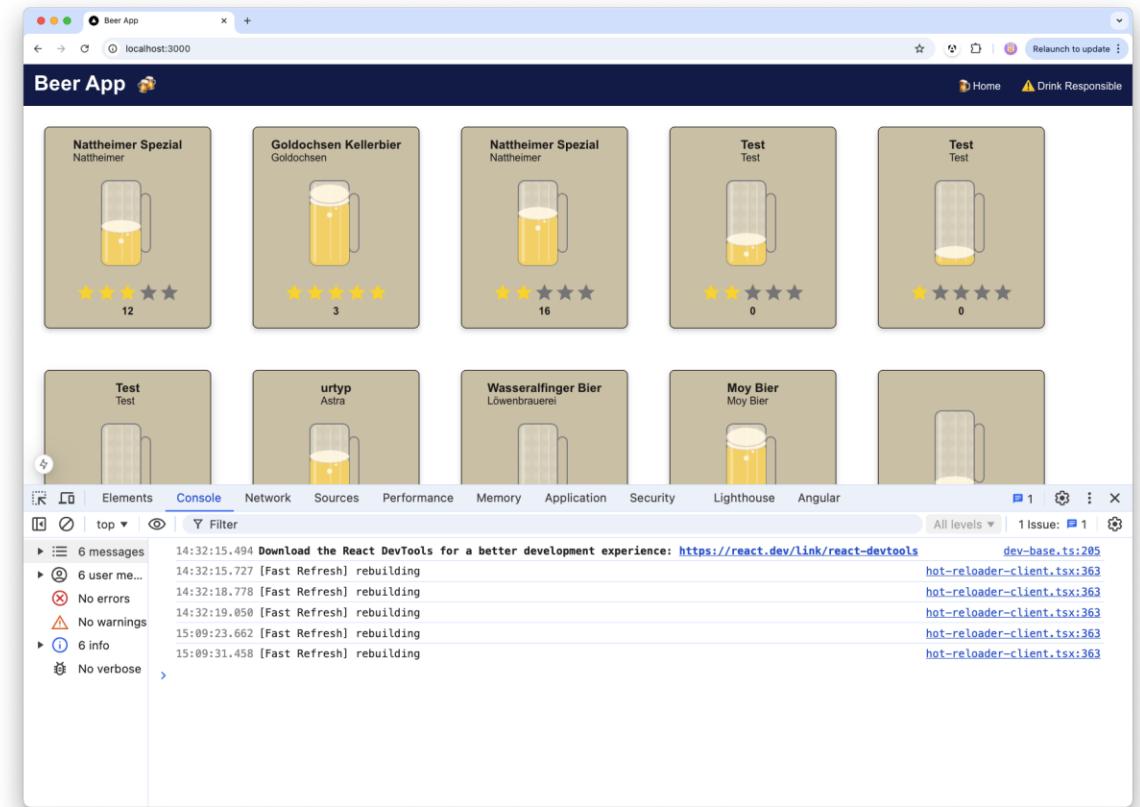
- Mit dem Elements Tool können Sie Live das HTML und CSS der aktuellen Seite bearbeiten und modifizieren
- Über den CSS Inspektor können Sie bspw. Live CSS Modifikationen ausprobieren und Abstände überprüfen
- Mit dem Selector (Links oben) können Sie auch einfach auf ein HTML Element klicken und kommen zur Implementierung



Debugging

Browser Console Toolbar

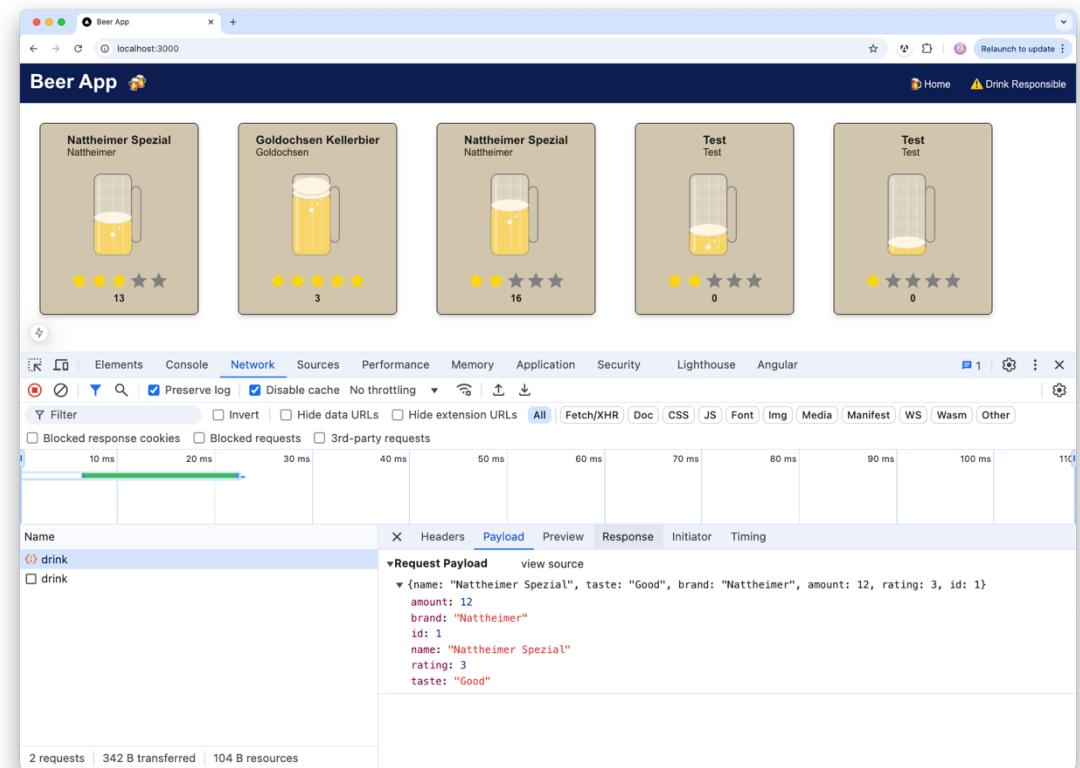
- Mit der Console können Sie direkt JavaScript auf der aktuellen Seite ausführen
- Hier tauchen auch alle ihre **console.log** Statements auf oder Fehler, welche von JavaScript geworfen wurden



Debugging

Browser Network Toolbar

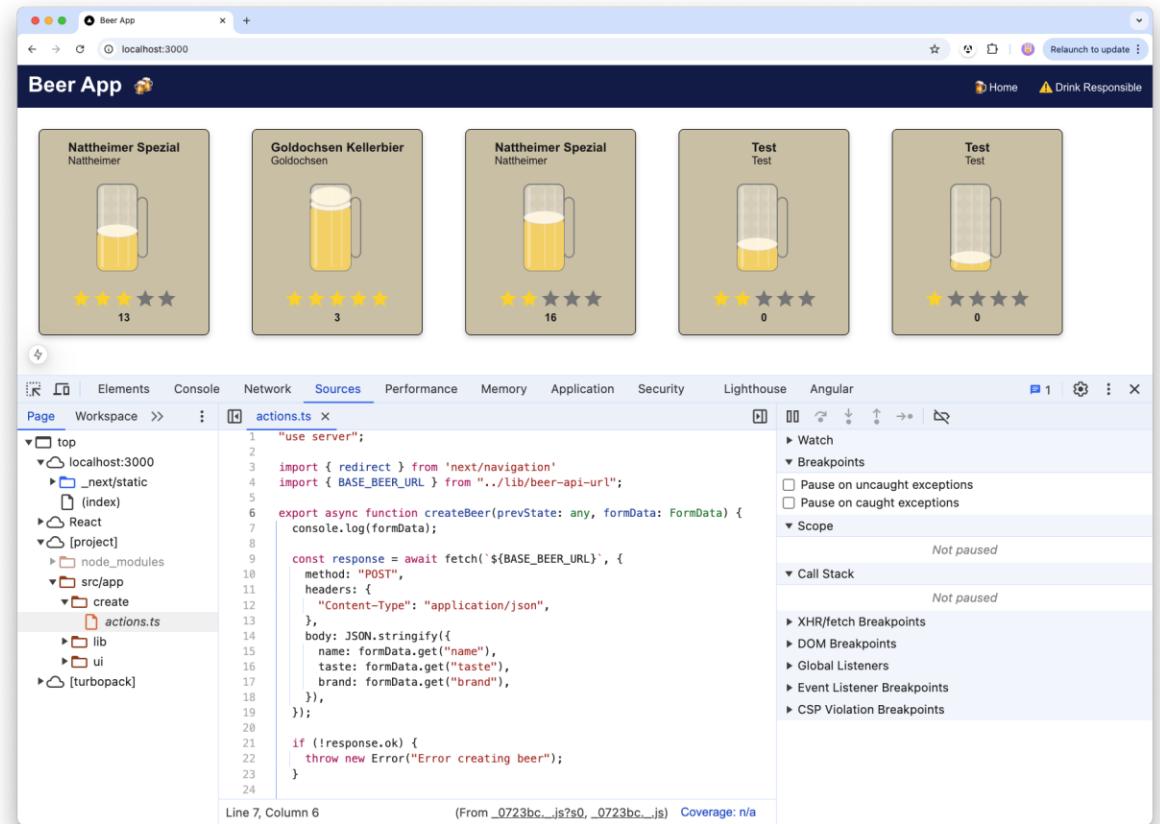
- Über das Network können Sie alle Netzwerkaktivitäten der aktuellen Webseite nachverfolgen
 - Wichtig beim ersten Mal Laden muss das Network offen sein, ansonsten werden die Anfragen nicht protokolliert
- Hier können Sie nach einzelnen HTTP Anfragen filtern und die Kommunikation zwischen Server und Client debuggen



Debugging

Browser Sources Toolbar

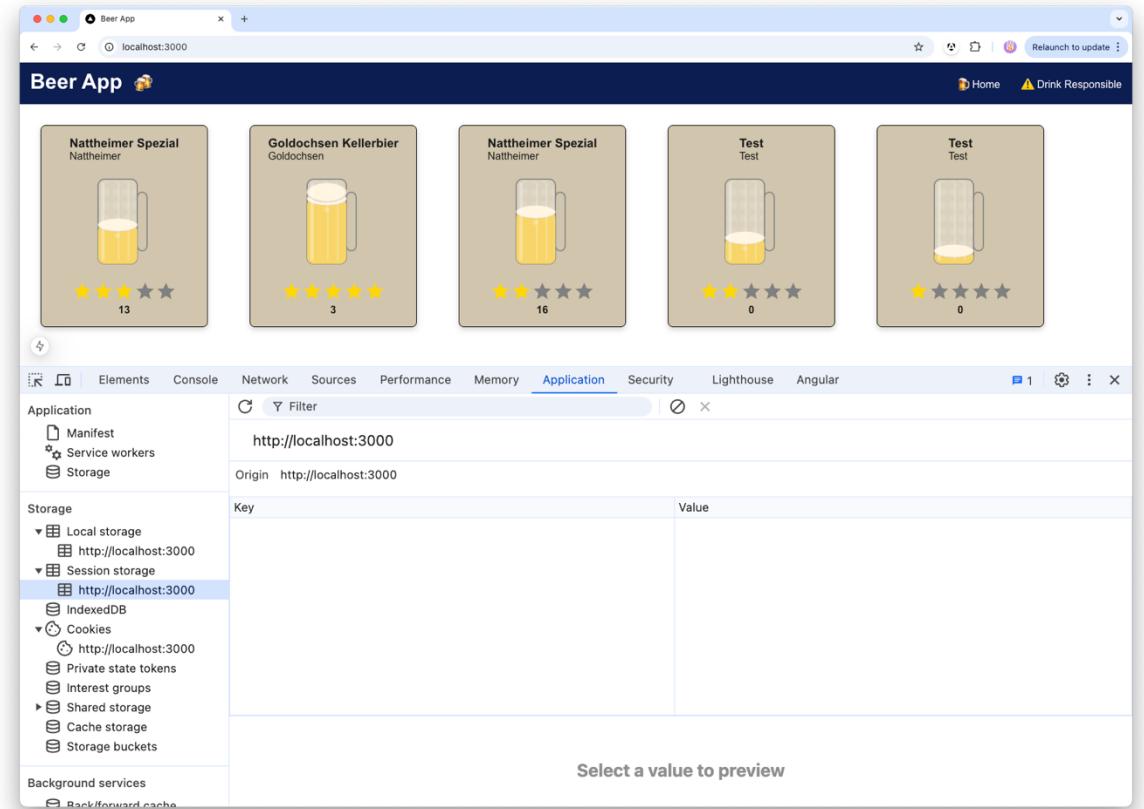
- In Sources sehen Sie den gesamten JavaScript Source Code und können u.a. Breakpoints setzen und Variablen Inspecten
- Wenn Sie ein Debugger Statement in ihrem Client Code haben wird hier auch der Prozess angehalten
 - Vgl. VSCode Debugger



Debugging

Browser Application Toolbar

- Unter Application können Sie den Webseiten Speicher auswerten und modifizieren
- Hier finden Sie u.a. Cookies und Zugriff zum Local und Session Storage



Debugging

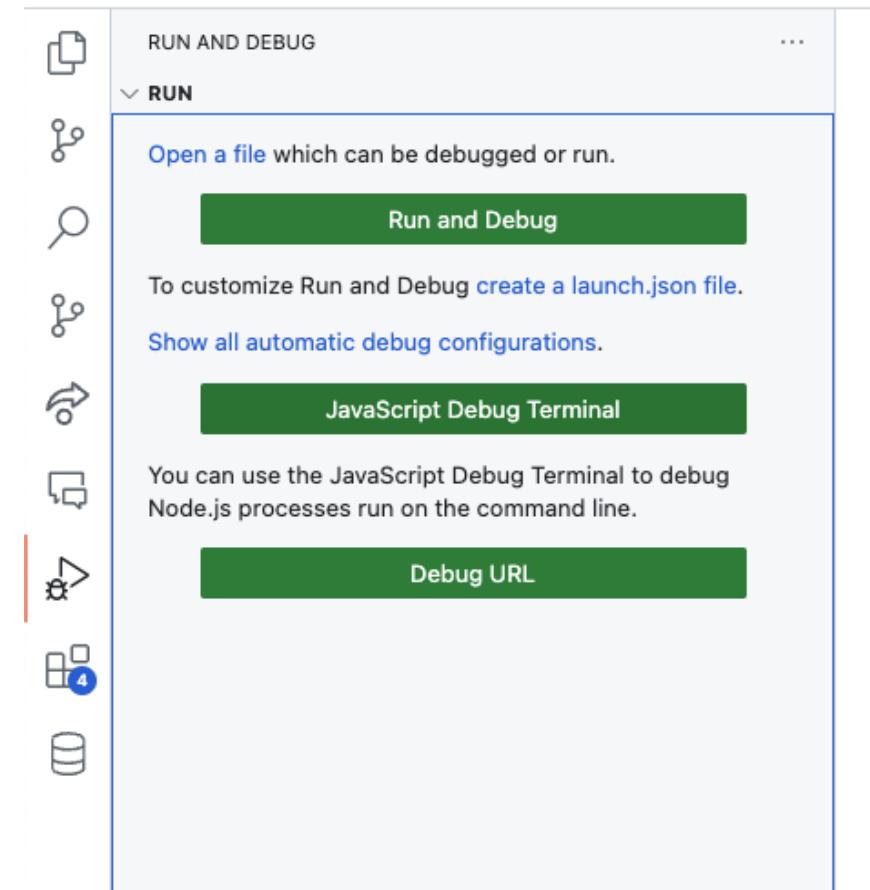
Browser Developer Tools

- Eine vollständige Erläuterung finden Sie auch unter folgendem Link
<https://developer.chrome.com/docs/devtools/>

Debugging

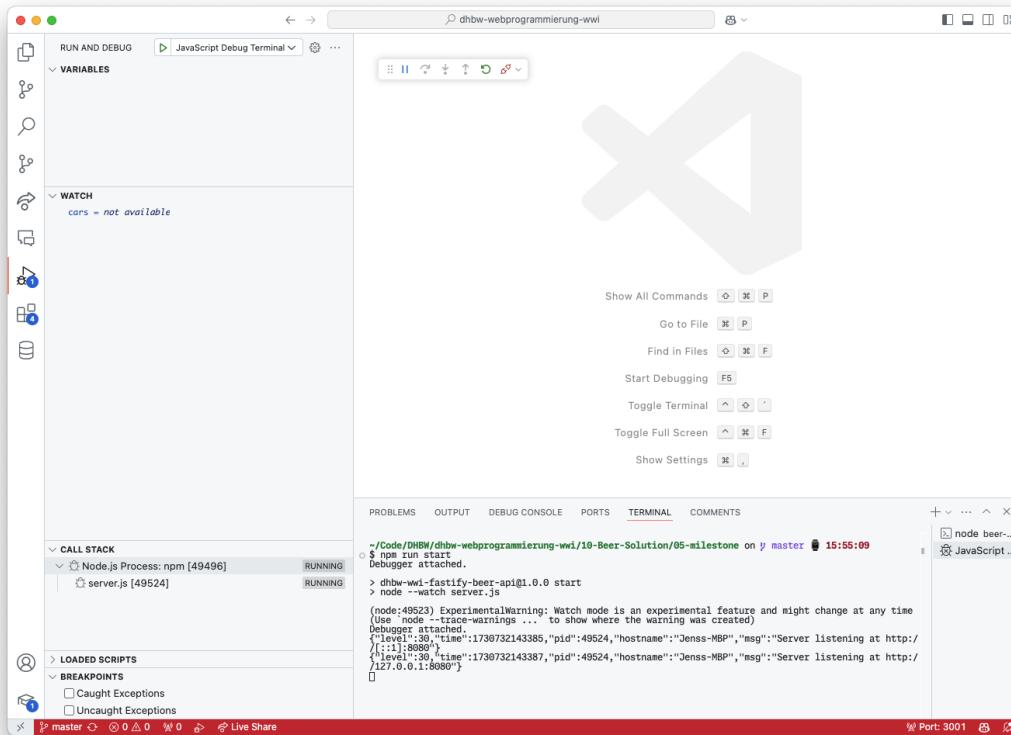
JavaScript Anwendungen

- In VSCode können Sie über das Run And Debug Menü sich ein neues JavaScript Debug Terminal öffnen
- Anschließend können Sie in das Verzeichnis navigieren, wo Sie einen Node Befehl ausführen möchten
- VSCode attached anschließend automatisch den Debugger



Debugging

JavaScript Anwendungen



Debugging

JavaScript Anwendungen

- Anschließend können Sie sich beliebige Breakpoints markieren
- Hierfür müssen Sie einfach links von der Zeilen Angabe klicken
- Wenn jetzt der Code Teil ausgeführt wird, hält VSCode Automatisch die Ausführung an

```
fastify.get("/:most", getBeerOptions, async (request, reply) => {
  const beer = getMostDrankBeer(fastify);

  if (!beer) {
    reply.code(400);
    return { error: "No beers available yet" };
  }

  return { beer: beer };
});

fastify.post("/:id/drink", drinkBeerOptions, async (request, reply) => {
  const id = parseInt(request.params.id, 10);

  const updatedBeer = drinkBeer(fastify, id);
  if (!updatedBeer) {
    reply.code(400);
    return { beer: updatedBeer };
  }

  fastify.post("/:id/rate", rateBeerOptions, async (request, reply) => {
    const rating = request.body.rating;
    const id = parseInt(request.params.id, 10);

    if (rating < 1 || rating > 5) {
      reply.code(400);
      return { beer: updatedBeer };
    }

    fastify.post("/:id/rate", rateBeerOptions, async (request, reply) => {
      const rating = request.body.rating;
      const id = parseInt(request.params.id, 10);

      if (rating < 1 || rating > 5) {
        reply.code(400);
        return { beer: updatedBeer };
      }
    });
  });
});
```

Debugging

JavaScript Anwendungen

- Sie können dann alle Variablen betrachten und Sehen alle Veränderungen
- Gleichzeitig können Sie über die Controls Schrittweise durch die Anwendung gehen

The screenshot shows a developer tools window for a JavaScript application. The main area displays the code file `beer.routes.js`. A breakpoint is set at line 82, which is highlighted with a yellow background. The code is as follows:

```
fastify.get("/:most", getBeerOptions, async (request, reply) => {
  const beer = getMostDrankBeer(fastify);

  if (!beer) {
    reply.code(400);
    return { error: "No beers available yet" };
  }

  return { beer: beer };
});

fastify.post("/:id/drink", drinkBeerOptions, async (request, reply) => {
  const id = parseInt(request.params.id, 10);

  const updatedBeer = drinkBeer(fastify, id);
  if (!updatedBeer) {
    reply.code(400);
    return { error: "Beer not found" };
  }

  return { beer: updatedBeer };
});

fastify.post("/:id/rate", rateBeerOptions, async (request, reply) => {
  const rating = request.body.rating;
  const id = parseInt(request.params.id, 10);

  if (rating < 1 || rating > 5) {
    reply.code(400);
  }
});
```

The left sidebar contains several panels:

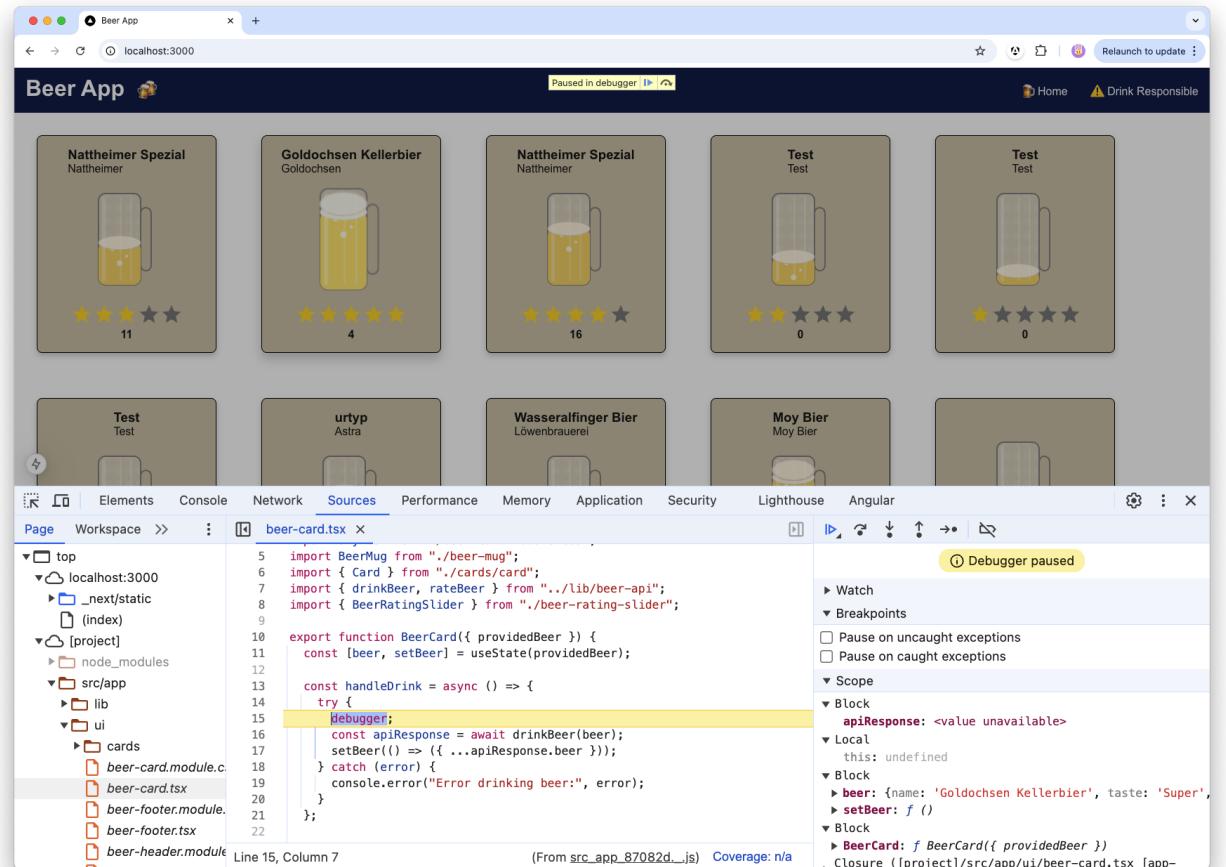
- VARIABLES**: Shows a local variable `updatedBeer` with the value `{id: 2, name: 'Goldochsen Kellerbier'}`.
- WATCH**: No items are listed.
- CALL STACK**: Shows the stack trace for the current process, with the top frame being `server.js [49524]` (Paused).
- LOADED SCRIPTS**: Shows the loaded script `beer.routes.js`.
- BREAKPOINTS**: Shows a single breakpoint at line 82.

At the bottom of the interface, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, PORTS, TERMINAL, and COMMENTS. The TERMINAL tab is active, showing the command `npm run start` and the output message "Debugger attached". The DEBUG CONSOLE tab shows log messages from the application, including experimental warnings about watch mode and trace warnings.

Debugging

JavaScript Anwendungen

- Für Clientseitige Anwendungen können Sie einfach ein **debugger** Statement an die gewünschte Stelle schreiben und im Browser die Browser Tools öffnen
- Der Browser stoppt dann automatisch, wenn dieser ein **debugger** Statement findet



Datenaustausch

Kommunikation mit HTTP und Fetch API

Datenaustausch

Hintergrund

- Wir möchten die Interaktion zwischen Client und Server einmal genauer betrachten
- Als Beispiel nutzen wir die Implementierung aus Milestone 05 für unseren Beer Server und die NextJS Anwendung für die Client Anfragen

Datenaustausch

Client Server Modell

- Für den Austausch von Daten ist es wichtig im Hinterkopf das Client Server Modell zu behalten
- Client und Server agieren komplett getrennt voneinander
- Das bedeutet in unserem Fall:
 - Der Client hat seine eigenen Daten und dessen Repräsentation
 - Der Server hat seine eigenen Daten und dessen Repräsentation

Datenaustausch

Am Beispiel der Beer App

- Unser Server ist als API Schnittstelle implementiert
- Im Hintergrund hat unser Server die komplette Logik für das Verwalten und Speichern von verschiedenen Bieren
- Mit dem Server kann aber jeder beliebiger Client (Webseite, Bruno etc.) sprechen

Datenaustausch

Am Beispiel der Beer App

- Unser Client ist als Webseite implementiert
- Unsere Webseite hat ihre eigene Logik und kommuniziert mit der Server Schnittstelle, um Daten auszutauschen
- Was die Webseite aber mit den Daten macht oder wie sie diese darstellt ist komplett der Webseite überlassen

Datenaustausch

Am Beispiel der Beer App

- Damit unsere Client Anwendung (Webseite) mit unserem Server sprechen kann verwenden wir das HTTP Protokoll für die Kommunikation über das Netzwerk
- Für das Ausführen von HTTP Anfragen im Browser gibt es die Fetch API
 - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Datenaustausch

Am Beispiel Bier Trinken

- Unser Server stellt eine Route bereit für das Trinken von einem Bier
 - Unter der URL:
<http://localhost:8080/:id/drink>
- Die Route erwartet als HTTP Methode POST

```
1 fastify.post("/:id/drink", drinkBeerOptions, async (request, reply) => {
2   const id = parseInt(request.params.id, 10);
3
4   const updatedBeer = drinkBeer(fastify, id);
5   if (!updatedBeer) {
6     reply.code(400);
7     return { error: "Beer not found" };
8   }
9
10  return { beer: updatedBeer };
11});
```

Datenaustausch

Am Beispiel Bier Trinken

- Unsere Webseite implementiert eine JavaScript Funktion, welche den Fetch Aufruf ausführt
- Weil Fetch eine Promise Operation ist (Asynchron) deklarieren wir unsere Funktion als `async`

```
1  export async function drinkBeer(beer) {
2    const response = await fetch(` ${BASE_BEER_URL} / ${beer.id} / drink ` , {
3      method: "POST",
4      headers: {
5        "Content-Type": "application/json",
6      },
7      body: JSON.stringify(beer),
8    });
9
10   if (!response.ok) {
11     throw new Error("Failed to drink beer");
12   }
13
14   const data = await response.json();
15   return data;
16 }
17
```

Datenaustausch

Am Beispiel Bier Trinken

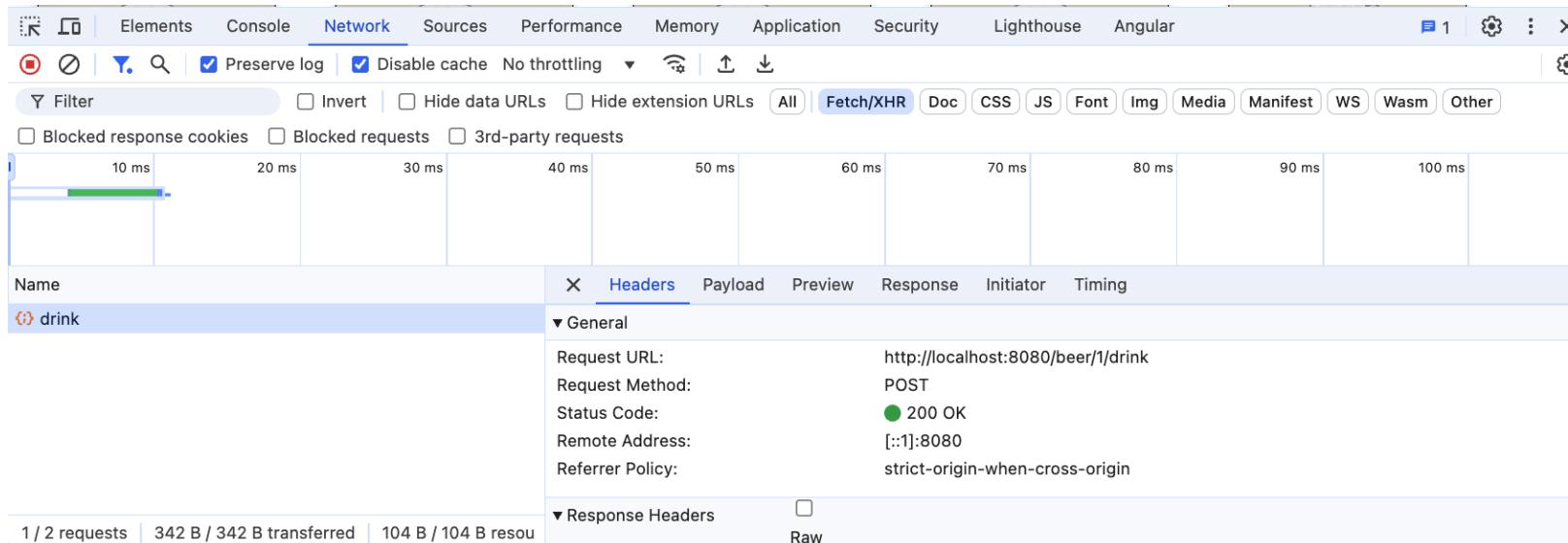
- Als User klicken wir jetzt auf der Webseite auf ein Bier, dass wir trinken möchten
- Der Browser führt einen Event Handler aus, auf diesen haben wir eine eigene Funktion gelegt im React Code
- Unser Event Handler (**handleDrink**) versucht anschließend die **drinkBeer** Funktion auszuführen

```
1 const handleDrink = async () => {
2   try {
3     const apiResponse = await drinkBeer(beer);
4     setBeer(() => ({ ...apiResponse.beer }));
5   } catch (error) {
6     console.error("Error drinking beer:", error);
7   }
8};
```

Datenaustausch

Am Beispiel Bier Trinken

- Im Hintergrund führt dann unser Browser die HTTP Anfrage an unseren Server aus
 - F12 -> Netzwerk



Datenaustausch

Am Beispiel Bier Trinken

- Der Server wiederrum nimmt unsere HTTP Anfrage entgegen und führt die eigentliche Logik aus, nämlich ein Bier zu trinken
- Als Antwort schickt der Server im HTTP Body das veränderte Bier zurück



The screenshot shows a browser's developer tools Network tab with a single request listed. The request has the following details:

- Headers:** Headers
- Payload:** (highlighted in blue)
- Preview:** Response
- Response:** Initiator
- Timing:**

Request Payload: view source

```
{name: "Nattheimer Spezial", taste: "Good", brand: "Nattheimer", amount: 11, rating: 3, id: 1}
amount: 11
brand: "Nattheimer"
id: 1
name: "Nattheimer Spezial"
rating: 3
taste: "Good"
```

Datenaustausch

Am Beispiel Bier Trinken

- Auf der Client Seite können wir uns dann überlegen, was wir mit der Server Antwort machen
- In unserem Beispiel updaten wir das aktuelle Bier mit dem neuem Bier vom Server

Datenaustausch

Hinweise

- Als Client wissen wir nicht was der Server für eine Logik im Hintergrund ausführt
 - Und das ist auch Wichtig in Bezug auf Security Aspekte und REST Prinzipien
- Als Client müssen wir aktiv werden, wenn wir eine Operation ausführen möchten, was wir mit der Antwort anschließend machen, ist wiederum uns überlassen
- Die Daten auf Client Seite sind unabhängig von den aktuellen Server Daten
 - Beim Starten lädt unser Client einmal alle Biere und bei jedem neu laden
 - Wenn im Hintergrund der Server neue Informationen hat, müssen wir aktiv diese aktualisieren

Datenaustausch

Ausblick

- Dieses Konzept der Client Server Interaktion können wir jetzt universell für jede andere HTTP Anfrage auch nutzen

Fastify Fluent Schema

Schema Definitionen mit Code programmieren

Fastify Fluent Schema

Hintergrund

- Für das Schreiben von Fastify Schema Definitionen bietet es sich an, diese mit dem offiziellen Fastify Fluent JSON Schema zu definieren
 - <https://github.com/fastify/fluent-json-schema>
- Mit einer Fluent API lassen sich Schemas auf Programmier Ebene einfach definieren und validieren
- Gleichzeitig können Sie auch auf bereits existierende Schemas zurückgreifen und diese wiederverwenden

Fastify Fluent Schema

Beispiel

```
1 const S = require("fluent-json-schema");
2
3 const ROLES = {
4   ADMIN: "ADMIN",
5   USER: "USER",
6 };
7
8 const schema = S.object()
9   .id("http://foo/user")
10  .title("My First Fluent JSON Schema")
11  .description("A simple user")
12  .prop("email", S.string().format(S.FORMATS.EMAIL).required())
13  .prop("password", S.string().minLength(8).required())
14  .prop("role", S.string().enum(Object.values(ROLES)).default(ROLES.USER))
15  .prop(
16    "birthday",
17    S.raw({ type: "string", format: "date", formatMaximum: "2020-01-01" }) // formatMaximum is an AJV custom keywords
18  )
19  .definition(
20    "address",
21    S.object()
22      .id("#address")
23      .prop("line1", S.anyOf([S.string(), S.null()])) // JSON Schema nullable
24      .prop("line2", S.string().raw({ nullable: true })) // Open API / Swagger nullable
25      .prop("country", S.string())
26      .prop("city", S.string())
27      .prop("zipcode", S.string())
28      .required(["line1", "country", "city", "zipcode"])
29  )
30  .prop("address", S.ref("#address"));
31
32 console.log(JSON.stringify(schema.valueOf(), undefined, 2));
33
```

State Management

Client Logik abbilden

State Management

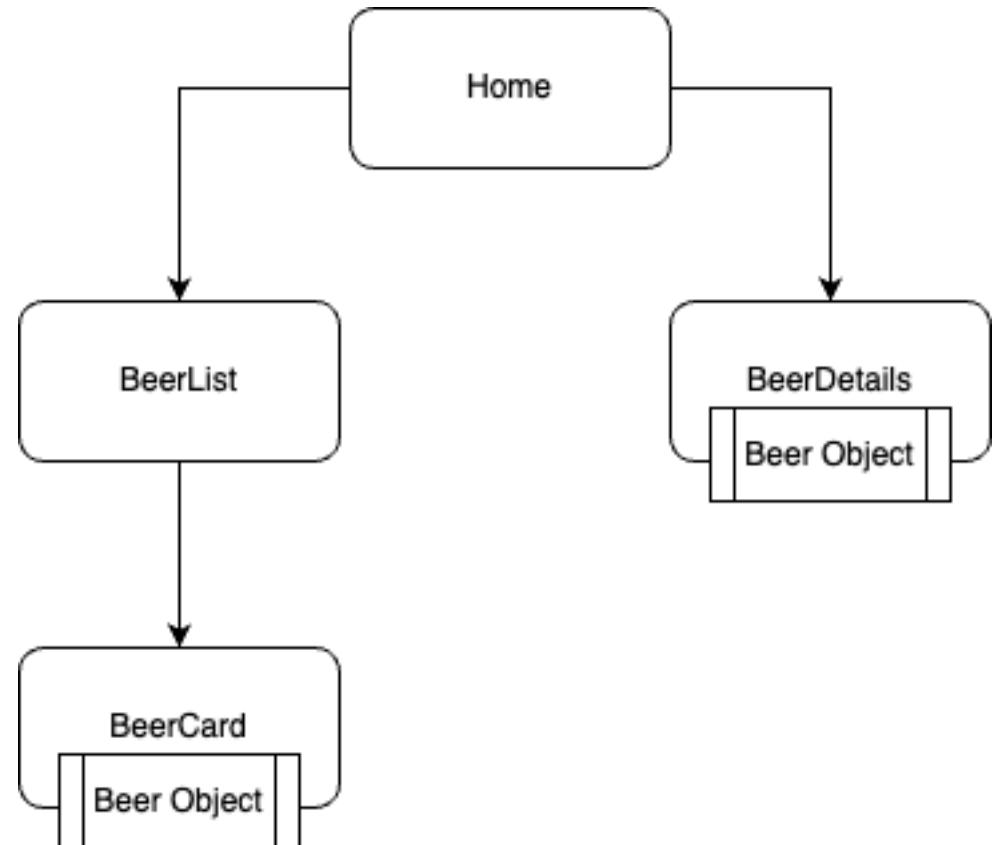
Hintergrund

- In größeren Applikationen ist es üblich, dass Sie Daten an mehreren Stellen parallel benötigen
 - Bspw. Eine Liste von allen aktuellen Bieren
 - Bspw. Die Information über ein aktuell ausgewähltes Bier
- Component basierte Frameworks sind immer Top To Bottom aufgebaut
 - Daten werden von Oben nach unten durchgereicht
- Es gibt aber Fälle, an denen Sie Daten in mehreren Components von außen benötigen

State Management

Hintergrund

- Aus diesem Grund benötigen Sie eine Möglichkeit, Daten aus anderen Ebenen an eine Component zu übergeben, ohne jedes Mal die Daten über 4-5 Ebenen durchzugeben
- Bspw. möchten wir das aktuelle Bier an verschiedenen Stellen nutzen, aber nur einmal auflösen



State Management

Technologien

React / NextJS

- In React gibt es eine spezielle Context API für das Weitergeben von Daten
 - <https://react.dev/learn/passing-data-deeply-with-context>
- Mit einem Context und einem Hook können Daten in Components gelesen werden
- Alternativ gibt es Third Party State Bibliotheken
 - <https://github.com/pmndrs/zustand>

Angular

- In Angular können Sie einfach Services in jede Component injecten und auf Variablen / Methoden vom Service jederzeit zugreifen
- Services sind dabei Singletons d.h. die Werte sind in jeder Component dieselben
- Auf diese Art können Sie also ihre komplette Logik in einem Service definieren

State Management

Hinweis

- State Management in Client Anwendungen kann sehr schnell sehr Komplex werden und gerade größere Anwendungen scheitern oft, weil das State Design schlecht gewählt ist
- State Management kümmert sich normalerweise um Globale Daten, die überall angewandt werden und u.a. Daten von APIs repräsentieren
- Es empfiehlt sich deshalb Veränderungen am State über eigene Methoden zu implementieren, um Seiten Effekte zu vermeiden
 - In Angular bspw. mit speziellen Service Methoden

State Management

Hinweise

- Versuchen Sie aus diesem Grund die meiste State Logik auf Server Seite zu implementieren und in Client Anwendungen nutzen Sie einfach nur Component Level State für unterschiedliche Darstellungen
- Die eigentliche Business Logik sollte immer über HTTP Operationen und eine Server Schnittstelle abgebildet werden
- So können Sie components sehr “einfach“ gestalten
 - Bekommt die Repräsentation der Daten vom Server
 - Bei einer Änderung wird ein HTTP Request an den Server geschickt, dieser führt die Logik aus
 - Der Server antwortet mit den neuen Daten und wir müssen nur unseren aktuellen Wert überschreiben

TypeScript

A superset of JavaScript

TypeScript

Hintergrund

- JavaScript ist dynamisch typisiert und alle Datentypen werden zur Laufzeit von JavaScript selbstständig ermittelt
- Häufige Probleme mit größeren JavaScript Anwendungen
 - Wartung und Weiterentwicklung sehr schwierig ohne gute Test Abdeckungen
 - Viele Fehler zur Laufzeit weil plötzlich ein String zu einer Number wird
 - Schwierig bestimmte Parameter in JavaScript zu dokumentieren
 - Alle Eingaben müssen geprüft werden

TypeScript

Definition

- TypeScript ist JavaScript mit einer speziellen Syntax für das Definieren von Datentypen
- TypeScript verwendet type inference und ist ein Tool in der Entwicklung von größeren JavaScript Anwendungen
- Beim kompilieren von einem TypeScript Objekt werden alle TypeScript Definitionen wieder entfernt und als Ergebnis bekommt man normales JavaScript
- TypeScript bietet auch Entwicklungsumgebungen (IDE) die Möglichkeit Vorschläge und Auto Completions anzubieten

TypeScript

Hintergrund

- Fällt Ihnen im folgenden JavaScript Programm ein Fehler auf?

```
1 function compact(arr) {
2   if (arr.length > 10) {
3     return arr.trim(0, 10);
4   }
5   return arr;
6 }
```

TypeScript

Hintergrund

- In Zeile 2 haben wir einen Schreibfehler: **orr** anstelle von **arr** im Editor wird das aber nicht als Fehler angezeigt

```
1 function compact(arr) {
2     if (orr.length > 10) {
3         return arr.trim(0, 10);
4     }
5     return arr;
6 }
```

TypeScript

Hintergrund

- Wenn wir jetzt die selbe Funktion in einer TypeScript Datei definieren, zeigt und VSCode **orr** als Fehler direkt an

The screenshot shows a code editor window for a file named '01-intro.ts'. The code contains a function that filters an array based on its length. A tooltip has appeared over the variable 'orr' in the second line of code, indicating a TypeScript error: 'Cannot find name 'orr'. ts(2304)'. The tooltip also includes 'any' as a suggestion and 'View Problem (⌃F8)' and 'Quick Fix... (⌘.)' options. The code itself is as follows:

```
ts 01-intro.ts 1, Cannot find name 'orr'. ts(2304)
any
30-miscellaneous View Problem (⌃F8) Quick Fix... (⌘.)
1  function filter(arr: string[]): string[] {
2    if (orr.length > 10) {
3      return arr.trim(0, 10);
4    }
5    return arr;
6  }
```

TypeScript

Types by Inference

- TypeScript versteht JavaScript und kann automatisch Typen erstellen bzw. ableiten
- Eine Variable die einen String zugewiesen bekommt, wird von TypeScript als String Type interpretiert
- Sollten Sie jetzt dieser Variable eine Nummer zuweisen wird TypeScript Ihnen einen Fehler anzeigen

TypeScript

Types by Inference

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

- Editor Tab:** TS 02-types-by-inference.ts 1, U X
- Code Editor Content:**

```
30-miscellaneous > 03-typescript > TS 02-types-by-inference.ts > ...
1 let car = 'Mercedes';
2 car = 25;
3
```
- Problems View:** PROBLEMS (2) OUTPUT DEBUG CONSOLE ... Filter (e.g. text, **/*.ts...) ▾
 - > TS 01-intro.ts 30-miscellaneous/03-typescript 1
 - ▽ TS 02-types-by-inference.ts 30-miscellaneous/03-typescript 1
 - 💡 Type 'number' is not assignable to type 'string'. ts(2322) [Ln 2, Col 1]

TypeScript

Umstellung von JavaScript auf TypeScript

- Der große Vorteil von TypeScript ist, dass Sie zu Beginn ganz normal ihr gewohntes JavaScript schreiben können aber TypeScript bereits automatisch Typ Sicherheit anbietet
- Sie müssen also in bestehenden Code Bases nicht erst alles anpassen und können direkt mit TypeScript starten

TypeScript

Definition von Typen

- Variablen werden von TypeScript automatisch typisiert
- TypeScript hat eine Erweiterung gegenüber JavaScript für das Beschreiben von Typen
- Typen in TypeScript können mit <variable>: <type> = <wert> deklariert werden
 - Für den Datentype können Sie bspw. alle bekannten aus JavaScript angeben

TypeScript

Typen für Objekte

- Objekte in TypeScript können mit einem Interface beschrieben werden
- Dieses Interface kann dann als eigener Datentyp verwendet werden

```
1 interface Car {  
2   name: string;  
3   brand: string;  
4   year: number;  
5   ev: boolean;  
6   affordable: boolean;  
7 }  
8
```

```
1 const eqs: Car = {  
2   name: "EQS",  
3   brand: "Mercedes",  
4   year: 2021,  
5   ev: true,  
6   affordable: false,  
7 }
```

TypeScript

Typen für Objekte

- Bedeutet umgangssprachlich gesprochen:
 - Das Objekt **eqs** ist vom Type **Car**
- Mit dem Interface können Sie alle Keys und Properties von einem JavaScript Objekt beschreiben
- Wenn Sie einen anderen Datentype für eine Property verwenden möchten, wird TypeScript einen Fehler werfen
- Gleichtes passiert, wenn Sie einen Key in einem Objekt nicht definiert haben
 - Tipp: Optionale Keys können mit einem **?** im Interface definiert werden

TypeScript

Typen für Funktionen

- Mit TypeScript ist es auch möglich Rückgabewerte für Funktionen zu definieren und Parametern einen Datentyp zuzuordnen
- Auch hier gilt wieder, wenn Sie keinen Rückgabewert explizit definieren versucht TypeScript diesen implizit selbst herauszufinden anhand vom Rückgabewert

```
1 function square(n: number): number {  
2     return n * n;  
3 }
```

TypeScript

Neue Datentypen

- TypeScript erweitert die Datentypen von JavaScript mit 4 neuen Datentypen:
 - **any** – Ein Wert kann jeden Datentyp annehmen (Schaltet TypeScript im Grunde aus)
 - **unknown** – Anwender von einer Funktion / Variable müssen Datentyp selbst definieren
 - **never** – Dieser Datentype repräsentiert etwas, was nicht eintreten kann (Taucht meistens in Fehlern auf)
 - **void** – Funktion ohne Rückgabewert gibt automatisch **undefined** zurück

TypeScript

Eigene Datentypen

- Mit TypeScript können Sie auch eigene Datentypen definieren
- Diese können bspw. auch eine Kombination aus mehreren anderen Datentypen sein
- Der **type** Ausdruck erlaubt die Definition von eigenen Datentypen

TypeScript

Eigene Datentypen

- Angenommen wir möchten die *brand* von Fahrzeugen auf drei Hersteller einschränken, welche nur gewählt werden können

The screenshot shows a code editor window with a red border. At the top left, it says "TS 05-your-own-types.ts 1, U X". Below the editor area, the file path is listed as "30-miscellaneous > 03-typescript > TS 05-your-own-types.ts > ...". The code itself is:

```
1 type Brand = 'Mercedes' | 'BMW' | 'Audi';
2
3 const car: Brand = 'Tesla';
4
```

Below the code editor, there is a "PROBLEMS" tab with a blue circle containing the number "1". To its right are tabs for "OUTPUT", "DEBUG CONSOLE", and "...". A search bar labeled "Filter (e.g. text, **/*.ts...)" is next to a magnifying glass icon. To the right of the filter are three small icons: a square with a dot, a square with a minus sign, and a vertical ellipsis.

The "PROBLEMS" section lists one error under "TS 05-your-own-types.ts 30-miscellaneous/03-typescript":

✗ TS 05-your-own-types.ts 30-miscellaneous/03-typescript 1
☒ Type '"Tesla"' is not assignable to type 'Brand'. ts(2322) [Ln 3, Col 7]

TypeScript

Eigene Datentypen

- In unserem *brand* Beispiel haben wir einen **Union** Type definiert
 - Also ein eigener Datentype der mehrere Werte annehmen kann
- Ein Union Type wird mit | beschrieben und kann bspw. auch bei Funktion Parametern verwendet werden

TypeScript

Generics

- Eine Alternative zu Union Types sind bspw. Generics
- Mit Generics können Sie Variablen als Typen angeben und diese übergeben
- Auf diese Art und Weise können Sie bspw. Objekte innerhalb von einem Array beschreiben

```
1 interface Car {  
2   name: string;  
3   brand: string;  
4   year: number;  
5   ev: boolean;  
6   affordable: boolean;  
7 }  
8  
9 type CarArray = Array<Car>;
```

TypeScript

Fazit

- TypeScript ist eine sehr mächtige Syntax Erweiterung von JavaScript und Sie können sehr komplexe Datenstrukturen über TypeScript automatisch abbilden
- Im Rahmen der Vorlesung beschränken wir uns aber auf ganz einfache TypeScript Funktionen und möchten einfach Variablen mit einem Datentyp definieren
- Alle vorgestellten Technologien in der Vorlesung kommen auch mit einer fertigen TypeScript Integration
 - Das händische aufsetzen von TypeScript ist leider gar nicht so einfach

TypeScript

Fazit

- Wenn Sie mehr über TypeScript lernen möchten empfehle ich Ihnen das Buch Total TypeScript Essentials von Matt Pocock
 - <https://www.totalthypescript.com/books/total-typescript-essentials>
 - Sie können das Gesamte Buch kostenlos Online lesen
- Alternativ hat TypeScript auch ein eigenes Handbuch [TypeScript: Handbook - The TypeScript Handbook \(typescriptlang.org\)](https://typescriptlang.org)
- Für VSCode gibt es darüber hinaus eine Extension, welche Ihnen TypeScript Funktionen direkt im Code erklärt [VSCode Extension | Total TypeScript](#)

Websockets

Echtzeitkommunikation im Browser

Websockets

Problemstellung

- Sie möchten in der Beer App automatisch alle neuen Informationen (Neues Bier, Bewertung, Anzahl getrunken) live aktualisieren, wenn auf einem anderen Gerät eine Bewertung vorgenommen wird
- Was für Probleme erkennen Sie mit den bis jetzt bekannten Konzepten?
- Wie könnte eine mögliche Implementierung mit den aktuell vorgestellten Technologien aussehen?

Websockets

Hintergrund

- In der Client Server Architektur muss der Client immer zuerst aktiv werden, um neue Daten zu bekommen
 - In unserem Fall: Der Server kennt bereits ein neues Bier aber unser Client hat noch nicht danach gefragt
- HTTP ist in sich abgeschlossen d.h. ein Request und eine Response sind immer vollständige Interaktionen

Websockets

Lösungsvorschlag

- Mit unserem aktuellen Wissen könnten wir uns folgende Lösung überlegen
- Wir definieren uns ein Zeitliches Intervall, dass bspw. jede Sekunde eine HTTP Anfrage an den Server stellt
- Mit diesem Mechanismus sieht es dann für den User so aus, als würden wir immer Daten direkt und live bekommen
- Aber wir haben keine echte Echtzeitkommunikation mit dem Server

Websockets

Definition

- Websockets sind eine Technologie für eine two-way interaktive Kommunikation zwischen einem Client (Browser) und einem Server
- Nachrichten können an einen Server geschickt werden und Antworten können ohne Polling an den Client geschickt werden
- Websockets arbeiten auf demselben Layer 7 Protokoll im OSI Model und mit dem Layer 4 TCP Protokoll

Websockets

Definition

- Mit einer Websocket Verbindung kann der Server Daten direkt an einen Client schicken, ohne dass der Client vorher eine Interaktion (Anfrage) gestartet hat
- Auf diesen Weg können bspw. Events (Neues Bier) direkt an den Client geschickt werden
- Websockets sind in allen gängigen Browser implementiert
 - Für Webserver wie NGINX und Apache HTTP gibt es auch fertige Implementierungen

Websockets

Ablauf

- Für die Verbindung hören Server auf einem standardisierten TCP Socket
 - Vgl. mit einer Routen Definition in Fastify
- Der Client muss einen Handshake mit dem Server absolvieren, um eine Websocket Verbindung aufzubauen
- Der Handshake erfolgt dabei über das bekannte HTTP Protokoll
 - Hier werden Meta Informationen zwischen Client und Server ausgetauscht

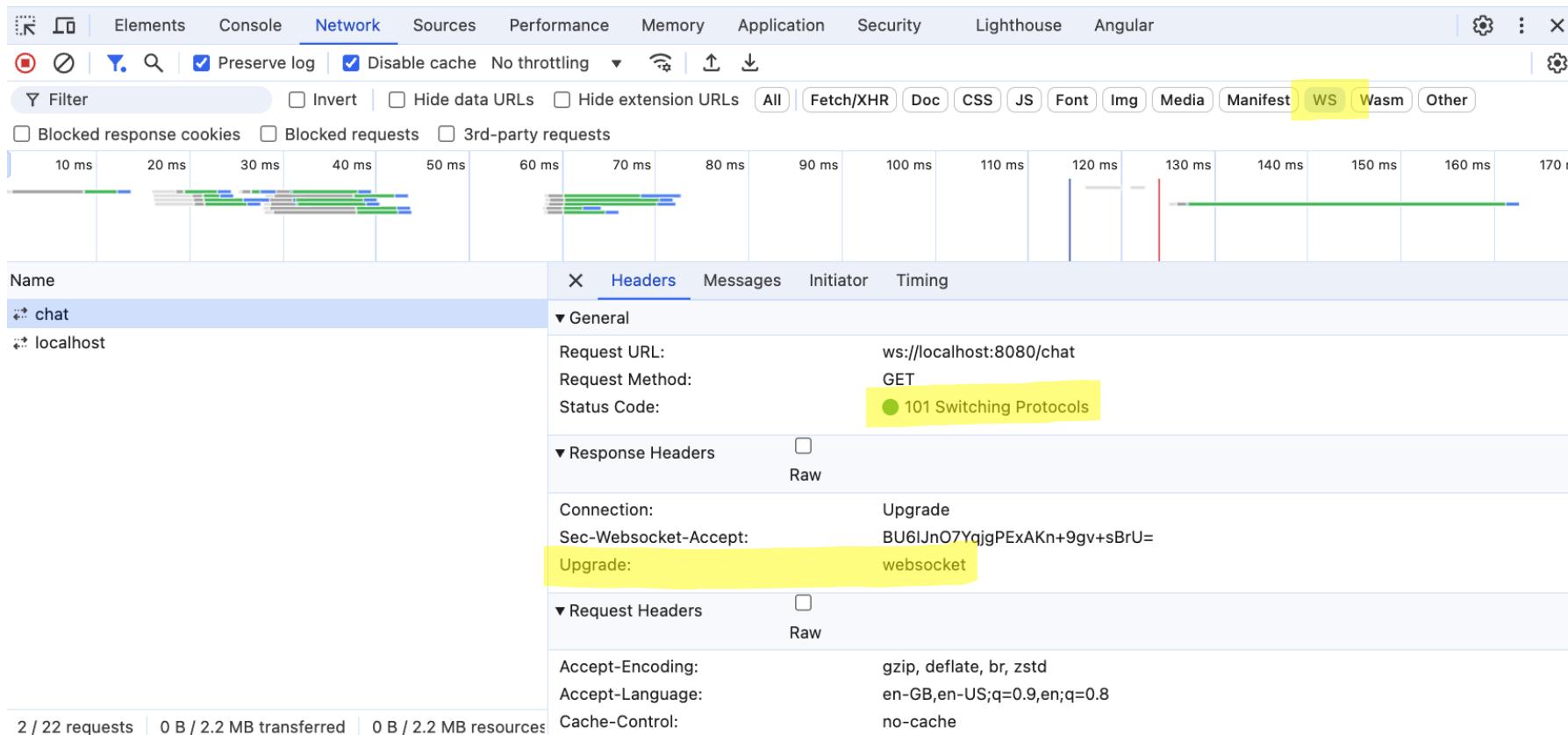
Websockets

Ablauf

- Der Client führt einen HTTP GET Request an den Server aus, um eine Websocket Verbindung aufzubauen
- Wenn der Server Websockets implementiert hat antwortet dieser mit dem Status Code 101 und wechselt das Übertragungsprotokoll
- Nach Abschluss des Handshakes können beide Seiten Daten über den Websocket senden

Websockets

Ablauf



Websockets

Ablauf

The screenshot shows a browser window titled "DHBW - Chat Client" at "localhost:4200". The page displays a "DHBW Live Chat" interface with a message log and a "Send Message" button. Below the browser is the Chrome DevTools Network tab, which is filtering for "WS" (WebSockets). The timeline shows several green and blue requests, with a red bar indicating a long-lived connection. The table below the timeline lists the messages sent between the client and the server, showing subscribe and chat messages.

Name	Initiator	Timing
↑ "chat"	All	12:52:15.542
↑ "chat"	All	12:52:15.542
↓ {"type": "users", "data": "1"}	All	12:52:15.544
↑ {"type": "chat", "data": "Hello from the Client!"}	All	12:56:09.273
↓ {"type": "chat", "data": "Hello from the Client!"}	All	12:56:09.280

Websockets

Umsetzung

- Für die Server Seite (NodeJS) gibt es verschiedene Community Packages für eine Websocket Implementierung
 - <https://github.com/websockets/ws>
- In unserem Beispiel (Fastify Server) nutzen wir das offizielle Fastify Websocket Plugin, welches ein Wrapper um WS ist
 - <https://github.com/fastify/fastify-websocket/tree/master>
- Im Browser selbst gibt es die Websocket API <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>

Websockets

Umsetzung

- In unserem Beispiel (Angular) nutzen wir die RxJS Websocket API, welches einen Wrapper für die Browser API anbietet
 - <https://rxjs.dev/api/webSocket/webSocket>
- Alternativ gibt es auch Socket IO <https://socket.io> oder von Microsoft SignalR <https://learn.microsoft.com/de-de/aspnet/signalr/overview/getting-started/introduction-to-signalr>

Websockets

Umsetzung

- Auf Github finden Sie ein fertiges Beispiel für einen einfachen Chat Server der Websockets verwendet
- <https://github.com/wasdJens/dhbw-webprogrammierung-wwi/tree/master/05-websockets/01-chat-system>