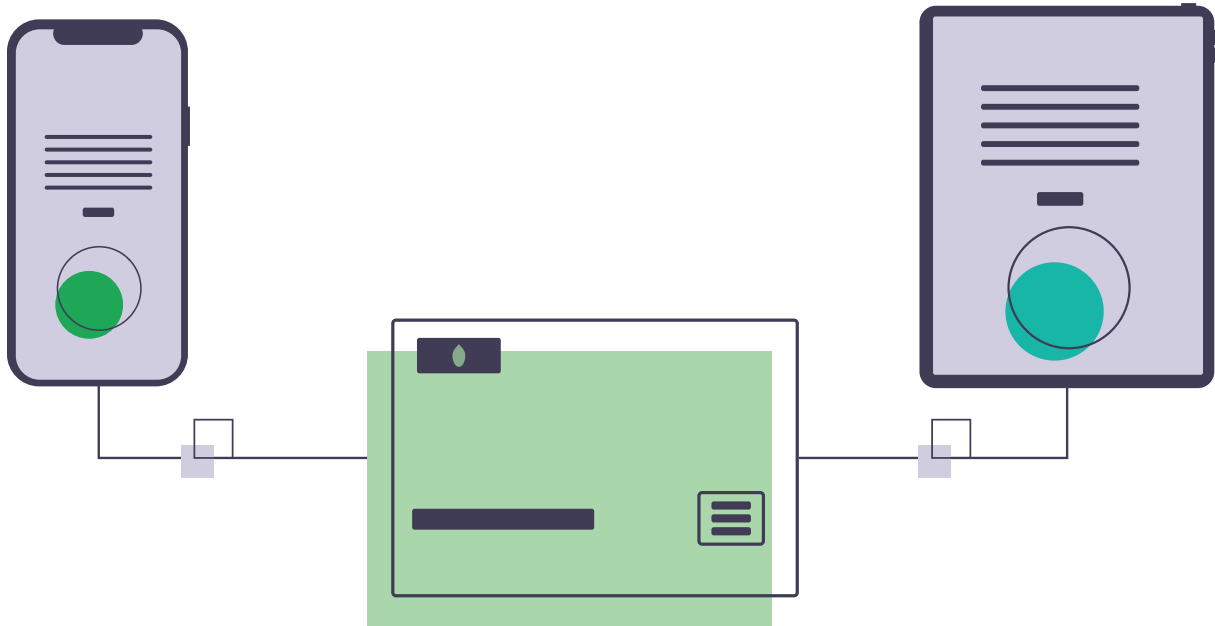


Projektarbeit

Full Stack Development mit JavaScript



Studiengang
Medien und Informationswesen
Semester
Sommersemester 2021
Betreut durch
Prof. Dr. Tom Rüdebusch

Tim Peters
185056

Eric Meyer
184888

Nils Mattmüller
184968

1.	JAVASCRIPT	3
1.1.	Entwicklung und Geschichte	3
1.2.	JS Engine	5
1.3.	Asynchroner Code	7
1.3.1.	Callbacks	7
1.3.2.	Promises	9
1.3.3.	Async/await	10
1.4.	TypeScript	11
1.5.	JSON	13
1.6.	Überblick heutiger Einsatzgebiete	16
1.6.1.	Frontend-Entwicklung	16
1.6.2.	Desktopanwendungen	16
1.6.3.	Internet of Things	17
1.6.4.	Backend-Entwicklung	18
2.	FULLSTACK DEVELOPMENT	19
2.1.	Praxisbeispiele und Motivation	19
2.2.	Client-Server-Modell	22
2.2.1.	Aufbau	22
2.2.2.	Kommunikation	23
2.3.	Clientseitige Webentwicklung	24
2.3.1.	Definition	24
2.3.2.	Frameworks	27
2.3.2.1.	React	28
2.3.2.2.	Vue	29
2.3.2.3.	Angular	32
2.3.2.4.	Vergleichender Überblick	34
2.4.	Serverseitige Webentwicklung	35
2.4.1.	Definition	35
2.4.2.	Klassische Backend-Programmierung	35
2.4.3.	Node.js	37
2.4.3.1.	Entstehung und Hintergrund	38
2.4.3.2.	Architektur	39
2.4.3.3.	Module	42
2.4.4.	Frameworks	46
2.4.4.1.	Express	46
2.4.4.2.	Socket.io	47
2.4.4.3.	KOA	48
2.4.5.	Datenbanken	49
2.4.5.1.	SQL	50
2.4.5.2.	No-SQL	54
2.5.	Architekturmodelle	55

2.5.1.	MERN	55
2.5.2.	MEAN	56
2.5.3.	PERN	56
3.	FULL STACK PRAXISPROJEKT “SKIMP”	58
3.1.	Beschreibung	58
3.2.	Anforderungsanalyse	58
3.2.1.	Recherche	58
3.2.2.	User Stories	59
3.3.	Technologie Stack	61
3.3.1.	Frontend	61
3.3.1.1.	React	63
3.3.1.2.	Erweiterungen	65
3.3.2.	Backend	66
3.3.2.1.	Node und Express	66
3.3.2.2.	MongoDB	68
3.3.2.3.	Zusätzliche Module	69
3.4.	Konzeption	70
3.4.1.	Clientseite	70
3.4.2.	Serverseite und Datenbank	71
3.4.3.	User Interface	73
3.5.	Implementierung	74
3.5.1.	Coupons hinzufügen	75
3.5.2.	Coupons abrufen	75
3.5.3.	Coupons einlösen	76
3.5.4.	Suchfunktion	77
3.6.	Deployment	78
4.	ERFAHRUNGEN UND ERGEBNISSE.....	81
4.1.	Aufgetretene Probleme	81
4.2.	Evaluation	83
5.	FAZIT	90
5.1.	Zusammenfassung.....	90
5.2.	Ausblick	92
6.	ANHANG	93
6.1.	Eigenständigkeitserklärung	93
6.2.	Abbildungsverzeichnis	94
6.3.	Quellenverzeichnis	97

1. JAVASCRIPT

Die Programmiersprache JavaScript ist mittlerweile aus nahezu jeder modernen Web-Anwendung kaum mehr wegzudenken und auch das Herzstück des Fullstack Developments.

Bevor dieses Thema im zweiten Kapitel detaillierter beleuchtet wird, soll zu Beginn ein Überblick über die zugrundeliegende Technologie geschaffen werden. So wird in den folgenden Abschnitten neben der Historie, Entwicklung und einer Zusammenfassung der vielfältigen Einsatzgebiete auch die grundlegende Architektur, Technologie und Syntax der Sprache behandelt. Aufgrund der Komplexität und Größe dieses Themas stellen die Kapitel besonders, die für die Fullstack-Entwicklung mit Node essentiellen Technologien, wie die asynchrone Programmierung in den Mittelpunkt.

1.1 Entwicklung und Geschichte

Die objektorientierte Skriptsprache **JavaScript** wurde 1995 vom Softwareunternehmen Netscape und dem Software-Hersteller Sun Microsystems zuerst unter dem Namen LiveScript veröffentlicht. Kurz darauf entschied man sich bei Netscape jedoch, den Namen in JavaScript abzuändern. Der Grund dafür war wohl, dass Java, welches zu dieser Zeit ebenfalls von Sun Microsystems entwickelt wurde, gerade sehr im Trend war und man von dessen Bekanntheit profitieren wollte. [1]

Die Ähnlichkeit der Namen sorgt bis heute für Verwirrung, jedoch haben Java und JavaScript nur wenige Gemeinsamkeiten. Während bei Java die Vererbung klassenbasiert ist, nutzt JavaScript eine prototypenbasierte Vererbung. [1] JavaScript sollte kein Konkurrent für Java sein, sondern eine simplere und somit zugänglichere Alternative für Entwickler. Der Name des Sprachkerns von JavaScript ist **ECMA Script (ES)**. Er entstand durch die Standardisierung durch die Normungsorganisation ECMA. [2]

Ursprünglich war die Skriptsprache dafür gedacht kurze Abläufe zu automatisieren, welche lokal auf dem Browser verarbeitet werden. Dadurch ergab sich die Möglichkeit statisches HTML auf dem Webbrowser durch dynamische Inhalte zu erweitern. JavaScript war als eine einfach zugängliche Sprache konzipiert, deshalb sollte die Entwicklung von Anwendungen möglichst schnell und unkompliziert sein.

Für kleine Applikationen eignete sich JavaScript deshalb ideal, weil hier die schlechte Performance, welche sich unter anderem durch die Verwendung eines Interpreters und einer dynamischen Typisierung ergab, kein Problem darstellte. Netscape integrierte JavaScript in ihren Webbrowser Netscape Navigator. Anwendungsbeispiele sind vor allem Formularvalidierungen oder auch kleine Animationen. [3] Im selben Jahr veröffentlichte Microsoft mit **JScript** für ihren Internet Explorer Version 3.0 eine eigene JavaScript-Engine, um mit Netscape zu konkurrieren. JScript baute ebenfalls auf ECMA Script auf, erweiterte JavaScript jedoch um einige Features.

Im Jahre 2005, 6 Jahre nach dem Erscheinen der neuesten ECMA Script Version 3.0, wurde das **Ajax** (*Asynchronous JavaScript and Xml*) Konzept veröffentlicht, welches es Webanwendungen ermöglicht, nur kleine Datenportionen mit den Webservern auszutauschen. Vorher musste nach jeder Anfrage an den Server ein komplett neues HTML-Dokument geladen werden. Mit Ajax war es von nun an also möglich nur die gerade benötigten Daten nachzuladen und ausgewählte Änderungen dem Server mitzuteilen. Es entstanden dadurch dynamische Interfaces im Browser, ganz ohne Plugins und Server-Roundtrips. [4] Ab diesem Moment wuchs die Verbreitung von JavaScript immer weiter an. Etliche JavaScript Bibliotheken wurden erstellt darunter JQuery, Prototype, Dojo Toolkit und MojoTools. Ebenso sind Open-Source Bibliotheken mitsamt großer Communities entstanden, die den Fortschritt weiter vorantreiben. [5]

Nachdem geplante Änderungen für ECMA-Script V4 auf Ablehnung trafen, wurde deren Veröffentlichung verworfen und erst mit Version 5.0 viele Neuerungen eingebracht. Ziel von ECMAScript 5 waren unter anderem Fehlerkorrekturen, ein *“strict mode“* für eine erweiterte Fehlerüberprüfung, die Gewährleistung einer Abwärtskompatibilität, sowie eine Unterstützung für das Datenformat **JSON**. [2]

Mit ECMAScript 6 auch bekannt unter dem Namen ES6 Harmony, fertiggestellt im Jahre 2015, folgte eine Grundlegende Überarbeitung von JavaScript. Dazu gehört eine Erweiterung der Syntax durch die Einführung der Schlüsselwörter `let` und `const`, um Variablen einfacher deklarieren zu können. [6]

Bei der clientseitigen Programmierung ist zu beachten, dass die Webseite abwärtskompatibel mit älteren ECMAScript Versionen sein sollte. Denn es braucht eine gewisse Zeit, bis alle Browser (*Engines*) die Erneuerungen unterstützen und alle Endnutzer die neueste Version des Browsers haben. Um dies sicherzustellen gibt es Software, die JavaScript-Code der aktuellen Syntax in die Syntax älterer Versionen konvertiert.

Die Laufzeitumgebung **Node.js** wurde im Jahr 2009 veröffentlicht, welche in der 2008 erschienenen JavaScript-Engine V8 ausgeführt wird. Sie ermöglichte es, JavaScript auch

serverseitig auszuführen. Über die Jahre wurde JavaScript immer beliebter und ist mittlerweile die meistgenutzte Programmiersprache für die Webentwicklung. Für größere Anwendungen wurde nun also nicht mehr Java, sondern ausschließlich JavaScript verwendet. Die schlechte Performance modernerer Anwendungen führte jedoch dazu, dass Änderungen an der Engine erforderlich waren. [3]

1.2 JavaScript-Engine

Die Hauptaufgabe einer JavaScript-Engine ist die Konvertierung von JavaScript-Code in schnellen, optimierten Code, den Browser und Webanwendungen dann interpretieren können. [7] Ziel ist es die bekannten Performance-Probleme von JavaScript auszugleichen. Dies kann man durch einen **Compiler** erreichen, welcher aus dem JavaScript-Quellcode optimierten Maschinencode erzeugt. Jedoch benötigt der Compiler etwas Zeit um die Anwendung zu kompilieren, was zur Folge hätte, dass ein Benutzer beim Öffnen der Website etwas warten müsste. Je nach Anwendungsfall würde das einige Sekunden dauern bis also eine Seite vollständig geladen wurde.

Da die Browserhersteller jedoch in starker Konkurrenz zueinander stehen und ein Nutzer natürlich großen Wert auf die Geschwindigkeit legt, ist jede Sekunde ausschlaggebend. Eine andere Möglichkeit ist ein **Interpreter**, der Interpreter führt den Quellcode direkt während der Laufzeit aus. Nach dem Parsen des Quelltextes kann sofort mit der Ausführung gestartet werden. Nachteil eines Interpreters ist jedoch die geringe Ausführungsgeschwindigkeit des Codes. [3]

Die Lösung besteht darin, Interpreter und Compiler zusammen zu verwenden. Zuerst wird der JavaScript-Quellcode in Bytecode umgewandelt. Der Bytecode interpretiert die Befehle des Quellcodes und dient als eine Art Zwischencode, bevor dieser dann entweder vom Interpreter direkt ausgeführt wird, oder vom Compiler zusammen mit Profiling-Daten, die während der Ausführung des Codes gesammelt werden, in optimierten Maschinencode kompiliert wird. Welche Teile des Codes optimiert werden, hängt davon ab, wie oft der Interpreter bestimmte Codeteile ausführt. Der Compiler greift also nur ein, wenn er zur Kompilierung solcher Code-Teile benötigt wird, was auch die Bezeichnung *"Just-in-Time Compilation"* erklärt. [3]

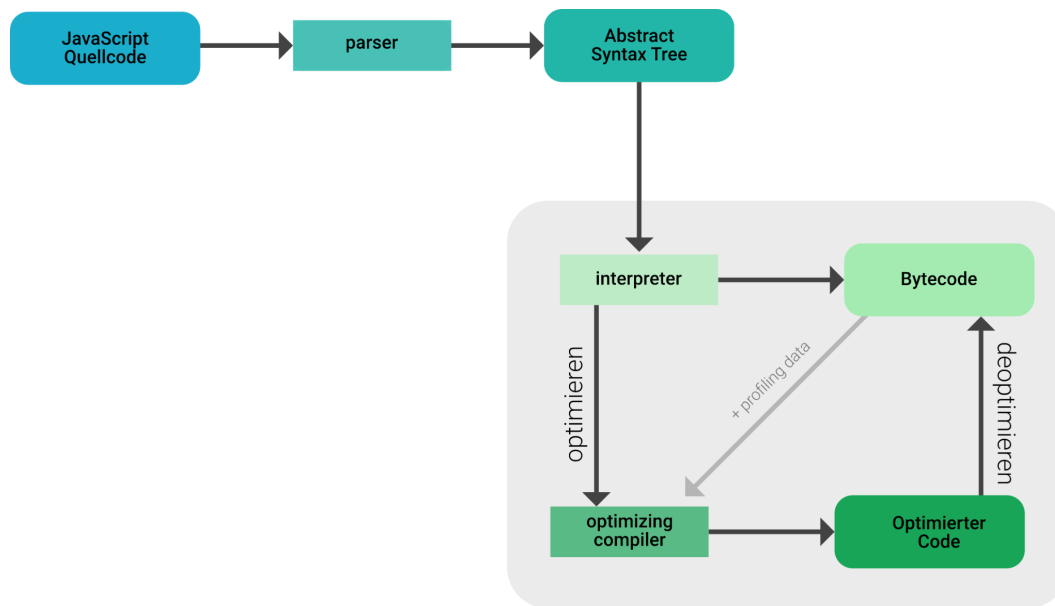


Abbildung 1: Vorgang des Compiling und der Ausführung von JavaScript

Die **dynamische Typisierung** in JavaScript ermittelt im Gegensatz zur statischen Typisierung erst zur Laufzeit den Typ, für die Entwickler entsteht so weniger Quellcode, jedoch führt dies auch zu geringerer Leistung, da der Interpreter während der Laufzeit immer wieder überprüfen muss, um welchen Typ es sich handelt. Außerdem könnte sich der Typ zur Laufzeit ändern, wodurch die Überprüfung wiederholt werden muss. Im Vergleich zur statischen Typisierung dauert dies länger, da hier die Typen unverändert bleiben. Um diese Schwachstelle auszugleichen, verwenden Engines Inline Caches und Shapes. [3] Bei der dynamischen Typisierung in JavaScript-Engines nutzt man Shapes von Objekten, um die Suche nach den Property-Informationen zu reduzieren. Zwei Objekte haben dieselbe Form, also Shape, wenn diese denselben Property-Namen in einem JavaScript-Programm besitzen. Informationen über Shape und Properties müssen nur einmal gespeichert werden, unabhängig davon, wie viele Objekte es mit derselben Form gibt.

```

1
2  const student1 = {vorname: "Eric", nachname: "Meyer", matrNr: 185064};
3  const student2 = {vorname: "Tim", nachname: "Peters", matrNr: 185056};
4  const student3 = {vorname: "Nils", nachname: "Matthüller", matrNr: 185233};
5

```

Abbildung 2: Objektsyntax in JavaScript

Sogenannte Inline-Caches (IC) verwenden die Shape von Objekten um die Anzahl von Nachschlagevorgängen zu reduzieren. Der IC merkt sich die Shape sowie den Offset an dem die Property gefunden wurde. Anstatt für jedes Objekt die Properties suchen zu müssen, muss immer

nur die Shape des Objektes verglichen werden, wenn diese übereinstimmen können die Werte direkt mit dem gemerkten Offset geladen werden. Mithilfe von Inline-Caches fällt für die JavaScript-Engine somit die Suche nach Property-Informationen weg, wodurch die Performance, vor allem von häufig auftretenden Operationen, erhöht wird. Falls eine Shape auftreten sollte, die nicht im Inline-Cache existiert, findet eine Deoptimierung statt, bei der die Ausführung wieder zum Interpreter zurückkehrt, wo neue Profiling-Daten gespeichert werden, um den Code erneut zu optimieren. [7]

Die derzeit bekannteste JavaScript-Engine ist V8 von Google Chrome, welche sich seit 2006 in Entwicklung befindet. Neben der Verwendung im Browser, wird sie auch in der JavaScript-Laufzeitumgebung Node.js verwendet. Der Interpreter in V8 heißt Ignition, der optimierende Compiler heißt Turbofan. Die Shapes tragen hier den Namen Maps. Im Gegensatz zu V8 nutzt die Engine Spidermonkey von Mozilla zwei Compiler anstatt einem. [7]

1.3 Asynchroner Code

Unter dem grundlegenden Begriff der synchronen Programmierung ist zu verstehen, dass Operationen wie beispielsweise eine Berechnung solange ausgeführt werden bis das Ergebnis vorliegt. Da jede Aufgabe im synchronen Code auf das Ende der vorherigen wartet, kann dessen Ausführung, besonders bei zeitintensiven Operationen, verzögert werden.

Die Idee der asynchronen Programmierung besteht dabei darin, dass sowohl Anfang als auch Ende einer Task zu unterschiedlichen Zeitpunkten bei der Ausführung des Programms erfolgen und daher dies nicht durch aufwendige Berechnungen länger unterbrochen wird. Auch JavaScript bietet die Möglichkeit asynchronen Code auf verschiedene Arten zu implementieren.

1.3.1 Callbacks

In JavaScript werden Funktionen als *“Bürger erster Klasse”* behandelt. Darunter versteht man, dass sich Funktionen identisch wie Variablen und Daten behandeln lassen und daher auch ähnlich wie Zahlen, Arrays oder Strings als Parameter oder Rückgabewerte in anderen Funktionen übergeben lassen. [8]

```

1
2   let numbers = [1,2,3,4,5];
3
4   numbers.forEach(function(number){
5       |   console.log(number);
6   });
7
8   console.log("Done");
9

```

Abbildung 3: Syntax einer Callback-Funktion in JavaScript

Das Grundprinzip von **Callbacks** soll in diesem Beispiel anhand des Abrufs aller Elemente eines Arrays (*numbers*) verdeutlicht werden. Der Aufruf der “*forEach*”-Funktion erwartet hierbei eine weitere Funktion als Parameter die für jedes einzelne Element des Arrays aufgerufen wird.

Typischerweise werden Callback-Funktionen meist als sogenannte anonyme Funktionen ohne Namen notiert. Oftmals werden diese nach neuerem ECMAScript-Standard auch als Lambda-Ausdrücke oder sogenannte Arrow-Functions geschrieben um den Code etwas übersichtlicher zu halten (siehe unten).

Gelegentlich stößt man auf die Auffassung, dass Callbacks asynchronen Code ausführen. Betrachtet man jedoch die Kommandozeile, nach dem Aufruf dieser Applikation, so kann man erkennen, dass die “*forEach*”-Funktion synchron arbeitet und “*Done*” erst nach Aufruf aller Zahlen des Arrays ausgegeben wird.

```

1
2
3
4
5
Done

```

Abbildung 4: Ergebnis der Ausführung der synchronen Funktion

Dennoch werden Callbacks durchaus genutzt um asynchronen Code zu schreiben, wie das folgende Beispiel verdeutlicht.

```

1
2   setTimeout(()=>{
3       |   console.log("Welt");
4   }, 3*1000);
5
6   console.log("Hallo");
7

```

Abbildung 5: Syntax einer asynchronen Callback-Funktion in der Lambda-Schreibweise

Die `setTimeout`-Funktion erwartet als Parameter zum einen eine Funktion und zum anderen einen Wert, um wie viele Millisekunden verzögert die übergebene Funktion ausgeführt werden soll. Die Besonderheit dabei ist, dass der Single Thread der Anwendung nicht blockiert wird, sondern nach dem Aufruf von `setTimeout` direkt zurückkehrt und mit der nächsten Anweisung fortfährt. Läuft anschließend der gesetzte Timer ab, wird der Callback ausgeführt.

```
PS C:\Users\timpe\Desktop\Node.js\Test> node app.js
Hallo
Welt
```

Abbildung 6: Ergebnis der Ausführung der asynchronen Funktion

Obwohl `setTimeout` vor dem Aufruf der Ausgabe von `"Hallo"` erfolgt, wird daher in der Kommandozeile zuerst `"Hallo"` und anschließend `"Welt"` geschrieben. [9]

1.3.2 Promises

Für einige Zeit waren Callback-Aufrufe das beliebteste Konzept für das Schreiben von asynchronem Programmcode. Wenn aufwendige und komplexe Anwendungen sehr viele und teilweise weit ineinander verschachtelte Callbacks enthalten, kann sich dies auf die Kosten der Lesbarkeit des Programmcodes auswirken, der zunehmend immer unübersichtlicher wird. Aus diesem Grund nutzen Entwickler heutzutage meist keine Callbacks mehr für das Handling von asynchronen Operationen. Stattdessen lassen sich Promises im Zusammenspiel mit den Schlüsselwörtern `"async"` und `"await"` verwenden, um so die Syntax des Codes schlank zu halten. [10]

Bei einem **Promise** handelt es sich um ein zurückgegebenes Objekt, an welches Callback-Funktionen angehängt werden können, anstatt diese direkt einer Funktion zu übergeben. Dieses Objekt stellt dabei einen Proxy-Wert für ein Ergebnis dar, dass man zu einem früheren Zeitpunkt noch gar nicht hat. [11]

Bei dem Aufruf eines Promise hat das Objekt den Zustand `"pending"`. Dies signalisiert, dass die aufrufenden Funktion weiterhin ausgeführt wird, bis das Promise aufgelöst wird und der Funktion die angeforderten Daten zur Verfügung stellt. Nach Auflösen des Promises ändert dies seinen Zustand, je nachdem ob die entsprechende Operation erfolgreich war, zu `"fulfilled"` oder `"rejected"` und die zuvor an `"then"` oder `"catch"` übergebenen Callback-Funktionen werden aufgerufen. [12]

```

1
2 let promise = new Promise((resolve, reject)=> { //Promise Constructor
3   |   setTimeout(()=> resolve("Hallo Welt"), 1000); //Executor
4   | });
5

```

Abbildung 7: Erstellen und auflösen eines Promise-Objekt

Wird ein neues Promise erstellt, wird die übergebene Funktion, auch Executor genannt, automatisch aufgerufen. Diese erhält zwei von JavaScript bereitgestellte Funktionen als Argumente. Schließt der Executor seine Operation ab, ruft er hier bei erfolgreicher Beendigung “resolve” auf, “reject” wird bei aufgetretenen Fehlern genutzt.

In diesem Beispiel ruft der Executor nach einer Sekunde Verarbeitung “resolve” mit dem Wert “Hallo Welt” auf. Dies ändert entsprechend den Status des Promise-Objekts.

```

5
6 promise.then((result)=> { //Promise resolved succesfully
7   |   console.log(result);
8   | });
9

```

Abbildung 8: Callback-Funktion wird an Promise gebunden

Mithilfe der “*.then*”-Funktion lässt sich nun ein Callback an das Promise binden, der aufgerufen wird, wenn das Promise aufgelöst wurde und in dem die, durch die “*resolve*”-Funktion bereitgestellten Daten, verarbeitet werden können. Diese werden in diesem Beispiel in der Konsole ausgegeben.

new Promise (executor)

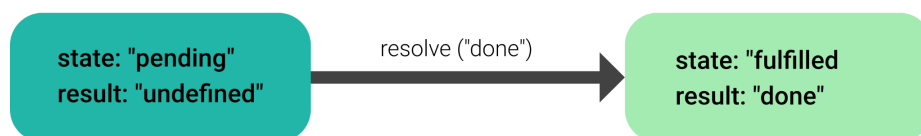


Abbildung 9: Lebenszyklus eines Promise-Objekts

1.3.4. Async/Await

Im ECMAScript-Standard 2017 wurden mit “*async*” und “*await*” zwei Schlüsselwörter eingeführt um den Einsatz mit Promises weiter zu erleichtern. Diese bieten eine alternative Syntax, mit der das Ergebnis bei der Auflösung eines Promis als Wert, wie in jeder anderen Funktion, zurückgegeben werden kann anstatt wie zuvor in “*.then*” oder “*.catch*”-Methoden zur Verfügung

zu stellen. Dazu können JavaScript-Funktionen mit dem Keyword `“async”` definiert werden um zu signalisieren, dass es sich um eine asynchrone Funktion handelt, die als Ergebnis ein Promise zurückgibt. [12]

Mit Hilfe des Ausdrucks `“await”` lassen sich als `“async”` deklarierte Funktionen pausieren, bis das Promise aufgelöst wird. Die Funktion wartet dabei auf die Entscheidung des übergebenen Promise und setzt dann die Ausführung, sobald das Promise-Objekt erfüllt oder abgewiesen wird, weiter fort. Das Schlüsselwort `“await”` darf dabei nur innerhalb von `“async”`-Funktionen verwendet werden.[13]

```
1
2  async function f () { //Async Function
3      let promise = new Promise((resolve, reject)=> { //Promise Constructor
4          setTimeout(()=> resolve("Hallo Welt"), 1000); //Executor
5      });
6
7      let result = await promise; //Wait until Promise resolves
8
9      console.log(result); //"Hallo Welt"
10 }
11
```

Abbildung 10: Definieren einer asynchronen Funktion mit den Schlüsselwörtern `“async”` und `“await”`

In diesem Beispiel wird die Ausführung der Funktion `“f”` in Zeile 7 pausiert und dann fortgesetzt wenn das Promise, hier nach einer Sekunde, aufgelöst wird. Der ermittelte Wert kann dann in der Variable `“result”` verarbeitet werden.

Vorteil ist, dass keine CPU Ressourcen verbraucht werden und die JavaScript-Engine bis zur Erfüllung des Promise das Skript fortsetzen oder auf Events reagieren kann. Zudem bieten die Schlüsselwörter eine übersichtlichere Syntax als mit `“.then”`-Funktionen das Ergebnis des Promises zu handeln.

1.4 TypeScript

Größere Applikationen können sehr schnell sehr komplex werden. In der Webentwicklung führt dies recht häufig zu Bugs, die erst zur Laufzeit im Browser entdeckt werden. Dies resultiert in unzuverlässigen Programmen, verärgerten Nutzern und auch in einer negativen Entwicklungserfahrung. Zwar ist JavaScript vielfältig einsetzbar und wird, wie in 1.5 erläutert, in verschiedenen Bereichen genutzt. Gerade in vielschichtigeren Projekten ist die Programmier-

sprache allerdings nicht immer leicht zu handhaben. [14] Entwickler die bereits Erfahrung in konventionellen Programmiersprachen wie Java oder C# haben, werden feststellen, dass JavaScript aufgrund der **dynamischen Typisierung** ungewohnt viele Freiheiten lässt. [15] Der Einsatz, der von Microsoft entwickelten Programmiersprache, soll insbesondere in größeren Projekten eben jene Nachteile der schwachen Typisierung in JavaScript durch neue Konzepte und Alternativen ausgleichen. [14] So können Entwickler durch die Kombination aus TypeScript und dessen Compiler Code schreiben, der besser gewartet werden kann, für Außenstehende einfacher zu verstehen ist und weniger Bugs enthält. Darüber hinaus wird der Workflow optimiert, indem mehr Fehler bereits während der Entwicklung erkannt und behoben werden können. [16]

TypeScript erweitert JavaScript, wie der Name schon andeutet, um eine optionale, statische Typisierung zum Zeitpunkt der Kompilierung. [14] Während im herkömmlichen JavaScript der Typ neuer Variablen nicht deklariert werden muss und erst zur Laufzeit eines Programms geprüft wird, kann der Entwickler bei diesem Konzept den Typ einer Variablen bereits bei der Erstellung explizit anzugeben, wodurch Konvertierungsfehler zwischen den Datentypen eliminiert werden können. Zu den primitiven Datentypen zählen hierbei *Boolean*, *String*, *Number*, *null*, *undefined* und *symbol*. [16]

```
1
2  let id : number = 2351525;
3  let loggedIn : boolean = true;
4  let lastName : string;
5
```

Abbildung 11: Deklaration des Datentyps einer Variablen in TypeScript

In diesem Beispiel wird das Schlüsselwort *“let”* genutzt um Variablen als Zahl, Boolean und String zu deklarieren, wobei die ersten beiden Variablen gleich initialisiert werden.

```
1
2  function getSum (number1:number, number2:number) :number{
3  |    return number1 + number2;
4  |  }
5
```

Abbildung 12: Deklarieren der Datentypen der Argumente in einer Funktion

Dieser Code-Ausschnitt verdeutlicht die Möglichkeit auch Funktionen entsprechende Datentypen zuzuweisen. Dazu können sowohl die Parameter, als auch der Rückgabewert mit einem festen Typ definiert werden.

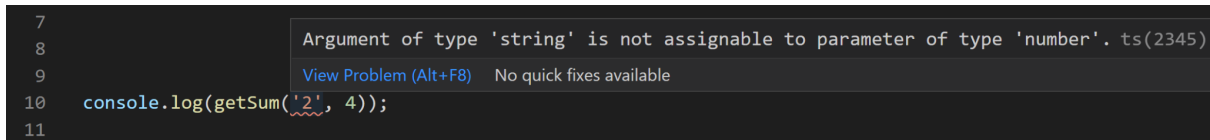


Abbildung 13: Fehlerbehandlung in TypeScript vor der Kompilierung

Falls beispielsweise hier für einen der Parameter ein String anstelle eine Zahl übergeben wird, wird der Entwickler direkt darauf hingewiesen, dass es sich um einen Fehler handelt, der Code kann nicht kompiliert werden und Bugs können dementsprechend früher erkannt werden.

Neben einem statischen Typsystem bietet TypeScript auch objektorientierte Konstrukte wie Schnittstellen, Klassen, Vererbung und Module sowie Lambda-Ausdrücke, die auch im ECMAScript 6 Standard übernommen wurden. Somit kommt die Übermenge TS dem Programmierstil von Hochsprachen wie .NET, C# oder Java wesentlich näher, als klassisches JavaScript. [17]

Damit TypeScript-Code auf einer Maschine lauffähig werden kann, muss das Programm in JavaScript Quellcode übersetzt werden. [14] TypeScript nutzt dazu einen Prozess namens **Transpilierung**. Dies ist eine Methode, bei der Code aus einer Sprache in eine andere Sprache kompiliert oder konvertiert wird. Das bedeutet, dass der gesamte TypeScript-Code letztendlich in JavaScript-Code umgewandelt wird, bevor er schließlich bereitgestellt und ausgeführt wird. [16] Von Vorteil für die Entwickler ist, dass daher zahlreiche Grundlagen von JavaScript auch für TypeScript gelten und gängige Bibliotheken wie Angular oder jQuery weiterhin eingesetzt werden können. [18]

1.5 JSON

Dateiendung: **.json**
MIME-Type: **application/json**
Zeichenkodierung: **UTF-8**

Die JavaScript Object Notation ist ein kompaktes Datenformat in einer einfach lesbaren Textform und dient dem Zweck des Datenaustausches zwischen Anwendungen. JSON ist von der Programmiersprache unabhängig. Parser und Generatoren existieren in allen verbreiteten Sprachen. [19] Es wird verwendet, um Daten zu speichern oder zu transportieren, meist für den Datenaustausch zwischen Internetseiten und dem Webserver (*Client und Server*). Objekte sind in JavaScript sehr zentral, da außer den primitiven Datentypen alle Werte in JavaScript Objekte sind, so z.B. Arrays, Funktionen, Daten, etc. [20] Es ist jedoch zu beachten, dass die JSON-

Syntax zwar von der JavaScript-Objekt-Syntax abgeleitet ist, aber das JSON-Format nur Text unterstützt. [21] JavaScript kann JSON-Dateien direkt in JavaScript-Objekte umwandeln.

Das JSON-Datenformat ist nicht an JavaScript gebunden, sondern kann durch jede andere Programmiersprache gelesen und generiert werden. Dies macht das Format auch für die Kommunikation mit anderen Programmen sehr nützlich.

JSON-Objekte beginnen mit `{` und enden mit `}`. Sie enthalten eine durch Kommas geteilte, ungeordnete Liste von Eigenschaften. Objekte ohne Eigenschaften („leere Objekte“) sind zulässig. Objekte bestehen aus **Schlüssel-Wert-Paaren**, welche jeweils durch einen Doppelpunkt getrennt sind und durch ein Komma von dem nächsten Schlüssel-Wert-Paar getrennt werden. Der Schlüssel ist hierbei eine Zeichenkette und der Wert ist ein beliebiges Element.

Die Syntax einer JSON-Datei sieht dabei wie folgt aus:

```
1  {
2    "vorname": "Kai",
3    "nachname": "Hasuer",
4    "alter": 14,
5    "schueler": true,
6    "beruf": null,
7    "adresse": {
8      "wohnort": "Karlsruhe",
9      "postleitzahl": 76133,
10     "straße": "Bergstraße",
11     "hausnummer": 13
12   },
13   "hobbies": ["tennis", "fußball"]
14 }
```

Abbildung 14: Darstellung eines JSON-Objekts

Unterstützte Datentypen



Abbildung 15: JSON Datentypen

Daten sind hierbei schachtelbar, so wäre zum Beispiel denkbar:

- Ein Objekt im Objekt im Objekt.
- Ein Array im Objekt, der wiederum weitere Objekte enthält.

Ein Nachteil des JSON-Datenformats ist, dass einige nützliche Datentypen nicht unterstützt werden. So beispielsweise: [20]

- **Funktionen** (werden verworfen)
- **Datum / Date** (wird zu String konvertiert)
- **Undefined**

Eine Erweiterung von JSON stellt BSON (*Binary JSON*) dar, welches viele weitere Datentypen unterstützt wie beispielsweise *Binary Data*, *Date*, *Undefined*, *Timestamp* oder *ObjectId* und in der Datenbasis MongoDB zum Einsatz kommt. Zudem ist es direkt in JavaScript möglich, Objekte in JSON-Syntax zu erstellen, zu verarbeiten und über die Punkt-Notation auf die einzelnen Eigenschaften und Werte zuzugreifen.

```
1
2 const car = {
3   "marke" : volvo,
4   "baujahr" : 2005,
5   "kilometerstand" : 129567.54,
6   "fahrer" : { "name" : "Peter Klase", "alter" : 25, "unfallfrei" : true },
7   "vorbesitzer" : [ "Michel Laase", "Paul Teel" ]
8 }
9
```

Abbildung 16: Erstellen eines Objekts in JavaScript

```
10
11  var age = car.fahrer.alter;
12  console.log(car.baujahr);
13
```

Abbildung 17: Zugriff auf Objekteigenschaften in JavaScript

1.6 Überblick der heutigen Einsatzgebiete

Die Sprache JavaScript, die ursprünglich zur Erzeugung von dynamischen HTML entwickelt und in Anfangszeiten auch weitgehend zur clientseitigen Programmierung im Browser genutzt wurde, findet inzwischen auch in einigen weiteren, vielfältigen Einsatzgebieten Verwendung. [22]

1.6.1 Frontend Entwicklung

Auch über 25 Jahre nach der Einführung von Netscape ist das Haupteinsatzgebiet der Skriptsprache nach wie vor die Entwicklung auf dem Client und ist in der heutigen Zeit aus nahezu keiner Website oder Anwendung mehr wegzudenken. Anfangs wurde JavaScript überwiegend dazu genutzt um statische Webseiten mit Interaktionen zu erweitern und auf die verschiedenen HTML-Elemente zur Laufzeit einfluss zu nehmen. [17]

Mittlerweile wird JavaScript allerdings auch dazu genutzt, um das Frontend komplexerer Web-Anwendungen zu realisieren oder Single Page Applications zu entwickeln, wobei dies insbesondere mit dem Aufkommen und der Verbreitung zahlreicher Frameworks oder Bibliotheken, die im 2. Kapitel genauer vorgestellt werden, deutlich vereinfacht wird. [23]

1.6.2 Desktop Applikationen

Inzwischen macht die Sprache sogar für die Entwicklung im Bereich von Desktopanwendungen eine gute Figur. Mit Hilfe von Frameworks wie Electron oder NW.js lassen sich dabei klassische Desktop-Applikationen auf Basis von CSS, HTML und JavaScript entwerfen, die gegenüber reiner Web-Apps den Vorteil bieten, die nativen Systemfunktionen des jeweiligen Betriebssystems zu nutzen.

Um plattformübergreifende Desktopanwendungen mit den bekannten Webtechnologien zu entwickeln, werden der OpenSource-Browser Chromium und Node.js kombiniert, um Client und Server in einer einzigen Applikation zu kombinieren. [24] Eine einheitliche API ermöglicht es, dass die Anwendungen systemübergreifend unter Linux, MacOS oder Windows erstellt und

ausgeführt werden. Beispiele für den Einsatz dieser Technologien sind das Prototyping- und UI-Design-Tool Figma, das Streaming-Portal Twitch, der Code-Editor Visual Studio Code oder der Messaging-Dienst Slack. [25]

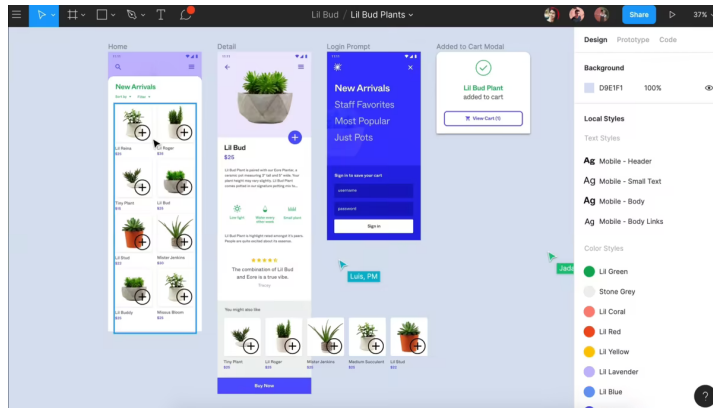


Abbildung 18: Desktopanwendung des Prototyping- u. Design-Tools Figma

1.6.3 Internet of Things

Das Internet der Dinge, oder auf Englisch das Internet of Things (*“IoT”*), ist ein Begriff, der in den vergangenen zehn Jahren immer mehr an Bedeutung gewann und mittlerweile eine Rolle im Leben vieler Menschen spielt. Es gibt keine offizielle Definition für diesen Ausdruck, darunter lässt sich jedoch sinngemäß die digitale Vernetzung beliebiger Gegenstände des Alltags auf der Basis standardisierter Internettechnologien verstehen. [26]

Grundlegendes Ziel dabei ist, dass die vernetzten Geräte miteinander interagieren und Daten austauschen können, um den Menschen dadurch einen gewissen Nutzen zu erbringen. [26] Die Miniaturisierung von elektronischen Bauelementen, die vor allem durch die Entwicklung der Smartphones deutlich forciert wurde, ermöglichte es, verschiedene, zu vernetzende Gegenstände mit Sensoren und Computerbauteilen, wie beispielsweise WLAN-Antennen oder NFC-Tags, auszustatten. [27]

Eine solche Vernetzung von alltäglichen Gegenständen wird insbesondere durch den Einbau von **Mikrocontrollern** realisiert. Die Hardware kann mit zusätzlichen Sensoren Informationen aus der realen Welt in digitale Daten umwandeln. Ein repräsentatives Beispiel für diese Smart Devices wäre ein, mit Kameras, Schließsensoren und Temperaturmessern, ausgestatteter Kühlschrank. Die Vernetzung des smarten Geräts macht es für den Besitzer möglich, beispielsweise über das Smartphone, Benachrichtigungen zu erhalten, wenn das Gefrierfach nicht richtig geschlossen ist oder vor dem Einkauf noch einmal einen Blick in den Kühlschrank zu werfen. [26]

Mögliche Programmiersprachen, die zur Realisierung von IoT-Anwendungen eingesetzt werden können reichen von etablierten Sprachen wie Java oder C++ bis hin zu sehr spezifischen Optionen wie Go oder Swift. Immer größerer Beliebtheit erfreute sich dabei in den letzten Jahren JavaScript im Zusammenspiel mit der Laufzeitumgebung Node.js, die nach einer Umfrage bei IoT-Entwicklern inzwischen weit verbreitet ist. [28]

Unter dem Namen Espruino wurde Anfang der 2010er Jahr sogar ein interaktiver JavaScript-Interpreter als Open-Source-Projekt veröffentlicht. Ein Vorteil dabei ist, dass die Software nur auf dem Controller selbst und nicht auf dem Entwicklersystem installiert wird. So reicht eine Verbindung über ein serielles Terminal aus um den Code zu schreiben oder diesen zur Laufzeit zu ändern und zu debuggen. Neben einer Vielzahl an verschiedenen, unterstützten Boards wurde zudem ein eigenes Espruino Board entwickelt. [29]

1.6.4 Backend Entwicklung

Viele größere Webanwendungen nutzen serverseitige Technologie zur dynamischen Darstellung von Informationen, wie zum Beispiel aus einer Datenbank, die je nach Bedarf bereitgestellt werden können. So wäre es bei der großen Anzahl an verschiedenen Produkten auf Online-Shops oder Beiträgen in Sozialen Netzwerken vollkommen ineffizient diese als komplett statische Seiten zu implementieren. Stattdessen werden hierfür statische Vorlagen erstellt und die Daten aktualisiert, wenn diese von einem Nutzer beim Aufruf einer Produktseite oder eines Posts benötigt werden. [30]

Während JavaScript in seinen Anfangszeiten, wie zuvor beschrieben, zunächst clientseitig eingesetzt wurde, ermöglichte die Laufzeitumgebung Node.js den Programmcode auch außerhalb eines Webbrowsers auszuführen, wodurch die Sprache erstmals dazu genutzt werden konnte um auch serverseitige Programme zu schreiben.

2. FULLSTACK DEVELOPMENT

Der Begriff des Fullstack Developments verbreitete sich in den letzten Jahren im Bereich der Webentwicklung immer mehr. Grundlegend ist darunter zu verstehen, dass moderne Webanwendungen in der Regel aus einem Stapel verschiedener Schichten, dem sogenannten Stack, zusammengesetzt sind.

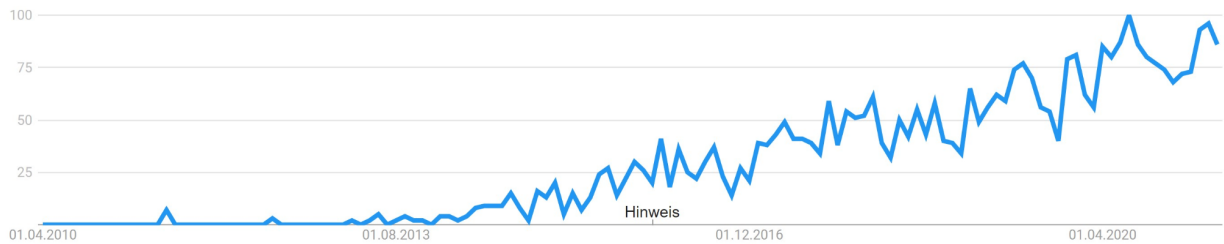


Abbildung 19: Google Search Trends für den Begriff "Full Stack Development" [31]

Da die einzelnen Bestandteile der Applikation mit Hilfe verschiedener Technologien entwickelt werden können, soll dieses Kapitel zunächst einen Überblick über den Aufbau des Stacks und der Kommunikation zwischen den Schichten geben, sowie mögliche Ansätze und Programmiersprachen vorstellen. Dabei wird insbesondere die Entwicklung, mit der in Kapitel 1 behandelten Sprache JavaScript beleuchtet. So ist inzwischen mit Hilfe verschiedener Frameworks und Erweiterungen möglich eine vollständige Anwendung von vorne bis hinten, d.h. vom Frontend bis zum Backend, mit einer einzigen Codebasis zu implementieren.

2.1 Praxisbeispiele und Motivation

Infolge der kontinuierlichen Weiterentwicklung der Sprache JavaScript und der immer größer werdenden Community begannen sich im Laufe der Zeit nach und nach mehr Entwickler mit der Idee zu beschäftigen, vollständige Webanwendungen mit der Skriptsprache zu implementieren. Den Grundbaustein dafür bildete die Einführung der Laufzeitumgebung Node, die in 2.4.3 genauer vorgestellt wird.

Aktuelle Beispiele aus der Praxis machen dabei deutlich, dass das Konzept "Fullstack Development mit JavaScript" längst zu einer echten Alternative im Vergleich klassischen Ansätzen geworden ist und bereits selbst Global Player des Internets wie beispielsweise Netflix, PayPal oder LinkedIn sich für diesen technologischen Ansatz entscheiden. Dabei spielen

Netflix

Warum setzt Netflix auf Fullstack JavaScript? [33]

- NETFLIX**

Home TV Shows Movies Recently Added My List

SEARCH KIDS DVD TV SHOWS

N SERIES MONEY HEIST

▶ Play + My List

Watch Part 3 Now

With millions of euros and their lives on the line, nine robbers attempt to pull off the greatest heist of all time.

TV-MA

New Releases

ON JUNE 12

Emmy Winner • Prison • Gal Pals

WHITNEY CUMMINGS: THE UNDISCOVERED COUNTRY NEW EPISODES

POUNCEWASHER INCARCERATED NEW EPISODES

OTHERHOOD

BASKETBALL

Continue Watching

MONEY HEIST NEW EPISODES

GREAT HACK

I AM MOTHER

WORLD'S SLICES WEIRDO

BLACK HOLE APOCALYPSE

PayPal

20

Warum setzt PayPal auf Full Stack JavaScript? [32]

- Fähigkeiten können in einem Team gebündelt werden
- Sprachbarriere zwischen Browser und Server wird aufgehoben
- Bessere Möglichkeit auf Nutzerbedürfnisse zu reagieren

LinkedIn

Das soziale Netzwerk der Businesswelt verfügt inzwischen über 575 Mio. registrierte Nutzer, die sich mit Firmen und anderen Kontakten vernetzen oder zu bestimmten Themen austauschen können. LinkedIn nutzt ebenfalls die Node-Plattform für die serverseitige Entwicklung der mobilen App. Während die ersten Versionen noch auf Ruby on Rails basierten konnte die Entwickler durch den Umstieg auf Fullstack JS die Performance optimieren und durch die ressourcensparende Technologie die Anzahl der Server von 30 auf 3 reduzieren. [32]

Warum setzt LinkedIn auf Fullstack JavaScript? [32][33]

- Gute Skalierbarkeit
- Performanceoptimierung auf der Clientseite
- Ressourcenreduktion auf der Serverseite

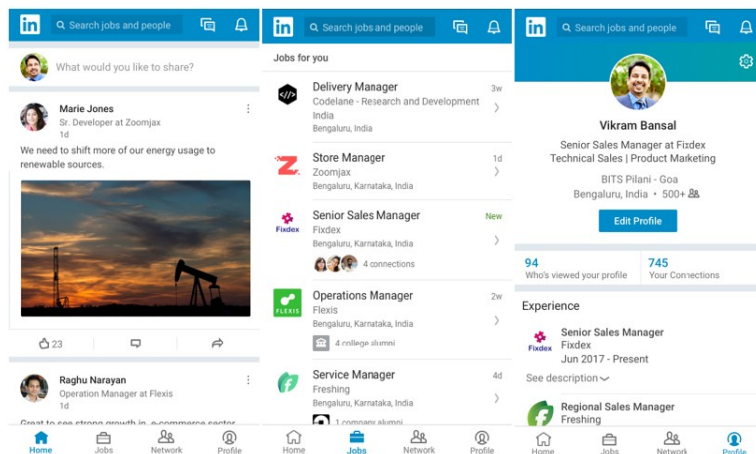


Abbildung 21: Neues Frontend von LinkedIn Mobile

Über

Das US-amerikanische Unternehmen bietet mittlerweile in vielen Städten dieser Welt Vermittlungsdienste für Personenbeförderung von Privatpersonen an. Alle Fahrer sind dabei in einer Datenbank gespeichert sind, die ständig, je nach Verfügbarkeit und Aufenthaltsort der Mitarbeiter, aktualisiert wird. Für das komplexe System, das Fahrer und Passagiere zuverlässig in

Echtzeit miteinander verbindet, entschied sich das Unternehmen seine Anwendung komplett auf JavaScript aufzubauen und war dabei eine der ersten Firmen, die auf dieses Konzept setzen. [32]

Warum setzt Uber auf Fullstack JavaScript? [32][33]

- Fähigkeit, große Datenmengen schnell und zuverlässig verarbeiten zu können
- Einfache Fehleranalyse
- Kontinuierliche Weiterentwicklung der Technologien durch die Open-Source Community

Doch nicht nur in großen Tech-Unternehmen ist der Ansatz *“Fullstack JavaScript”* eine immer beliebtere Lösung. Gerade einzelnen Entwicklern ist so die Möglichkeit gegeben, eigene Webanwendungen zu umzusetzen, ohne sich dabei zuerst zusätzliche Kenntnisse in anderen Programmiersprachen aneignen zu müssen.

Die End-to-End Programmierung mit JavaScript wird wohl in der Zukunft eine immer wichtigere Rolle in den vielfältigen Bereichen des Webdevelopments einnehmen und Fullstack Entwickler aufgrund ihres breiten Spektrums insbesondere bei kleineren Start-Ups, aber auch bei größeren Firmen sehr gefragt sein.

2.2 Client-Server-Modell

2.2.1. Aufbau

Das Client-Server-Modell, oder auch **Client-Server-Architektur** genannt, ist ein Architekturkonzept zur Verteilung von Diensten und Aufgaben in einem Netzwerk. [35] Client und Server sind zwei Programme, die in einer Beziehung zueinander stehen, um diese Aufgaben zu erledigen. Beide Teilnehmer kommunizieren gemeinsam lokal oder über ein Netzwerk miteinander, könnten aber auch auf demselben Rechner laufen, wodurch deutlich wird, dass es sich um eine Softwarearchitektur handelt. [36]

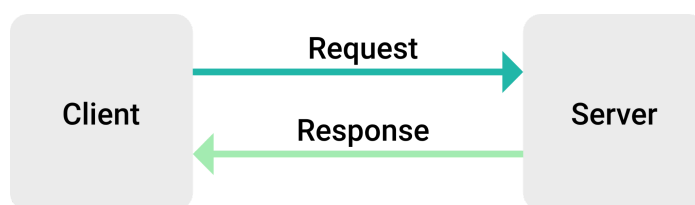


Abbildung 22: Client-Server-Architektur

Ein Server bietet seinen Dienst an und kann bei Bedarf mehrere Clients parallel bedienen. Der Server ist jederzeit in Bereitschaft und wartet passiv auf eine Verbindungsaufnahme durch einen Client. Server sind eigenständige Programme, es können also mehrere Server auf einem einzigen Rechner laufen. [36]

Im Gegensatz zum Server, welcher durchgehend verfügbar sein muss, kann der Client, also eine Anwendung die auf einem Computer wie z.B. einem Desktop-PC oder einem Smartphone läuft, auch ausgeschaltet werden. Ein Programm wird in der Regel dann zu einem Client, wenn es einen Dienst von einem anderen Programm (*Server*) fordert. [36]

Typische Anwendungen des Client-Server-Modells, sind die Abwicklung des E-Mail-Verkehrs, der Zugriff auf Internetseiten eines Webserver, das Transferieren von Dateien oder das Auflösen von Domainnamen im Domain Name System (*DNS*). Protokolle wie FTP (*File Transfer Protokoll*), HTTP (*Hypertext Transfer Protocol*) oder SNMP (*Simple Network Management Protocol*) regeln den Ablauf der Kommunikation und den Informationsaustausch zwischen dem Client und dem Server. [35]

2.2.2. Kommunikation (asynchron/synchron)

Die erste Anfrage (*Request*) wird immer vom Client an den Server gesendet, indem er beispielsweise einen Dienst anfordert. Anschließend wartet der Client auf die Antwort dieses Servers. Der Server, welcher sich in demselben oder auch in einem anderen Netzwerk befinden kann, verarbeitet diese Anfrage und sendet die Antwort (*Response*) zurück an den Client. Ein Server hat auch die Möglichkeit eine Aufgabe weiter zu delegieren, indem er auf andere Server zugreift. Hierdurch nimmt der Server dann selbst die Rolle eines Clients ein. [35] Die Kommunikation zwischen Client und Server kann sowohl synchron als auch asynchron erfolgen:

Bei der **synchronen Kommunikation** (*linke Seite*) kann der Client erst dann die nächste Anfrage senden, wenn er die Antwort vom Server erhalten hat. Zwischen Request und Response muss der Client warten und kann in dieser Zeit keine anderen Aktivitäten ausführen.

Im Falle der **asynchronen Kommunikation** (*rechte Seite*) sendet der Client mehrere Request auf einmal. Die Responses des Servers werden ebenfalls direkt hintereinander gesendet, jedoch kommen diese weder gleichzeitig, noch in der richtigen Reihenfolge beim Client an.

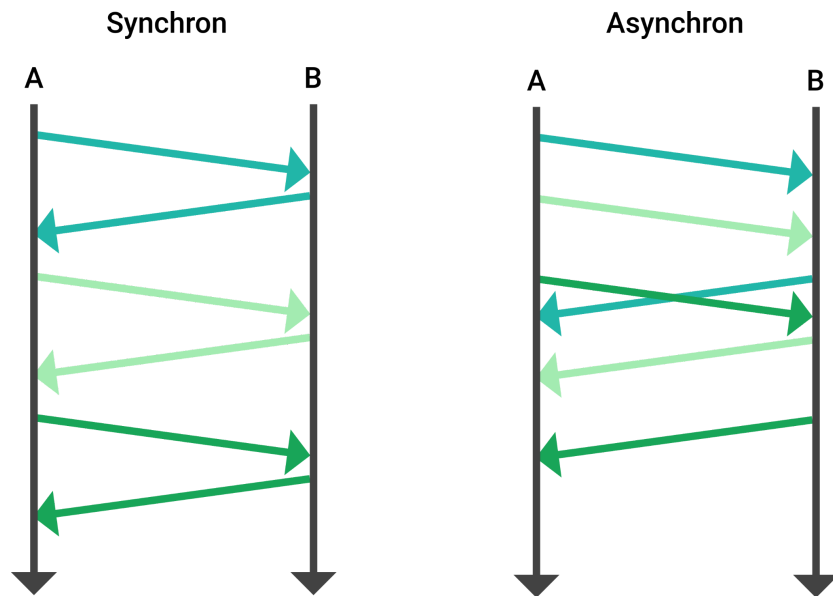


Abbildung 23: Synchrone und asynchrone Client-Server-Kommunikation

2.3 Clientseitige Webentwicklung

2.3.1 Definition und Überblick

Unter dem Begriff “*Frontend*” lässt sich die Präsentationsschicht einer Website oder Anwendung zusammenfassen. Das Frontend umfasst dabei den Teil der Applikation, den der Nutzer sehen und mit dem er interagieren kann. Das Frontend befindet sich daher grundsätzlich auf der Clientseite, wobei die verschiedenen eingesetzten Technologien vom Interpreter des Webbrowsers übersetzt und dem User beispielsweise in Form einer graphischen Benutzeroberfläche (*GUI*) dargestellt werden. Die drei wichtigsten Technologien der clientseitigen Webentwicklung sind neben der JavaScript HTML und CSS. [37]

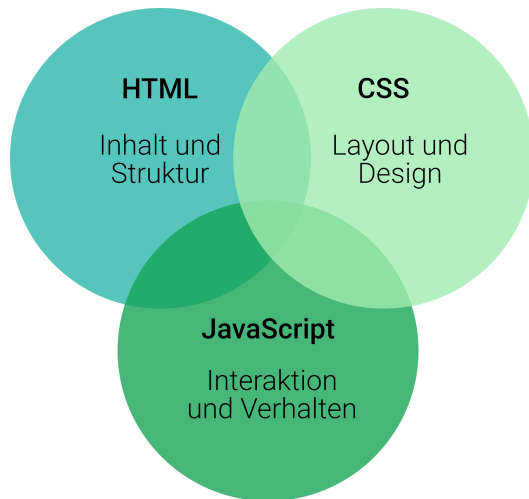


Abbildung 24: Grundlegende Bestandteile und Technologien einer Webseite

HTML

Bei der Hypertext Markup Language, kurz HTML, handelt es sich um eine textbasierte Auszeichnungssprache zur Strukturierung von Webseiten. Die 1992 veröffentlichte und seither stetig weiterentwickelte Sprache bildet den Grundbaustein des Webs und wird von allen Browsern genutzt, um die vom Server abgerufenen Seiten für den Nutzer zu interpretieren.

Die einzelnen HTML-Elemente bestehen aus sogenannten **Tags**. Dabei umschließen immer in der Regel zwei Tags, ein Start-Tag und End-Tag, den Inhalt eines Elements. Jeder Tag hat, abhängig vom Namen eine andere Bedeutung wodurch sich beispielsweise Überschriften, Bilder, Listen, Buttons, Hyperlinks und viele weitere Bestandteile definieren lassen. Das HTML-Dokument weist eine hierarchische Baumstruktur auf und ist grundlegend in einen Head und einen Body aufgeteilt. Der erste Teil enthält Metainformationen für den Browser oder Suchmaschinen und ist nicht direkt für den User sichtbar, während der Body, den auf dem Bildschirm sichtbaren Inhalt umfasst. [38]

CSS

Cascading Stylesheets, kurz CSS, ist eine Auszeichnungssprache, mit der sich das Aussehen der HTML-Dokumente beeinflussen lässt. Während mit Hilfe der HTML-Tags die einzelnen Inhalte auf einer Webseite bestimmt werden können, ermöglicht es CSS, diese zu gestalten und auf der Präsentationsfläche anzuordnen.

Um so das User Interface und die dargestellten Inhalte anzupassen und zu layouten, können verschiedene Regeln definiert werden. Über verschiedene Selektoren lassen sich dann einzelne oder mehrere Elemente bestimmen, für die diese Gestaltungsregeln gelten sollen. Wie der Name der Stylesheet-Sprache schon erkennen lässt, können diese Selektoren auch ineinander verschachtelt werden. [39]

Bei kleineren Projekten können die Stylesheets über einen Tag direkt in den Head der HTML-Seite geschrieben werden. Gerade bei komplexen Webseiten macht es allerdings Sinn die Sheets separat zu definieren und als externe CSS-Datei einzubinden. [38]

JavaScript

Den letzten grundlegenden Bestandteil der clientseitigen Webtechnologien bildet die Sprache JavaScript, welche im ersten Kapitel bereits ausführlich vorgestellt wurde. Im Gegensatz zu HTML und CSS, die nicht als Programmiersprachen definiert werden, lassen sich in JavaScript Schleifen, Bedingungen und Algorithmen schreiben und auf Benutzerinteraktionen reagieren.

Mit Hilfe von JavaScript können so Webseiten auch dynamisch angepasst werden. Dabei dient das **Document Object Model**, kurz DOM, als Schnittstelle zwischen JavaScript und HTML. Zweck des DOM ist es dabei, den Zugriff auf die einzelnen Inhalte zu gewähren um damit das Löschen, Hinzufügen oder Bearbeiten von Elementen und Styles möglich zu machen. Um die hierarchische Struktur im Document Object Model korrekt wiederzugeben, greift die Schnittstelle dazu auf die bereits beschriebene Baumstruktur des HTML-Dokuments zurück, um die auf dem Bildschirm gerenderten Objekte ansprechen zu können. [40]

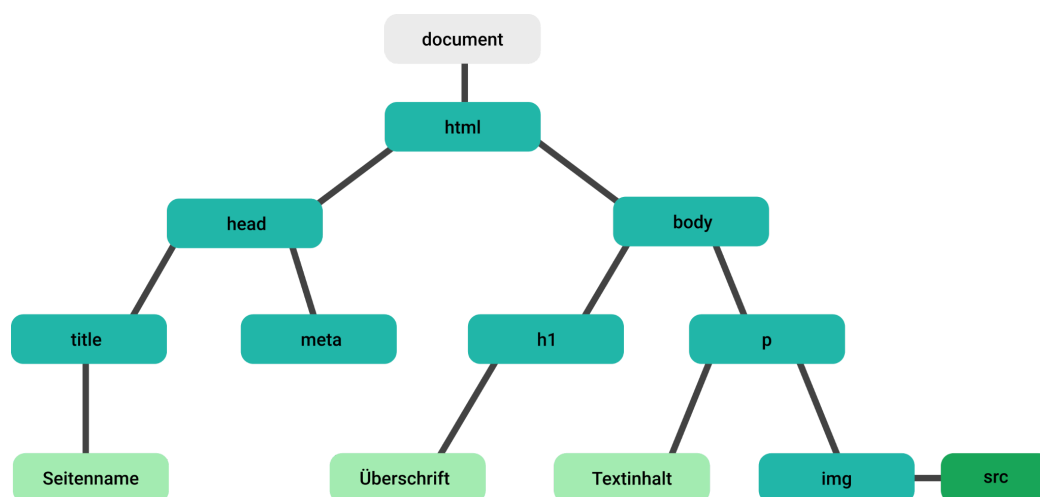


Abbildung 25: Beispielhafte Darstellung der einzelnen Knoten des Document Object Models

2.3.2 Frameworks

Nach dem Durchbruch von JavaScript gegen Ende der 90er Jahre verwandelte sich das Web zunehmend von einem Ort, um ausschließlich statische Informationen abzurufen, zu einem um auch selber Dinge zu tun. Die Bekanntheit von JavaScript nahm in den darauf folgenden Jahren stark zu und immer mehr Entwickler schrieben kleine, hilfreiche Tools in wiederverwendbaren Paketen.

Diese sogenannten **Bibliotheken** dienten dazu, Lösungen zu bestimmten Problemen mit anderen Entwicklern zu teilen, wodurch sich im Laufe der Jahre ein großes Ökosystem entwickelte, das die Entwicklung des Internets maßgeblich beeinflusste. So lassen sich heute neben statischen Webseiten auch immer mehr interaktive, hochdynamische Anwendungen finden, die ermöglichen Aufgaben umzusetzen, welche früher nur als native Anwendungen realisiert wurden. Aus den losen Ansammlungen von verschiedenen Funktionen in Bibliotheken entstanden dann feste Programmiergerüste, sogenannte Frameworks, die es den Entwicklern erleichtern skalierbare und interaktive Webanwendungen zu erstellen. [41]

Mittlerweile gibt es eine Vielzahl verschiedener Frameworks für die clientseitige Entwicklung von Webapplikationen, die teilweise von großen Unternehmen oder Einzelpersonen entwickelt wurden und auch durch den regen Austausch in den immer größer werdenden Communitys kontinuierlich optimiert und erweitert werden. Da die jeweiligen Frameworks unterschiedliche Stärken, Schwächen und Features haben und sich auch teilweise in ihrer grundlegenden Architektur unterscheiden, muss hierbei je nach Anwendungszweck entschieden werden, auf Basis welcher Option man sein Frontend aufbauen möchte. Im folgenden werden mit React, Angular und Vue die drei aktuell am weitesten verbreiteten und eingesetzten Frameworks betrachtet. [42]

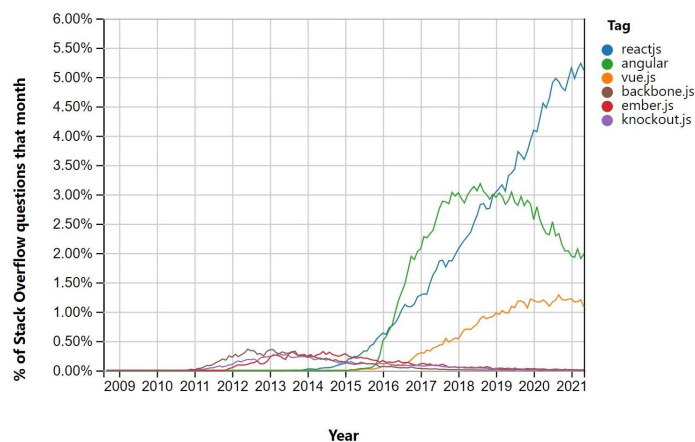


Abbildung 26: Vergleich verschiedener Frontend Frameworks in den StackOverflow-Trends [43]

2.3.2.1 React

React, oftmals auch als React.js oder ReactJS bezeichnet, wurde von **Facebook** entwickelt und bereits einige Jahre intern eingesetzt, bevor das Framework dann im Jahr 2013 veröffentlicht wurde. Es gibt viele Diskussionen darüber ob React als Framework oder als Library bezeichnet werden soll. Technisch gesehen handelt es sich bei React nur um eine Bibliothek, diese kann jedoch auch als Framework zur Erstellung von User Interfaces der Single-Page-Apps genutzt werden. [41]

Im Gegensatz zu anderen Frameworks, die das MVC bzw. MVVM-Muster nutzen (*Model-View-Controller* bzw. *Model-View-Viewmodel*), deckt React nur die View-Komponente, d.h. die für den User sichtbare Präsentationsebene des Models ab. [44] Dazu setzt die Library auf einen **virtuellen DOM**. In Anwendungen und Seiten, die herkömmliches JavaScript nutzen, wird bei Benutzerinteraktionen, der bereits in 2.2.1 beschriebene Baum des HTML-Dokuments über die DOM-Schnittstelle geändert. Während hier jedoch bei schon den kleinsten Änderungen wie dem Einfügen eines Div-Elements oder dem ändern einer Überschrift das komplette DOM neu gerendert werden müsste, um die aktualisierte UI abzubilden (*“Re-paint”*), bietet das von React genutzte Konzept deutliche Performance-Vorteile und ist weitaus weniger schwerfällig.

Bei dem virtuellen DOM handelt es um eine Art abstrakte, schlankere Kopie des realen DOMs. Bei Änderungen wird zunächst nur der Virtual-DOM angepasst und dieser schließlich bei jedem Update über einen Algorithmus mit dem realen DOM verglichen und die Unterschiede dort angepasst, sodass nicht immer alle Knoten überprüft werden müssen. [39]

Eine weitere Besonderheit von React ist der Einsatz von JavaScript XML, kurz **JSX**. Bei einem ersten Blick in den Programmcode einer React-Applikation erscheint dies zunächst etwas ungewöhnlich. Hierbei handelt es jedoch einfach um eine Syntaxerweiterung, die erlaubt, auf einfache Weise HTML-Elemente direkt in JavaScript einzubetten und so den Code übersichtlich zu halten. Damit die Anwendung letztendlich im Browser funktioniert, ist ein Präprozessor notwendig, der die JSX-Syntax in simples JavaScript umwandelt. [45]

```
2
3  const Component = () =>{
4      return(
5          <div>
6              <h3>Hallo Welt!</h3>
7              <p>Lorem Ipsum dolor</p>
8          </div>
9      );
10 }
```

Abbildung 27: JSX-Syntax in React, die ermöglicht HTML-Elemente in JavaScript zu schreiben

Ein wichtiges Merkmal der Library ist außerdem, dass geschriebene Applikationen, wie bei einigen anderen Frameworks auch, **komponentenbasiert** sind. Das bedeutet, dass die Präsentationsebene aus einzelnen Blöcken wie beispielsweise Buttons, Dropdown-Menüs oder Card-Elementen zusammengesetzt wird. React erlaubt es dabei, die Benutzeroberfläche in unabhängige Teile aufzubrechen, wobei jede einzelne Komponente isoliert zu betrachten ist und daher auch problemlos wiederverwendbar ist. [46]

Zusätzlich gibt es inzwischen ein Vielzahl an anderen Libraries wie Material von Google oder Bootstrap, die mit React zusammenarbeiten und eine große Auswahl an bereits vordefinierten und einfach zu stylenden UI-Komponenten bieten. [45]

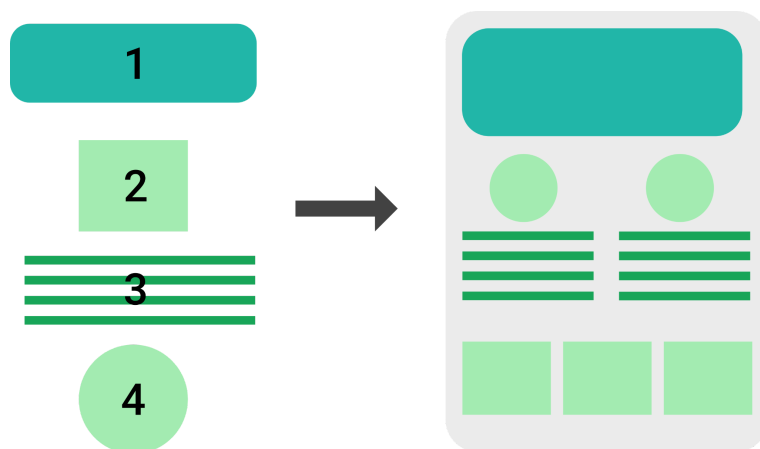


Abbildung 28: Beispielhafter Aufbau einer komponentenbasierten Anwendung

React zielt auf einen minimalen Ansatz ab und bietet den Entwicklern viele Freiheiten und Möglichkeiten bei der Konzeption einer eigenen, flexiblen Architektur für individuelle Probleme die richtige Lösung zu finden. Bei Facebooks Library ist man daher eher gefordert auf die vielfältigen, von der Community gepflegten Module zu setzen, was darin resultiert, dass oftmals mehrere Wege zum Ziel führen. Der Aufwand bei Pflege und Integration der Erweiterungen ist hier höher als z.B. in Angular, jedoch sind React Projekte meist auch simpler und leichtgewichtiger.

2.3.2.2 Angular

Neben React zählt auch Angular zu den populärsten Frameworks für die Erstellung von Single-Page-Applications. Das Framework wurde Anfang der 2010er Jahre mit der Unterstützung von **Google** entwickelt und im Herbst 2016 schließlich in einer grundlegend überarbeiteten, zweiten Version als Angular veröffentlicht. Im Zusammenhang mit dem Framework stößt man oft auch auf die Bezeichnung AngularJS. Damit ist allerdings nur die frühe, erste Version gemeint,

die dann schließlich durch die zweite Version ersetzt wurde. Da es sich hierbei um eine nahezu vollständiges Upgrade mit vielen neuen Konzepten handelte, sind die jüngeren Versionen auch nicht ohne weiteres zum anfänglichen AngularJS abwärtskompatibel. [47]

Im Gegensatz zur Library React, handelt sich bei Angular um ein tatsächliches Framework, welches über die Funktionen einer Bibliothek hinausgeht. Das Programmiergerüst gibt dabei eine bestimmte Struktur und Architektur vor, in dessen Rahmen die eigene Anwendung integriert wird. [48]

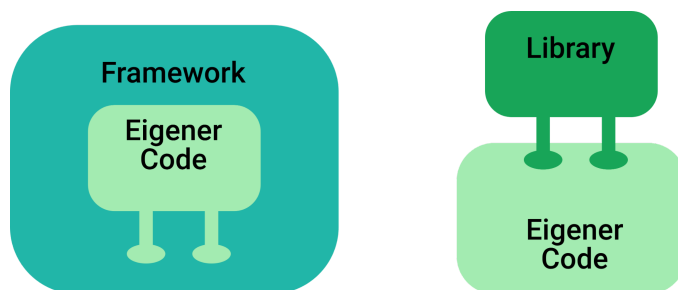


Abbildung 29: Unterschied zwischen einem Framework und einer Library

Googles Angular setzt auf weitgehend auf Konzepte des **MVVM-Architekturmodells** ("Model-View-ViewModel") zu Erstellung der Benutzeroberflächen. Das Viewmodel dient als Bindeglied für das Model mit der Anwendungslogik und der View mit der grafischen Benutzeroberfläche und enthält die UI-Logik.[44]

Das Framework nutzt auf Basis dieser Architektur eine **bidirektionale Datenbindung**. Im Gegensatz zu einer einfachen Datenbindung, wie sie beispielsweise in React zum Einsatz kommt, wird bei dieser Art eine Verknüpfung zwischen den Elementen der View und den Datenstrukturen des Models hergestellt. Änderungen wirken sich daher direkt auf die Logik der Komponenten aus und umgekehrt werden Updates in der Logik direkt in der View der Komponenten übernommen, wodurch keine speziellen Listener benötigt werden und eine manuelle Synchronisation entfällt. [49]

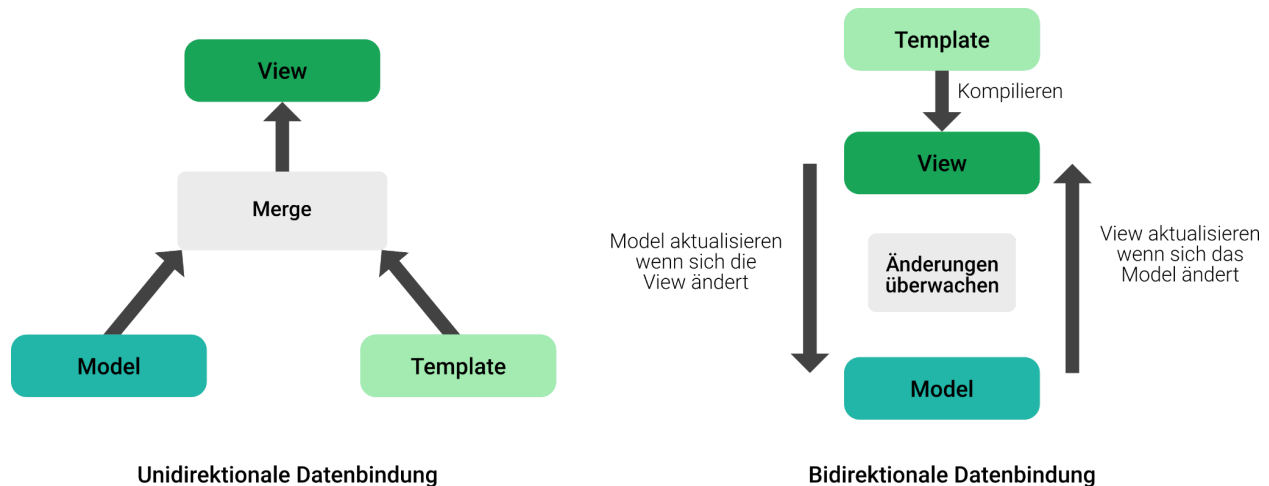


Abbildung 30: Vergleich zwischen unidirektionaler und bidirektionaler Datenbindung

Wie in React auch sind die wichtigsten Bausteine einer Angular-Anwendung sogenannte Komponenten. Diese bestehen immer aus einem **HTML-Template**, einer Klasse, die die View-Logik zur Verfügung stellt und einem Stylesheet. Übergreifende Daten und Programmlogik, die nicht zwingend an eine Komponente gekoppelt sind, lagern die Entwickler in sogenannten Services aus. [50]

Man erkennt also, dass die Applikationen aus den Grundbausteinen des Webs HTML, CSS und JavaScript besteht. Innerhalb des HTML-Templates wird dann Angular-Syntax und Befehle genutzt. Eine Besonderheit ist der Einsatz, der in 1.3 erläuterten Obermenge **TypeScript**, um die Klassen zu definieren und auch vom Angular 2-Team herangezogen wird um das Framework zu entwickeln. [44]

Angular zielt klar auf das Einsatzgebiet der professionellen Entwicklung in Enterprise Projekten ab. Durch die strengereren Vorgaben hinsichtlich der Struktur und Architektur lassen sich gut wartbare und skalierbare Softwarelösungen umsetzen. Aufgrund des massiven Grundgerüsts hat Angular mitunter den Ruf etwas überladen und kompliziert zu sein, bietet auf der anderen Seite allerdings auch ohne externe Ressourcen einen Funktionsumfang, der bei Frontend Frameworks seinesgleichen sucht. [50]

Da das Google Framework in erster Linie für neue Applikationen *“from scratch”* gedacht ist, gibt es jedoch Schwierigkeiten bei dem nachträglichen einbinden und dynamisieren einer bereits vorhandenen Webseite, wofür sich React oder Vue deutlich besser eignen. [42]

2.3.2.3 Vue

Anders als bei Angular und React steht hinter diesem Framework kein großer Global Player sondern mit **Evan You** eine einzelne Person als Initiator und Schöpfer. Der Chinese, der in den Anfangsjahren Mitglied in dem Entwicklerteam hinter Angular war, beschrieb die Vision des 2014 veröffentlichten Open-Source Frameworks wie folgt: *“I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight.”* [42]

Vue.js bietet daher im Gegensatz zu Angular bewusst kein mächtiges, umfassendes Gesamtpaket. Stattdessen beschränkt sich das Framework in seinem Kern bewusst nur auf den **View-Layer** des MVC-Modells, wodurch auch der Name, der im Englischen “vju” ausgesprochen wird, abgeleitet wurde. Durch die schlanke und überschaubare Core-Library kann Vue ähnlich wie React problemlos in bestehende Projekte integriert werden. Zudem bietet das umfangreiche Ökosystem auch Bibliotheken, mit denen Vue bei Bedarf beispielsweise um serverseitiges Rendering oder Routing erweitert werden kann. Damit ist Vue in seinem Ansatz nicht ganz unähnlich zu React, mit dem Unterschied, dass die Zusatzmodule nicht aus der Community sondern direkt aus der Hand von Evan You und seinem Team stammen. [51]

Zudem gilt das Framework als relativ einsteigerfreundlich und hat zu Beginn, gerade im Vergleich mit Facebooks oder Googles Alternativen, eine sehr steile Lernkurve. Ein Grund dafür ist, dass, anders als bei React mit JSX und Angular mit TypeScript, neben HTML, CSS und JavaScript vorerst keine weitere Sprache oder Syntaxerweiterung genutzt werden muss, sondern die HTML-Datei einfach mit dem Vue-Markup angereichert werden kann. Typisch für das Framework ist dabei dessen Templatesyntax mit geschweiften Klammern, die oftmals auch als *“Mustaches”* bezeichnet werden, wodurch dynamisch Daten der Anwendung übergeben werden können.

```
10
11   <div id="app">
12     |   <h1>Message: {{msg}}</h1>
13   </div>
14
```

Abbildung 31: Vue.js Template-Syntax

Wie die bereits vorgestellten Frameworks auch, ermöglicht Vue eine komponentenbasierte Entwicklung dynamischer Webapplikationen, wobei diese auch in **Single-File-Components** zusammengefasst werden können. Dabei wird das HTML-Template, das Styling und der JS-Code in einer einzigen Datei gebündelt, was besonders die Arbeit in komplexeren Projekten vereinfachen kann. [52]

Eine Besonderheit bei Vue.js ist die Verwendung, der an Angular orientierten Direktiven. Diese Richtlinien dürfen hier allerdings nur in Attributen von HTML-Elementen eingesetzt werden und lassen sich an dem Präfix “v-” erkennen, mit dem Elemente ein- und ausgeblendet werden können, sich überwachen oder Iterationen ausführen lassen. Beispiele hierfür wären die Directive “v- if”, mit dem die Erzeugung von DOM-Elementen an eine erfüllte Bedingung geknüpft ist oder “v- bind”, das eine **bidirektionale Datenbindung** ermöglicht. Aufgrund diesem, auch in Angular genutzten, Konzept wird Vue als reaktives Framework bezeichnet. So kümmert sich Vue.js automatisch darum, dass immer wenn sich irgendwo in der Anwendung Daten ändern, diese auch überall, wo sie auf der Website genutzt werden, ebenfalls aktualisiert werden.

Neben vielen Gemeinsamkeiten zu Angular weißt das von Evan You konzipierte Framework auch Parallelen zu React auf und setzt so ebenfalls einen **virtuellen DOM** ein, um Änderungen im realen HTML-Baum zu überprüfen. [53]

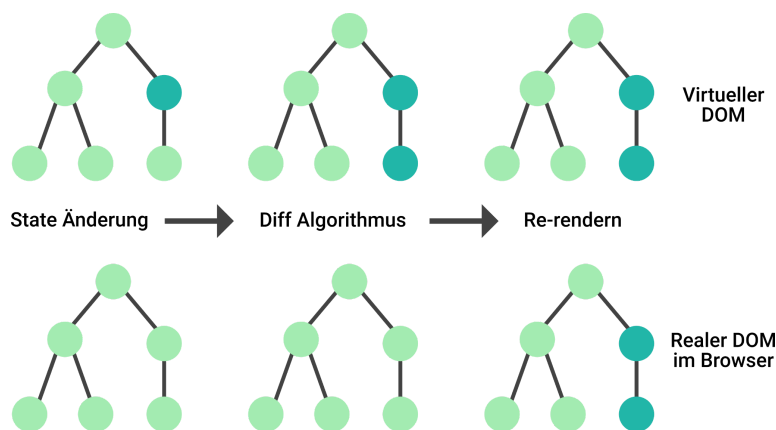


Abbildung 32: Rendern der UI-Elemente mit Hilfe des virtuellen DOM

Vue hat sich in den letzten Jahren zu einer echten Alternative zu den Frameworks von Google und Facebook entwickelt und bedient dabei etwas die Anforderungen zwischen den beiden großen Konkurrenten, indem es zum einen klare Strukturen bietet, aber auf der anderen Seite weiterhin versucht leichtgewichtig zu bleiben und in seinem Kern mit wenig Overhead auszukommen.

Zwar ist die Community um Vue noch nicht so groß wie bei Angular oder React, jedoch gibt es eine gute Dokumentation. So lässt sich das Framework ohne Bedenken gerade für kleinere bis mittelgroße Anwendungen einsetzen, stößt bei komplexen Enterprise-Projekten allerdings an seine Grenzen. [42]

2.2.2.4 Vergleichender Überblick

	React	Angular	Vue
Gründer	Facebook	Google	Evan You
Erstveröffentlichung	März 2013	September 2016	Februar 2014
Lizenz	MIT/Open-Source	MIT/Open-Source	MIT/Open-Source
Geschrieben in	JavaScript	TypeScript	JavaScript
Datenbindung	Unidirektional	Bidirektional	Bidirektional
Model	Virtual DOM	MVVM + realer DOM	Virtual DOM
UI-Rendering	Cient- oder serverseitiges Rendering	Clientseitiges Rendering	Cient- oder serverseitiges Rendering
Struktur und Komponenten	HTML, JSX	HTML-Templates, CSS, TypeScript	HTML, CSS, JS
Lernkurve	Moderate Lernkurve für Anfänger	Flache Lernkurve für Anfänger	Steile Lernkurve für Anfänger
Ideal für	moderne Single-Page- oder Progressive Web Apps,nachträgliches dynamisieren von bestehenden Websites	hochskalierende, umfangreiche Enterprise-anwendungen	leichtgewichtige Single-Page-Apps und kleine bis mittelgroße Start-Up-Projekte
Popularität	145k GitHub Sterne, 10,9 Mio Npm Downloads/Monat	58k GitHub Sterne, 0,58 Mio Npm Downloads/Monat	158k GitHub Sterne, 2,8 Mio Npm Downloads/Monat
Einsatz in Unternehmen	Uber, Netflix, PayPal, Twitter, Reddit, Instagram	Forbes, Wix, YouTube, Weather.com, GitHub, Samsung	Xiomi, Alibaba, Adobe, Nasa, BMW, Apple

2.4 Serverseitige Webentwicklung

2.4.1 Definition

Bei der serverseitigen Webentwicklung werden die Anwendungen direkt auf dem Server ausgeführt, um eine dynamische Webseite für einen bestimmten Client zu erstellen, wenn er diese zur Anzeige anfordert. Die Skripte besitzen Zugriff, auf die sich auf dem Server befindlichen Datenbanken, deren Inhalte werden genutzt, um diese in das Webprojekt einzubauen. Im Gegensatz zur clientseitigen Programmierung im Web, bei dem das Skript direkt im Browser ausgeführt wird, bleibt der Quellcode der Skripte für den User verborgen, da er nur auf die HTML-Seite zugreift. [54]

Ein Nachteil der serverseitigen Webentwicklung ist die Notwendigkeit, des permanenten Sendens von Anfragen des Clients über das Netzwerk an den Webserver, um neue oder veränderte Inhalte zu erhalten und um diese anschließend wieder in einem Web-Browser darzustellen. Dies führt zum einen zu einer schlechteren Nutzungserfahrung für den User durch eingeschränkte Performance und zum anderen werden die Server durch diese permanenten Anfragen stark ausgelastet, da zusätzlich noch eine dauerhafte Verbindung zwischen Client und Server vonnöten ist. [55]

Während für Webanwendungen früher C-Programme und Perl-Skripts verwendet wurden, welche anschließend vom Server-Betriebssystem ausgeführt, interpretiert und deren Ausgabe anschließend über das Common Gateway Interface zum Webserver übermittelt wurde, können moderne Webserver mittlerweile Skripte direkt ausführen. Für serverseitige Webprogrammierung verwendete Skriptsprachen zur Erzeugung dynamischer Webseiten sind unter anderem PHP, Python, Ruby und Java. [55]

2.4.2 Klassische Backend Programmierung

LAMP ist eine Stack-Architektur, die aus vier Schichten besteht:

- **L**inux
- **A**pache
- **M**ySQL
- **P**HP

Linux ist das Open Source Betriebssystem, auf dem die restlichen Programme laufen und bildet somit die unterste Schicht. *Apache* ist die Web-Server-Software, welche den Client mit der

Internetseite verbindet und seine Requests bearbeitet. *MySQL* ist die Datenbank auf Schicht Zwei und dient dem Speichern von Daten. *PHP* ist die Programmiersprache, welche den dynamischen Teil der Web-Anwendung realisiert, also das Anpassen der Inhalte an Client- Anfragen oder aktuelle Inhalte der Datenbank und das anschließende Einbetten in das HTML-Dokument. Diese bildet die oberste Schicht. Die Webseiten und Web-Services laufen auf dieser Schicht. [56]

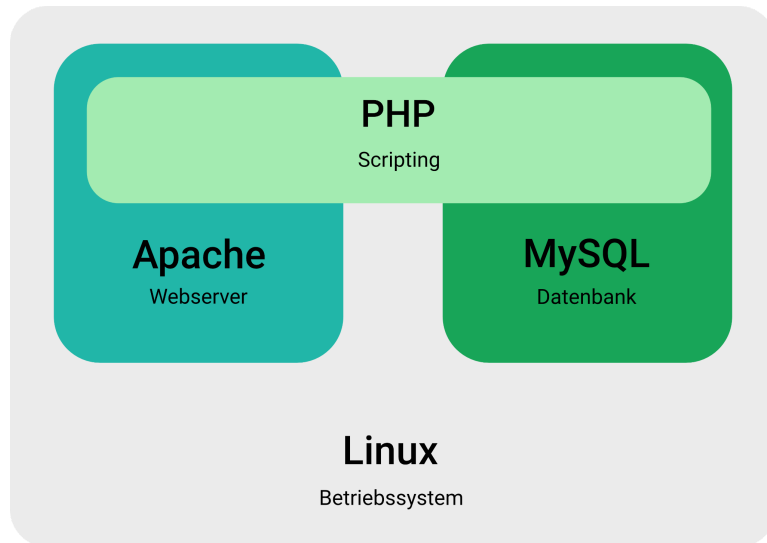


Abbildung 33: Schichtenarchitektur des LAMP-Stack

LAMP war einer der ersten Open-Source-Stacks für das Web und bleibt noch immer einer der gängigsten Wege, um Web-Anwendungen umzusetzen. Es ist sehr weit verbreitet und wird z.B. bei Wordpress und Drupal eingesetzt. Die Architektur gilt als „**stable, simple and powerful**“ (*d.h. stabil, simpel und mächtig*). Zudem gilt es als sehr sicher dank seiner Architektur und den bewährten Verschlüsselungsverfahren. [56] Der Stack ist sehr flexibel. So kann an Stelle von PHP beispielsweise auch Python oder Perl eingesetzt werden. Auch alternative Betriebssysteme sind möglich: „*WAMP*“ mit Microsoft Windows oder „*MAMP*“ mit Mac OS. [57]

Der limitierende Faktor der Skalierbarkeit einer LAMP-Anwendung ist MySQL, da die Performance bei einer hohen Anzahl von Datenbank-Interaktionen (*“read/write”*) nicht ausreicht und die Datenbank so einen Flaschenhals bildet. [56]

Es gibt viele serverseitig einsetzbare Programmiersprachen zum Erzeugen dynamischer Internetseiten. Einige dieser Sprachen sind: [58]

- **PHP**
- **ASP (Active Server Pages)**
- **C++**
- **C#**
- **Java / Java Servlets / Java Server Pages**
- **Python**
- **Ruby / Rails (Ruby on Rails)**
- **GO**
- **Jakarta EE / Java Enterprise Edition**
- **.NET**
- **Perl**
- **Swift**

Zudem Datenbanksprachen wie :

- **SQL**
- **MongoDB**

Weitere gängige Stacks / Architekturen sind: [58]

- **LEMP Stack** bestehend aus JavaScript, Linux, Nginx, MySQL und PHP
- **Django Stack** bestehend aus JavaScript, Python, Django und MySQL
- **Ruby on Rails** bestehend aus JavaScript, Ruby, SQLite und Rails

Wenn man diese Architekturen betrachtet sieht man, dass für die Umsetzung von Projekten mit diesen Stacks mehrere Programmiersprachen verwendet werden müssen. Die Syntax der Sprachen kann dabei sehr unterschiedlich sein und die Interaktion zwischen den verschiedenen Programmen ist nicht immer leicht umzusetzen. Genau da liegt der Hauptvorteil der Full Stack Javascript Entwicklung: Man benötigt für die Umsetzung nicht mehrere Programmiersprachen und hat somit sehr konsistenten Code, der in allen Bereichen gut zusammenarbeitet. Was man zum Beispiel bei LAMP in PHP und SQL programmieren muss, kann man in Node, das im Folgenden detaillierter beleuchtet wird, komplett in JavaScript schreiben. Zudem wird bei Node-Anwendungen auch keine Webserver-Software wie Apache auf der zweiten Schicht benötigt, da dessen Funktionen ebenfalls komplett in der Laufzeitumgebung implementiert werden können.

2.4.3 Node.js

“Node.js eröffnet dem Frontend-Entwickler die Welt des Backends”, erklärte Dmytro Zharkov, langjähriger Software Engineer bei Zalando, in seinem Talk auf der International JavaScript Conference 2018 in München. So erlaubt es die Laufzeitumgebung End-to-End Anwendungen mit einer einzigen, aus der clientseitigen Webentwicklung vertrauten Sprache zu implementieren: JavaScript.

Dieses Kapitel legt daher den Blick genauer auf die Open-Source-Plattform und behandelt neben der Entstehungsgeschichte und Motivation des Initiators Ryan Dahl die grundlegende Architektur von Node.js und stellt die wichtigsten Schnittstellen der Kernbibliothek und den populären Paketmanager NPM vor.

2.4.3.1 Entstehung und Hintergrund

Lange Zeit war JavaScript als eingebundene Skriptsprache für den Einsatz in Webbrowsern ausgelegt. Dies änderte sich erst mit der Entwicklung der Laufzeitumgebung Node.js, wodurch es möglich wurde JavaScript-Code erstmalig auch außerhalb von Browsern auszuführen. Die plattformübergreifende Umgebung wurde 2009 von dem Programmierer Ryan Dahl entwickelt. [59]

Das ursprüngliche Problem, welches Ryan Dahl auf diese Idee brachte, war, als dieser nacher einer einfachen Lösung suchte, einen Fortschrittsbalken für Up- und Downloads in einer Webanwendung zu implementieren. [60] Da ihm Applikationen, geschrieben in PHP oder dem Ruby-Framework, zu ineffizient und langsam waren und die CPU-Kapazität nahezu auslasteten, experimentierte Dahl zunächst zunächst mit Mongrel. Der Webserver für Ruby-basierte Anwendungen, konnte ganz im Gegensatz zu den damaligen, klassischen Servern auf Nutzeranfragen reagieren und anstelle statischer HTML-Seiten auch dynamische Antworten generieren.

Dies war zwar ein erster Ansatz für die Umsetzung der Progressbar, jedoch störte sich Ryan Dahl an der Vielzahl der Requests, die gesendet werden mussten, um den aktuellen Fortschritt darstellen zu können. Kurzzeitig erwog er, aufgrund der Möglichkeit, mehrere Threads nutzen zu können, eine Umsetzung in C, verwarf diesen Ansatz allerdings wieder, da sich Dahl sicher war, dass sich kaum Webentwickler von diesem Einsatzgebiet überzeugen ließen.

Die Suche nach einer perfekten Sprache für sein Problem führte ihn über die funktionale und asynchrone Programmiersprache Haskell bis hin zu JavaScript. Da die Skriptsprache alle seine

Anforderungen erfüllte und diese bereits viele Jahre im Web etabliert war, gab es bereits leistungsstarke Engines und eine Vielzahl an Programmieren. [61]

In seinen Anfängen war Node.js noch ein Open-Source-Projekt, weitgehend ohne Unterstützung von außen. Um sein Projekt weiter vorantreiben zu können, machte sich Ryan Dahl auf die Suche nach Sponsoren und Partnern und stellte seine Plattform auf der *“JSConf”*-Veranstaltung in Berlin vor. So erläuterte Dahl in einer Präsentation die Architektur und Grundlagen von Node.js und demonstrierte, wie mit wenigen Zeilen Code, ein funktionsfähiger Webserver lauffähig gemacht werden kann. Der Vortrag des Amerikaners war dabei so gut und überzeugend, dass das auf Cloud-Computing spezialisierte Unternehmen Joyent Dahl einstellte und ihm die nötigen Ressourcen ermöglichte, um seine Arbeit an dem Projekt weiter fortsetzen zu können. [60]

Die ersten Versionen von Node hatten oftmals mit Performanceproblemen zu kämpfen und häufige, größere Änderungen und Brüche in der Schnittstelle führten dazu, dass sich die kritischen Stimmen gegenüber der Umgebung schnell mehrten. Das Team um Ryan Dahl arbeitete jedoch kontinuierlich an den vielen Problemen und konnte schließlich mit der Version 0.6 einige wichtige Änderungen hinzufügen. Neben der Einführung des Node Package Managers wurde hier eine weitere Signifikante Schwachstelle behoben und Node konnte endlich problemlos unter Windows installiert und eingesetzt werden. [60]

Im Jahr 2012 übergab Dahl schließlich die führende Leitung von Node an seinen langjährigen Kollegen Isaac Schlueter. Die inzwischen gewachsenen Community hatte zunächst seine Zweifel wie die Weiterentwicklung des Projekts ohne den Gründer weiterlaufen würde. Schlueter führte die Arbeit an der Plattform jedoch nahtlos fort und verbesserte in den darauffolgenden Versionen vor allem die Performance und Stabilität der JS-Umgebung. [61]

Nachdem Node in der Praxis zunächst kaum genutzt wurde änderte sich dies mit dem großen Release der Version 1.0, mit der auch immer mehr größere Unternehmen begannen auf die Plattform zu setzen, die Downloadzahlen schlagartig in die Höhe gingen und neue Frameworks wie Express, Socket oder Koa veröffentlicht wurden. [62]

Mittlerweile ist für die Weiterentwicklung der Laufzeitumgebung, die 2015 gegründete, unabhängige Node.js Foundation zuständig. So ist im Gegensatz zu den Anfangsjahren inzwischen keine einzelne Person mehr an der Spitze des Projekts sondern ein zwölfköpfiges Komitee, welches die regelmäßigen Releases der Plattform steuert. [61]

2.4.3.2 Architektur

V8 - Engine

Herzstück der Node.js-Plattform ist Googles V8-Engine. Erst durch den Einsatz der V8-Laufzeitumgebung, die ursprünglich für den Chrome-Browser entwickelt wurde, ist es möglich den JavaScript-Programmcode außerhalb von Webbrowsern auszuführen. [63] Die zugrunde liegende Engine selbst ist in C++ geschrieben und läuft daher auf verschiedenen Plattformen und Betriebssystemen. Zudem ist sie als Open-Source-Software für alle Entwickler zur eigenen Verwendung und Optimierung verfügbar, wodurch sie problemlos in jede beliebige C++-Anwendung integriert werden kann. [61]

Node.js dient dabei als Wrapper für die V8-JavaScript-Runtime, die so angepasst wird, dass die Engine außerhalb von Browsern besser arbeiten kann. So erweitert Node V8 beispielsweise um einige zusätzliche Schnittstellen, die vom ursprünglichen JavaScript-Sprachkern nur schlecht unterstützt werden und die Laufzeitumgebung für die Anwendung im Serverumfeld verbessert. [64]

Typisch für JavaScript wird der Programmcode nicht schon vor der Ausführung kompiliert, sondern die Dateien mit dem Quellcode beim Starten der Anwendung direkt eingelesen, wobei ein neuer Node.js-Prozess gestartet wird. Eine Besonderheit der V8-Engine hierbei ist es, dass der Code nicht direkt interpretiert wird, sondern vor der Ausführung zuerst in die Maschinensprache der zugrunde liegenden Hardware übersetzt wird. Auf Basis des kompilierten Maschinencodes wird schließlich die eigentliche Anwendung ausgeführt. [61]

Bei diesem sogenannten Just-in-Time-Compiler (*JIT*) wird, im Gegensatz zu Programmiersprachen wie C, der Code erst bei der Ausführung kompiliert. Dadurch verbessert sich unter anderem die Ausführungsgeschwindigkeit und der Code kann zudem, anders als bei den klassischen Ahead-of-Time-Compilern, für jeden User, je nachdem wie er die Anwendung nutzt, dynamisch optimiert werden. [65]

Event Loop und Non-Blocking I/O

Im Vergleich zu vielen anderen serverseitigen Sprachen setzt Node.js auf ein **Single-Threading Konzept**. Dies bedeutet, es steht nur ein einzelner Programm-Thread für die Applikation zur Verfügung, wodurch nur ein Befehl auf einmal ausgeführt werden kann. Das steht im Gegensatz zum Multi-Threading, bei dem Aufgaben in mehreren parallelen Threads ausgeführt werden.

Um den laufenden Prozess nicht bei einer Anfrage für die Folgenden zu blockieren setzt Node.js ein ausgelagertes Input/Output-System ein. [66] Dazu bietet Node einige sogenannte Nicht-Blockierende-Bibliotheken, dies sind Schnittstellen für Lese- oder Schreiboperationen zu Dateisystemen auf der Festplatte oder Datenbanken.

Da die Kernsprache JavaScript ist, zeichnet sich die Laufzeitumgebung darüber hinaus durch eine Event-gesteuerte Architektur aus. Node macht sich diese Eigenschaft in Form der sogenannten **Eventschleife** oder auch Event Loop zunutze, mit Hilfe derer sich skalierbare Server einfach programmieren und performant ausführen lassen. Im Zusammenspiel des Non-Blocking I/O und der Loop wird bei der Verarbeitung von Events die weitere Ausführung des Programmcodes nicht behindert. Node wird stattdessen einfach von dem nicht-blockierenden Interface benachrichtigt, sobald die entsprechende Lese- oder Schreiboperation beendet ist. Dies lässt sich analog zu einem Webbrowser sehen, der den Code informiert wenn beispielsweise ein “onclick”-Event eingetreten ist. [64]

Bei dieser Funktionsweise werden Callbacks eingesetzt, die bei eintretenden Ereignissen die Anfragen an das Betriebssystem auslagern, während Node die Kontrolle über die Anwendung behält. Wenn die externe Operation beendet ist, wird dann wiederum das Ergebnis an die Ereignisschleife übermittelt und die Event Loop sorgt dafür, dass die zugehörigen Callback-Funktionen ausgeführt werden. [61]

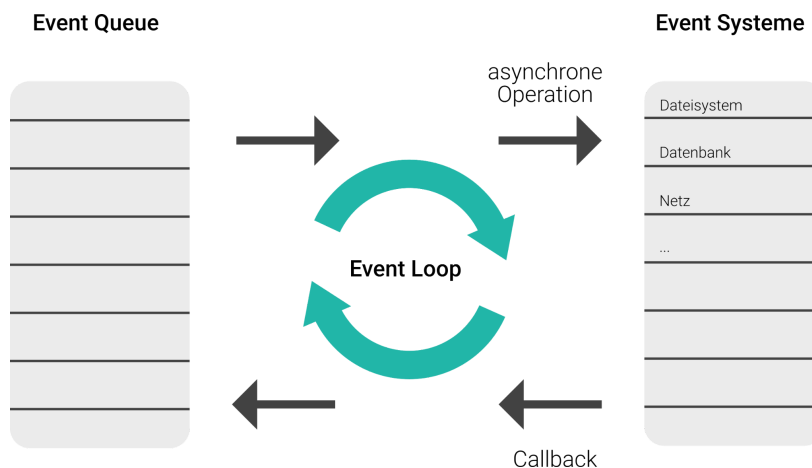


Abbildung 34: Auslagern von I/O-Operationen in die Eventschleife

In den ersten Versionen von Node.js wurde der Kern um die V8-Engine durch die Bibliotheken “libev” um die Eventschleife und “libeio” um den asynchronen Dateisystemzugriff ergänzt. Problematisch war dabei jedoch, dass die Plattform zunächst nur unter Linux problemlos lauffähig war, da im Gegensatz zur Engine, die bereits einige Jahre im Chrome-Browser unter Windows funktionierte, einige andere Komponenten umgeschrieben werden mussten.

Aus diesem Grund wurde ab der Version 0.6 mit *“libuv”* eine weitere Bibliothek eingeführt, mit der *“libev”* und *“libeio”* nicht mehr direkt eingebunden sondern abstrahiert werden. Der Einsatz von *“libuv”* machte es in Folge möglich, dass Node auch auf Windows-Systemen lauffähig ist und war ein weiterer wichtiger Baustein zur Verbreitung von Node und der systemunabhängigkeit der Umgebung. [61]

Wie zuvor beschrieben, besteht ein zentrales Konzept von Node.js darin, aufwendigere Input- und Output-Operationen in die Eventschleife auszulagern, wo diese dann asynchron abgearbeitet werden können. Bei eigenem Code hingegen ist der Entwickler selbst dafür verantwortlich, dass Programm so zu schreiben, dass Node.js nicht blockiert wird. [66]

Um das Problem, dass der einzelne Thread beispielsweise bei zeitintensiven Aufgaben blockiert wird, zu lösen, nutzt man die Möglichkeit der **asynchronen Programmierung** in JavaScript (siehe 1.5). Im Gegensatz zu einer synchronen Ausführung werden hier die Codezeilen nicht nacheinander sondern in einer anderen Reihenfolge, als sie geschrieben wurden, abgearbeitet. Dabei kann man sich die Architektur von Node zu nutze machen und aufwendige Aufgaben mit der Ereignisschleife verwalten. [10]

2.4.3.3 Module

Modularer Aufbau

Der Aufbau von Node.js ist im darauf ausgelegt, dass die Laufzeitumgebung aus verschiedenen Komponenten bestehen soll, die jeweils für einzelne, unterschiedliche Aufgaben zuständig sind. Die einzelnen Komponenten arbeiten über Schnittstellen zusammen und bilden so das Gesamtsystem. [61]

Dieses Konzept zieht sich durch die grundlegende Struktur, von der Kernbibliothek über Erweiterungen bis hin zur entwickelten Server-Anwendung. Nodes Core-Library ist dabei bewusst relativ kompakt gehalten und bietet lediglich Basis-Funktionalitäten für Interaktionen mit dem Betriebssystem oder für die Netzwerkkommunikation. Für nahezu alle weiteren denkbaren Module, die bei der Umsetzung einer Applikation benötigt werden, können Entwickler auf ein umfangreiches Sortiment an Drittanbietermodulen zurückgreifen (siehe NPM). Falls kein entsprechendes Paket verfügbar sein sollte, ist es außerdem möglich selbst Module für spezielle Anwendungsfälle zu schreiben und einzusetzen. [67]

Auch für die eigene Applikation macht es Sinn auf eine modulare Architektur zu setzen. Das bedeutet, dass der Entwickler keinen monolithischen Ansatz verfolgen, sondern die Anwendung in viele, kleinere Komponenten aufteilen sollte. Jedes Modul liegt dafür in einer eigenen Datei und enthält eine spezielle Funktionalität.

Die Modularisierung der App bringt dabei neben einer besseren Übersichtlichkeit weitere Vorteile mit sich: [61]

- Wiederverwendbarkeit von Teilkomponenten
- Bessere Testbarkeit
- Austauschbarkeit durch definierte Schnittstellen
- Verbesserte Stabilität und Fehleranfälligkeit

Basismodule

Node verfügt von Haus aus über eine Reihe von hilfreichen Basismodulen, auf die der Entwickler ohne externe Installation zurückgreifen kann. Wird ein oder mehrere Module davon benötigt, können diese einfach über die Funktion `require()` geladen werden. Diese erwartet einen String, der in der Regel den Namen der entsprechenden Komponente enthält. Besonders dabei ist, dass die Module auch ohne genaue Pfadangabe gefunden werden. [67]

```
1  const http = require('http');  
2  
3  var server = http.createServer();  
4
```

Abbildung 35: Einbinden des HTTP-Moduls über die `require`-Funktion und erstellen eines Servers

Dieses Beispiel zeigt, wie das HTTP-Modul über `require` eingebunden wird. Anschließend steht es dem Entwickler in seinem vollen Umfang zur Verfügung und dieser kann auf dessen Funktionen zugreifen, um beispielsweise wie hier einen HTTP-Server zu definieren.

- **HTTP Modul**

Da eines der häufigsten Einsatzgebiete der Laufzeitumgebung die Verwendung als HTTP-Server ist, zählt das HTTP-Modul zu den wichtigsten Kernkomponenten der Node-API. Der Server wird, wie oben zu sehen, über die Methode `createServer` des HTTP-Objekts erstellt. Das neue Server-Objekt hat so jedoch weder Funktionalität, noch eine Verbindung nach außen geöffnet. [64]

```

2
3  const server = http.createServer(function(req, res){
4      res.write("Hallo Welt!");
5      res.end();
6  });
7
8  server.listen(3000);
9

```

Abbildung 36: Einrichten des Servers und Absenden einer Response

Beim Aufruf von `createServer` eine anonyme Funktion als Callback übergeben. Hier kommt der in 1.5 beschriebene, wichtige Aspekt der asynchronen Abarbeitung von Aufgaben ins Spiel. Der Webserver reagiert auf Ereignisse, in diesem Fall auf eine Anfrage des Clients. Dafür wird eine Callback-Funktion übergeben, die jedoch erst ausgeführt, sobald ein solcher Request eintrifft. [61] Diese Funktion nimmt mit `req` und `res` zwei Objekte entgegen, mit denen die eintreffende Anfrage verarbeitet und auf diese reagiert werden kann. Hier wird der String `"Hallo Welt"` mit `res.send` als Response an den Client gesendet und anschließend die Verbindung wieder beendet. Zur Öffnung der Verbindung muss der Server noch an einen Port gebunden werden, wozu die `listen`-Funktion dient, die als Parameter die entsprechende Portnummer erwartet. [68]

- **Event Modul**

Wer bereits mit JavaScript im Browser programmiert hat, der ist mit Sicherheit schon mit dem Eventmodell des DOM in Berührung gekommen, welches auf Benutzerinteraktionen basiert. In Node.js dient das Event-Modul als Grundlage für die Implementierung eines Ereignismodells. Da das Eventmodell im Browser mit dem DOM verbunden ist, wurde in Node die Klasse *EventEmitter* entwickelt um eine entsprechende Funktionalität anbieten zu können. [6] Diese Klasse stellt mit `emit` und `on` zwei Funktionen bereit, über die mit Ersterer Events ausgelöst werden können und mit der Zweiten durch das Registrieren von Callback-Funktionen auf diese reagiert werden kann. [61]

- **OS Modul**

Dieses Modul stellt die Schnittstelle zum Betriebssystem dar. Über dessen Methoden lassen sich beispielsweise Informationen wie die zugrunde liegende Plattform, die Speicherauslastung oder die Anzahl und der Typ der verfügbaren Prozessoren abrufen. [68]

- **FS Modul**

Eine weitere nützliche Basiskomponente ist das Filesystem-Modul, das Node.js den Zugriff auf Dateien der Festplatte ermöglicht. Neben den üblichen Operationen Erzeugen, Umbenennen, Verschieben oder Löschen stehen auch Optionen zur Verfügung, um Dateien zu lesen oder zu schreiben. [21] Eine Besonderheit dieses Moduls stellt die Möglichkeit dar, alle Methoden synchron oder asynchron zu implementieren. Um auf mehrere Dateien gleichzeitig zuzugreifen und die Laufzeit zu verringern bietet es sich jedoch meistens an, asynchrone Funktionen einzusetzen. [64]

NPM

Obwohl Node im Kern bereits anwendungsfertige Module enthält, gibt es dennoch einige Anforderungen darüber hinaus, die noch nicht durch die relativ schlanken Basisfunktionen abgedeckt werden. Für viele verschiedene Anwendungsfälle existieren bereits Module, die von der Community geschrieben, gepflegt und in der Regel lizenzfrei zur Nutzung bereitgestellt werden. Diese reichen von einfachen Hilfsbibliotheken über Datenbanktreiber bis hin zu kompletten Applikations-Frameworks. Zum einfachen Einbinden in die eigenen Projekte und zur Verwaltung der Abhängigkeiten zwischen den einzelnen Modulen dient die Software *“Node Package Manager”*, oftmals auch als NPM abgekürzt. [68]

Der hilfreiche und inzwischen weit verbreitete Paketmanager wurde ab 2009 während der Entstehung von Node.js parallel vom späteren Projektleiter Isaac Schlüter entwickelt und ist schließlich ab der Version 0.6.3 seit 2011 fester Bestandteil der Laufzeitumgebung. [61] Mittlerweile steht hinter der Entwicklung und Verwaltung die npm.inc, die seit Anfang letzten Jahres zu GitHub und somit zum Microsoft-Konzern gehört. [69]

Heute stehen über 350.000 Open-Source-Pakete zur freien Anwendung im NPM-Repository zur Verfügung. Darüber hinaus wird für private Pakete eine kommerzielle Version angeboten. Der Paketmanager ist dabei nicht mehr nur auf externe Module für die Backend-Entwicklung mit Node beschränkt, denn inzwischen lassen sich auch nahezu alle beliebigen Frontend-Frameworks wie auch React, Vue oder Angular, Bibliotheken und zusätzliche Erweiterungen mit einem einfachen Befehl in der Kommandozeile dem aktuellen Projekt hinzufügen.

Pakete können mittels NPM auf zwei Arten installiert werden. Während bei der lokalen Installation, die normalerweise genutzt wird, ein Modul direkt in den Kontext einer Applikation eingebunden wird, kann dies bei der globalen Installation systemweit zur Verfügung gestellt werden. Da im Rahmen der lokalen Installation die Erweiterungen in ein Unterverzeichnis des

Projekts hinzugefügt werden, ist es möglich für jede Applikation eine individuelle Auswahl an Modulen je nach Anwendungszweck zu nutzen.

Um die lokale Installation eines entsprechenden Moduls durchzuführen muss dem Package Manager in der Kommandozeile neben den Parameter *“install”* der Name des jeweiligen Moduls übergeben werden. Ohne zusätzliche Angaben wird so immer die aktuellste Version in ein entsprechendes Unterverzeichnis integriert. In manchen Fällen können bestimmte Erweiterungen von anderen Modulen abhängen. Der NPM kann in solchen Fällen die Abhängigkeiten automatisch auflösen und installiert die benötigten Zusätze automatisch. Bei erfolgreicher Installation lassen sich die Module dann, wie schon zuvor beschrieben, über die Funktion *“require”* zur Node-App hinzufügen und anschließend nutzen. [68]

2.4.4 NodeJS Frameworks

Frameworks dienen dem vereinfachen bestimmter Aufgaben, wie zum Beispiel das Routing, und beschleunigen so den Entwicklungsprozess. Die drei verbreitetsten Node-Frameworks sind Express, KOA und Socket.IO.

2.3.4.1 Express

Express ist das populärste Node-Framework und dient auch als Bibliothek für weitere Frameworks. Es ist ein minimalistisches und flexibles Modul, was die Entwicklung von Anwendungen im gesamten Middleware-Bereich, besonders auf der Serverseite, erleichtert und somit die Anwendungsentwicklung beschleunigt. Es verbessert die Kommunikation zwischen Server und Client und erleichtert unter anderem das Handhaben von Requests, das Routing und das Rendern von HTML. Zudem gibt es zahlreiche Erweiterungen, z.B. für das Arbeiten mit Cookies, Sessions oder Logins. Es arbeitet nach dem Model-View-Controller-Architekturmuster. [70]

Express kann man bei allen Arten von Projekten einsetzen, da es sich in allen Bereichen bewährt hat und auch kompatibel mit vielen weiteren Frameworks ist.

Vorteile:

- Große Community (Nützlich beim Lernen, Fehlersuche, etc.).
- Es ist leicht zu erlernen.
- Schnelle Anwendungsentwicklung.
- Einfache Einbindung externer Services und Middleware.
- Benutzerfreundlicher Projektgenerator.
- Mit vielen datenbanken kompatibel.
- Gutes Error-Handling.
- Gute Sicherheits-Features.

Express wird eingesetzt bei: *Twitter, Uber, IBM* [71]

2.4.4.2 KOA

KOA wurde von den Express-Machern entwickelt. Es ist ein kleines und robustes Framework für Web-Services und APIs und macht das Programmieren von Servern durch die zur Verfügung gestellten Funktionen schnell und angenehm. Durch das Verwenden von “*async*”-Funktionen müssen Callbacks nicht mehr behandelt werden, wodurch das Error-Handling und die Übersichtlichkeit verbessert werden. Es verwendet Generatoren und ist dadurch nicht mit anderen Frameworks, auch nicht mit Express, kompatibel. Eine KOA-Anwendung selbst ist ein Objekt, das eine Reihe von Funktionen enthält, die als Reaktion auf Requests stapelartig zusammengesetzt und ausgeführt werden. [72]

KOA eignet sich für die Entwicklung von Web-Services und APIs und umfangreiche Web-Anwendungen, welche eine gute Performance benötigen.

Vorteile: [71]

- Sehr leicht (Nur 550 Zeilen Code)
- Starke Performance
- Minimiert das Auftreten von Errors im gesamten Stack
- Effizienter Umgang mit HTTP-Middleware.
- Verbessertes Error-Handling.
- Programmfluss gut nachvollziehbar (gut wartbarer Code)
- Gute Sicherheitsoptionen

Nachteile:

- Kleine Community
- Nicht mit anderen Frameworks kompatibel

KOA wird eingesetzt bei: *Indeed* [73]

2.4.3.3 Socket.IO

Socket.IO ist eine Bibliothek, die bidirektionale und ereignisbasierte Echtzeitkommunikation zwischen dem Browser und dem Server ermöglicht. [74] Diese besteht aus einem clientseitigen und serverseitigen Framework. Clientseitig gibt es ein Framework für den Browser (JavaScript), aber auch für andere Programmiersprachen wie zum Beispiel Java, Python oder Swift. Es dient rein der schnellen und stabilen Client-Server-Kommunikation mittels einer WebSocket-Verbindung. WebSocket ist ein Kommunikationsprotokoll, das einen Vollduplex-Kanal zwischen Server und Client bereitstellt und so den Echtzeit-Datenfluss in beide Richtungen einer Verbindung ermöglicht. Die Verbindung ist dabei immer offen und so ist beim entstehen neuer Daten kein „Refresh“ nötig. Daten werden gepusht, ohne dass der Client diese erst über einen Request anfragen muss. Falls keine WebSocket-Verbindung möglich ist, dann wird automatisch eine HTTP-Verbindung eingesetzt. [74] Es wird meistens in Verbindung mit Express genutzt.

Socket.IO eignet sich für Echtzeit-Anwendungen mit mehreren beteiligten Clients, bei denen die Kommunikations-Performance zwischen Client und Server in beide Richtungen sehr stark und schnell sein muss, also z.B. Chatrooms, Videokonferenzen oder Online-Multiplayer-Spiele.

Vorteile:

- Gute Dokumentation
- Sehr stabile, schnelle und Performante Client-Server-Verbindung.
- Sehr schnelle und performante Verbindung.
- Automatisches Wiederverbinden nach Verbindungs-Verlust.
- Multiplexen und Broadcasten möglich.
- Paket-Buffering, wenn Internetverbindung unterbrochen ist.
- Mit anderen Frameworks kompatibel.

Socket.IO wird eingesetzt bei: *Patreon* [75]

2.4.5 Datenbanken

Eine Datenbank speichert eine organisierte Sammlung von strukturierten Informationen oder Daten und stellt sie zur Benutzung bereit. Die einzelnen Daten werden mit Metabeschreibungen mittels "*Data Dictionary*" und Informationen für ihre Verarbeitung versehen. Damit die Verarbeitung der Daten möglichst effizient ist, werden die Daten in den meisten Datenbanken in Zeilen und Spalten von Tabellen modelliert, wo sie dann abgerufen, verwaltet, modifiziert, aktualisiert, kontrolliert und organisiert werden können. [76]

Die ersten Datenbanken erarbeitete man in den 1960er Jahren, dabei handelte es sich um hierarchische- sowie Netzwerk-Datenbanken, die zum Speichern und Bearbeiten der Daten genutzt wurden. Da diese jedoch zu simpel und unflexibel waren, begann man mit der Entwicklung **relationaler Datenbanken**. Diese waren deutlich leistungsfähiger und verbreiteten sich deshalb schnell. Dazu gehörte vor allem die Datenbanksprache SQL von Oracle. [76] Anfang der 2000er Jahre entstanden NoSQL-Datenbanken, welche anstatt relationaler Datenbanken hauptsächlich alternative Datenbankmodelle nutzen. Der Grund zum Wechsel auf **NoSQL-Datenbanken** liegt an der höheren Geschwindigkeit und Verarbeitung der riesigen Menge an unstrukturierter Daten im Internet. Heute verwendet man Datenbanken vor allem als Unternehmenssoftware, wo sie für viele Unternehmensprozesse genutzt werden. Im Hintergrund jeder Webseite oder Software hat die Datenbank die Aufgabe, die Informationen für den Betrieb der Anwendung bereitzustellen. Für Web- und Softwareentwicklung kommen meist relationale SQL-Datenbanken zum Einsatz. [77]

Eine Datenbank besteht aus einem Datenbankmanagementsystem, dem Data Dictionary und der Datenbank, also die Menge der Daten selbst. Das **Datenbankmanagementsystem (DBMS)**, ist verantwortlich für die Datenverwaltung in einem Datenbanksystem. Es erkennt aus der Anfrage welche Daten benötigt werden, entnimmt die gewünschten Informationen der Datenbank selbst und übermittelt diese dann an den Anfragenden. Zusätzlich ist das DBMS für den Schutz des Datenbestandes vor dem Zugriff durch Unbefugte verantwortlich. Dies geschieht unter anderem durch das beschneiden der Zugriffsrechte. [77] Ein DBMS ist eine Software, die frei oder käuflich erwerbbar sein kann, z.B. MySQL, Oracle 10g oder DB2. [78]

Relationale Datenbanken

Beim relationalen Datenbankmodell werden alle Informationen in Tabellen abgelegt. Eine relationale Datenbank besteht aus einer Sammlung von Tabellen, die miteinander in Beziehung stehen. [77] Eine relationale Tabelle besteht aus einem Relationsschema, bei der ein Name für das Schema definiert wird, Attribute für die Struktur der zu speichernden Daten und der Relation,

welche in einzelne Tupel unterteilt ist. Jedes Tupel beschreibt einen Datensatz. Der Wert jedes Attributs muss atomar sein, er kann somit nur einen, oder keinen Wert besitzen. Mittels Wertebereiche wie *integer*, *string* oder *boolean* wird vorgegeben, welche Werte für ein Attribut zugelassen sind. Für die eindeutige Identifizierung eines bestimmten Tupels nutzt man einen Primärschlüssel. Dieser darf nur einmalig in der Datenbank vorkommen und sein Wert darf sich nicht verändern. Zusätzlich gibt es einen Fremdschlüssel, welcher ebenfalls ein Attribut einer Relation ist und auf einen Primärschlüssel in derselben oder einer anderen Relation verweist, um so Relationen einer Datenbank miteinander zu verbinden. [79]

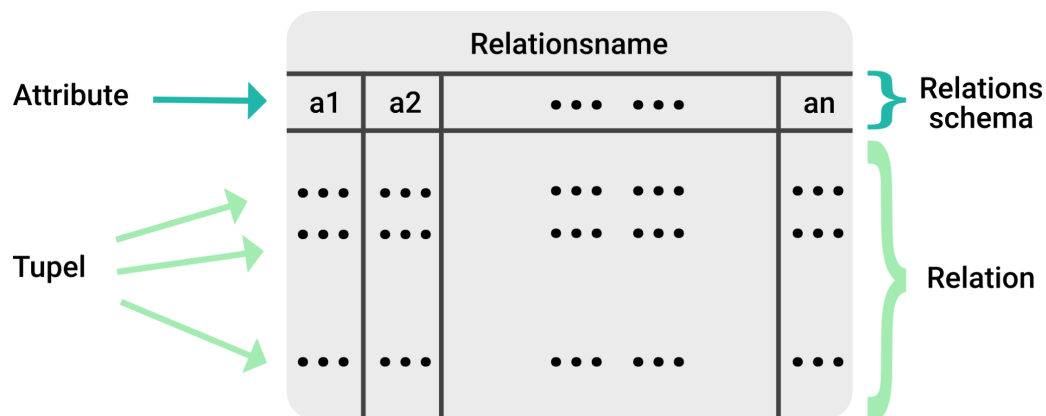


Abbildung 37 : Relationsschema

Ein Relationsschema für einen Studenten könnte beispielsweise sein:

- Student (matrikelNr, vorname, nachname , semester)

Das entsprechende Tupel für den Studenten würde so aussehen:

- Student (187654, max, mustermann,7)

2.4.5.1 SQL

Die Abkürzung SQL steht für Structured Query Language und ist eine Sprache zum Umgang mit relationalen Datenbanken. SQL ist standardisiert, besitzt eine einfach gehaltene Syntax, welche an die englische Umgangssprache angelehnt ist und basiert auf relationaler Algebra. Die meisten Datenbanksysteme verfügen über SQL-Schnittstellen, womit ein Zugang auf deren Daten ermöglicht wird. SQL-Datenbanken kommen vor allem im Rechnungswesen, in der Webentwicklung, der Content-Verwaltung, im Controlling oder bei Warenwirtschaftssystemen zum Einsatz. [80] Viele Internetauftritte, Onlineshops und Internet-services basieren außerdem auf einer Kombination von PHP-Skripten und SQL-Datenbanken. Dadurch ergibt sich die Möglichkeit, dynamische Webinhalte auf Webservern zu erzeugen. [81]

Es kann zwischen vier Kategorien von SQL-Befehlen unterschieden werden. [80][81][82]

- **DCL (Data Control Language)**

Die Datenbankkontrollsprache wird verwendet, um Berechtigungen zu vergeben mit *GRANT* um einen Nutzer beispielsweise die Erlaubnis zum Lesen oder verändern einer Tabelle zu geben. Um Rechte zu entziehen verwendet man den *REVOKE*-Befehl.

- **DML (Data Manipulation Language)**

Mit DML-Befehlen lassen sich Datensätze bearbeiten, einfügen oder löschen. Zur Datenmanipulationssprache gehören Befehle wie *UPDATE* zum Aktualisieren, *INSERT* zum Einfügen von Tupel und *DELETE* zum Löschen.

- **DQL (Data Query Language)**

Bei der Datenabfragesprache handelt es sich lediglich den Befehl *SELECT*. Wenn man ihn mit anderen Attributen kombiniert, können eine Vielzahl an Abfragen ausgeführt werden.

- **DDL (Data Definition Language)**

Mittels der Datendefinitionssprache ist es unter anderem möglich Tabellen anzulegen oder zu löschen, durch den *CREATE TABLE* bzw. den *DROP TABLE* Befehl. Der Befehl zum Ändern von Tabellen lautet *ALTER TABLE*.

Zu jedem Attribut einer Tabelle wird bei SQL außerdem ein passender standardisierter Datentyp angegeben. Zu den wichtigsten Standardtypen gehören:

- **Integer** für ganze Zahlen
- **Numeric** und **Decimal** für Festkommazahlen
- **Float**, **Real** und **Double** für Gleitkommazahlen
- **Charakter** und **Text** für Zeichenketten
- **Date** und **Time** für Datums- und Zeitangaben

Die Grundform der Datenselektion womit sich die drei relationalen Grundoperationen Selektion, Projektion und Join darstellen lassen: [83]

```
SELECT <Attribut-1, Attribut-2, ..., Attribut-n>  
FROM <Relation-1, Relation-2, ..., Relation-m>  
WHERE bedingung
```

Folgendes Relationsschema „*Student*“ soll als Beispiel dienen:

matrikelNr	vorname	name	semester
184968	Nils	Mattmüller	6
184888	Eric	Meyer	6
185697	Tim	Peters	6
187654	Max	Mustermann	7

Projektion:

Eine Projektion erfolgt, indem man in der ersten Zeile Attribute angibt, wodurch nur diese Spalten ausgegeben werden. Ein einfacher SQL-Befehl, um sich ausschließlich die Matrikelnummern anzeigen zu lassen:

```
SELECT matrikelNr  
FROM Student
```

Selektion:

Mit einer Selektion ist es möglich bestimmte Zeilen einer Tabelle durch Angabe einer Selektionsbedingung auszugeben. [83] Um sich die Nachnamen aller Studenten anzuzeigen welche in Semester 7 sind:

```
SELECT *  
FROM Student  
WHERE semester = 7
```

Join:

Um einen Join durchzuführen werden zwei Attribute von zwei unterschiedlichen Relationen in Beziehung gesetzt, um verschiedene Tabellen zu verknüpfen. [83] Eine weitere Tabelle „hoert“: hoert (matrikelNr, vorlesungsNr), welche die Matrikelnummer und die Nummer einer Vorlesung welcher dieser Student besucht, beinhaltet:

matrikelNr	vorlesungsNr
184968	6003
184888	6009
185697	6003
187654	7006

Ein SQL-Befehl zum Verknüpfen der Tabellen „Student“ und „hoert“:

```
SELECT *  
FROM Student a, Buch b  
WHERE a.matrikelNr = b.matrikelNr
```

Die Verknüpfte Tabelle würde nun so aussehen:

matrikelNr	vorname	name	semester	vorlesungsNr
184968	Nils	Mattmüller	6	6003
184888	Eric	Meyer	6	6009
185697	Tim	Peters	6	6003
187654	Max	Mustermann	7	7006

2.4.5.2 No-SQL

No-SQL (*Not only SQL*) sind Datenbanken, welche eine Ergänzung zu relationalen SQL-Datenbanken darstellen. Stattdessen verwendet No-SQL eine Reihe alternativer Datenbankmodelle. Grund für die Einführung von No-SQL Datenbanken waren die riesigen unstrukturierten Datenbestände im Internet, auf welche sehr viele Nutzer gleichzeitig, mit hoher Geschwindigkeit zugreifen wollen. [84]

Zur Lösung dieser Aufgabe gibt es vier verschiedene Lösungsansätze:

- **Dokumentenorientierte Datenbanken** werden für große, strukturlose oder unterschiedlich strukturierte Datensätze verwendet. [84] Die Daten werden als Dokumente mit unterschiedlicher Länge gespeichert.
- In **Key-Value-Datenbanken** werden Daten in Form von Schlüssel-Werte-Paaren gespeichert. Eine Key-Value-Datenbank besteht aus der "Key"-Spalte welche die Identifikatoren um den Wert zu bestimmen. Die "Value"-Spalte beinhaltet dann die jeweiligen Werte. Vorteil sind sehr kurze Zugriffszeiten, selbst bei großen Datenmengen. Daten werden im Hauptspeicher gehalten, wodurch man die Zeit beim Lesen von bzw. Schreiben auf der Festplatte spart. [84] Beispiele für Schlüssel-Wert-Datenbanken sind Riak and Berkeley DB. [85]
- **Graphdatenbanken** bilden Beziehungen der Daten mithilfe von Kanten und Knoten untereinander ab. Beziehungen werden durch Knotenpunkte und deren Verbindungen zueinander dargestellt. Diese werden hauptsächlich im Bereich Social Media genutzt, da Graphdatenbanken bei stark vernetzten Informationen, wie hier zwischen den Nutzern, eine bessere Performance haben. Beispiele für Graphdatenbanken sind Neo4J and Giraph. [85]
- Desweiteren gibt es **spaltenorientierte Datenbanken** wie etwa Cassandra und HBase. [85] Hier werden die Daten in Spalten gespeichert, wodurch sich kürzere Lesezeiten und eine höhere Leistungsfähigkeit ergibt. Sie sind für verteilte Datenhaltung in analytischen Informationssystemen, Suchmaschinen oder sozialen Netzwerken geeignet. [84]

2.5 Architekturmodelle

Ein **Stack** (*Stapel*) beschreibt den Technologie-Einsatz einer Web-Anwendung. Der Stack stellt dabei den Kern dar und kann durch weitere Technologien ergänzt werden. Es sind viele Stack-Varianten denkbar, da z.B. viele clientseitige- sowie serverseitige Frameworks zur freien Auswahl stehen. Im Folgenden sollen drei der beliebtesten Ansätze kurz vorgestellt werden.

2.5.1 MERN

Bestehend aus:

- **MongoDB**, eine dokumentenorientierte Datenbank, die für die Arbeit mit JavaScript und Daten in Form von JSON (intern BSON) optimiert ist
- **Express**, ein serverseitiges Node.js Framework
- **React**, ein clientseitiges JavaScript Framework
- **Node** als JavaScript Webserver

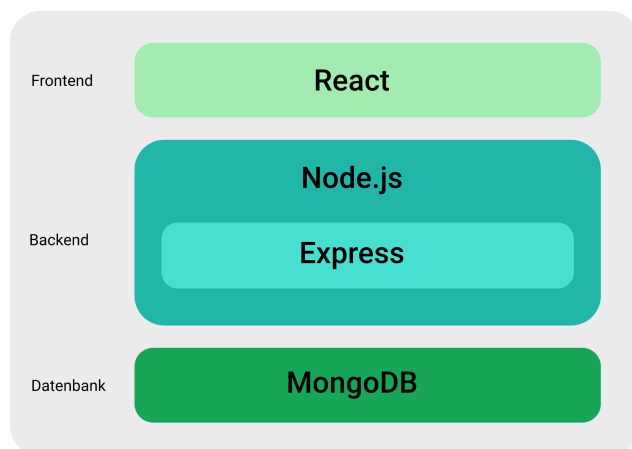


Abbildung 38: Graphische Darstellung des Aufbaus einer MERN-Anwendung

Daten werden von React in Form von JSON-Dokumenten am Frontend erzeugt und an den Express-Server gesendet. Dort können Sie mittels MongoDB direkt in dieser Form gespeichert werden. Ähnliches gilt bei Daten-Anfragen: Express kann die Daten direkt in JSON-Form auslesen und so an den Client senden, wo sie von React verarbeitet werden. Dieser natürliche Datenfluss erleichtert die Programmierung und das Debugging enorm. MERN ist ideal für für JSON-lastige, cloud-native Anwendungen mit dynamischen Web-Interfaces. [86]

2.5.2 MEAN

Bestehend aus: [87]

- **MongoDB**, einer dokumentenorientierte Datenbank
- **Express**, ein serverseitiges Node.js Framework
- **Angular**, ein clientseitiges JavaScript Framework
- **Node** als JavaScript Webserver

Diese Architektur funktioniert wie der MERN-Stack, doch nutzt das mächtigere und umfassendere Angular als Frontend-Framework.

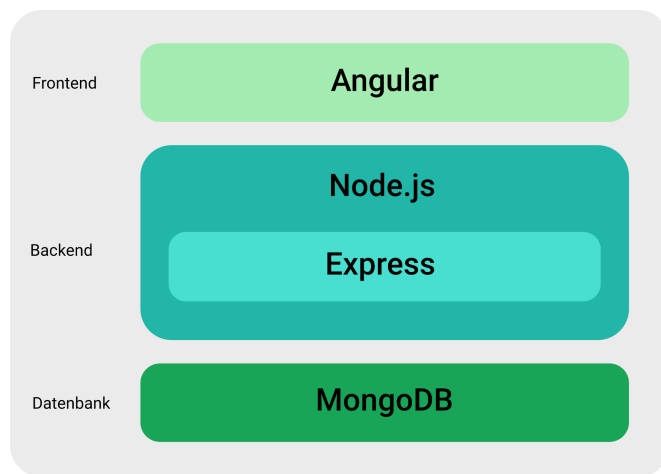


Abbildung 39: Graphische Darstellung des Aufbaus einer MEAN-Anwendung

2.5.3 PERN

Bestehend aus: [89]

- **PostgreSQL**, eine objektrelationale Datenbank
- **Express**, ein serverseitiges Node.js Framework.
- **React**, ein clientseitiges JavaScript Framework.
- **Node**, als JavaScript Webserver.

Diese Architektur nutzt PostgreSQL als Datenbank. Daten werden in tabellarischer Form gespeichert. PostgreSQL ist weitgehend konform mit dem SQL-Standard und stellt somit den Großteil der Funktionen und des Verhaltens davon zur Verfügung. Es ist Open Source und das Angebot an Erweiterungen ist groß. [90] Der Stack ist zuverlässig, stabil, skalierbar sowie sicher und für komplexe, datengetriebene Anwendungen entwickelt. [91]

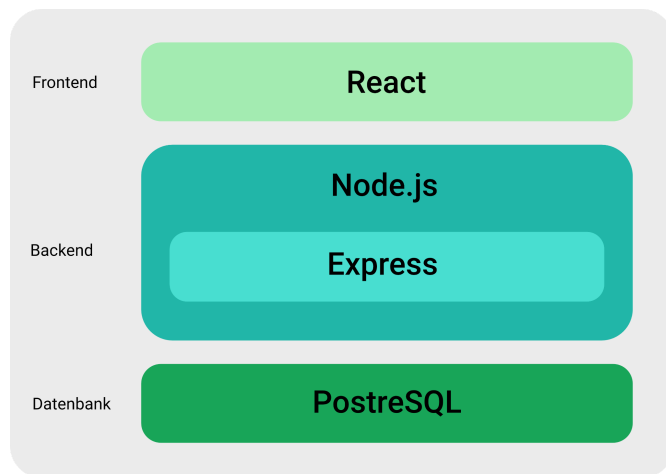


Abbildung 40: Graphische Darstellung des Aufbaus einer PERN-Anwendung

3. FULL STACK PRAXISPROJEKT “SKIMP”

Bei dem Praxisprojekt geht es darum, das erlangte Wissen durch praktisches Anwenden zu vertiefen und zu festigen. Die Anwendung muss aus einer dynamischen Clientseite und einem Server mit Datenbank bestehen, welche über das Internet miteinander kommunizieren.

3.1 Beschreibung

Skimp (*knausern, sparen*), ein Internetportal, in dem man personalisierte Rabattcodes und Gutscheincodes teilen kann, die man zum Beispiel bei einer Bestellung mitgeliefert bekommt und selbst nicht nutzen würde, zum Beispiel ein Coupon mit einer Ermäßigung für das nächste Mal Tanken, welche ein zu frühes Verfallsdatum hat, um es selbst einlösen zu können.

3.2 Anforderungsanalyse

Bei der Anforderungsanalyse werden die Anforderungen an das zu entwickelnde System ermittelt, strukturiert und geprüft. Das Ergebnis bildet die Grundlage für die spätere Projektumsetzung.

3.2.1 Recherche

Ähnliche Anwendungen gibt es bereits in vielen Formen. Allerdings dienen diese Anwendungen nicht dem Teilen personalisierter Rabattcodes, sondern dem Teilen allgemeiner Rabattcodes auf welche jeder Zugriff hat, z.B. solche welche aus einer Werbung stammen. Eine Anwendung, die unserer Idee entspricht konnten nicht gefunden werden und existiert somit unseren bisher in dieser Form zufolge nicht.

Folgende Anwendungsformen gibt es für den bereits genannten Anwendungsbereich:

In Form eines Chrome-Browser-AddOns gibt es beispielsweise “*honey*” von PayPal. Dies installiert man im Browser und anschließend trackt es den Client beim Shopping und wenn er im Shop zur Kasse geht, dann prüft honey, ob es für den aktuellen Shop einen Coupon gibt und informiert anschließend den Nutzer mittels eines Popup-Fensters. Solche AddOns gibt es auch shop-spezifisch, wie zum Beispiel “*Amazon Coupons*”, was ähnlich wie honey arbeitet, nur

beschränkt auf den Amazon-Shop. Es gibt zudem einige Internetseiten, auf der Rabattcodes aufgelistet sind wie beispielsweise *loveminty.de*. Hier kann man die Codes nach vielen Kategorien (Shop-Name, Kategorie, Art,...) sortieren, es gibt eine Suchfunktion und neue Coupons sind sehr schnell und einfach angelegt. Der Kunde sieht bei jeden Rabattcode sofort: Wert, % oder €, wie oft er bereits eingelöst wurde, kurze Beschreibung, wann er zuletzt eingesetzt wurde, Ablaufdatum und einen *“Code anzeigen”*- Button. Mittels Anlegen eines Accounts ist es möglich, Codes auf eine Merkliste zu setzen oder zu favorisieren. Die Seite bietet aber auch die Option als Browser-AddOn installiert zu werden und bietet so die gleichen Funktionen wie honey.

Zudem gibt es Apps und mobile Applikationen wie *“Mein Deal”*. Sie sind in der Funktion ähnlich wie loveminty.de und kann einem zusätzlich Push-Nachrichten senden, wenn es neue Coupons gibt, diese können den Einkauf des Clients allerdings nicht tracken und dementsprechende Coupons vorschlagen.

3.2.2 Anforderungen

Pflichtanforderungen

Als Nutzer der Webapp möchte ich beim Aufruf alle verfügbaren Coupons angezeigt bekommen, um eine Übersicht der verschiedenen Angebote zu erhalten.

Als Nutzer von Skimp möchte ich eigene Codes, die ich nicht mehr benötige, hinzufügen können, um diese mit anderen Nutzern zu teilen.

Als Nutzer der Seite möchte ich alle Inhalte auf kleineren Displays ohne zu zoomen erkennen können, damit ich auch auf meinem Smartphone die Anwendung bedienen kann.

Als Verwalter der Anwendung möchte ich, dass Formulareingaben vor dem Absenden überprüft werden, damit die Coupons korrekt in der Datenbank gespeichert werden.

Als Nutzer der Seite möchte ich die Gutscheine einlösen können, um bei einem Einkauf in einem Online-Shop den Code nutzen zu können.

Als Nutzer der Seite erwarte ich, dass bereits eingelöste Coupons gelöscht werden, damit ich nur noch gültige Codes in der App vorfinde.

Als Seitenbetreiber möchte ich, dass das User Interface der Anwendung ansprechend gestaltet ist, um den Besuchern ein positives Nutzererlebnis zu bieten.

Als Nutzer möchte ich schnell und ohne Verzögerungen durch die Webanwendung navigieren können.

Als Nutzer der Anwendung möchte ich auf der Seite nach verschiedenen Shops suchen können, um schnell einen passenden Coupon zu finden.

Als Verwalter der Seite erwarte ich, dass die Inputfelder des Formulars zurückgesetzt werden, nachdem ein Coupon hinzugefügt wird, damit der Kunde die Eingaben nicht selbst löschen muss.

Optionale Anforderungen

Als Seitenbetreiber möchte ich, dass die Coupons ab einer größeren Anzahl progressiv nachgeladen werden, um die Wartezeit zu Beginn zu verringern.

Als Nutzer der Seite möchte ich, dass die Nutzer den Code eines eingelösten Gutscheins per Buttonklick in die Zwischenablage kopieren können, damit beim späteren Einfügen keine Fehler entstehen.

Als User erwarte ich eine Option für manuellen Darkmode, damit ich die UI nach meinen Displaypräferenzen anpassen kann.

Als Seitenbetreiber möchte ich, dass abgelaufene Coupons automatisch aus der Datenbank gelöscht werden, damit den Nutzer keine ungültigen Gutscheincodes angezeigt werden.

Als Nutzer will ich direkt sehen, ob die Einlösung des Coupons an bestimmte Bedingungen geknüpft (z.B Mindestbestellwert) geknüpft ist, damit ich erkenne ob ein Einlösen lohnenswert ist.

Als Verwalter der Seite möchte ich eine Progressive Web App, damit die Nutzer diese auf dem Endgerät Installieren können.

Als Nutzer möchte ich einen User-Account anlegen können, um Coupons zu speichern oder einen Überblick über meine hinzugefügten oder eingelösten Coupons zu haben.

3.3 Technologie Stack

Für die Umsetzung der Anwendung wurde die in 2.5.1 beschriebene **MERN-Architektur** eingesetzt. Aufgrund des einfachen Set-Ups, der guten und verständlichen Dokumentation sowie der großen Community inklusiver zahlreicher Tutorials und Beispiele fiel die Wahl auf das Framework React. Die Node.js-Basis auf der Serverseite soll mit der Datenbasis MongoDB verknüpft werden, für deren Einsatz insbesondere die JSON-ähnliche Syntax ausschlaggebend war, die den kompletten JavaScript-Stack perfekt ergänzt. Im Folgenden werden alle eingesetzten Technologien und deren Grundlagen für die Implementierung auf der Client- und Serverseite genauer erläutert.

3.3.1 Frontend

Für die Entwicklung des Frontends der Applikation wurde neben den klassischen Technologien HTML, für die Grundstruktur und der Auszeichnungssprache CSS, für das Design der einzelnen Komponenten, das Framework React eingesetzt, welches bereits in 2.3.2.1 vorgestellt wurde. Um spezifische Probleme bei der Entwicklung zu lösen oder die Syntax des Codes zu vereinfachen kamen des weiteren einige zusätzliche Module und Hilfsbibliotheken für das Framework zum Einsatz, die über den Node Package Manager im Anwendungsverzeichnis installiert wurden.

3.3.1.1 React

Da die Architektur und die Grundlagen von Facebooks Library bereits im vorherigen Kapitel bei der Vorstellung der verschiedenen Frontend Frameworks beleuchtet wurden, sollen nun hier genauer einige Konzepte erläutert werden, die bei der Implementierung des Frontends der Anwendung von Bedeutung sind.

Create-React-App

Mit Hilfe des Befehls *“npx create-react-app”* lässt sich einfach über die Kommandozeile des Rechners oder des Code-Editors das Grundgerüst einer React Single-Page-Applikation automatisch herunterladen. So wird über diese Anweisung ein Build-Prozess ausgelöst, der den bereits ausführbaren Basis-Code zusammen mit einigen Abhängigkeiten einrichtet, wodurch sich einiges an Aufwand für das Erstellen der Verzeichnisse und Einbinden der Bundler und Präprozessoren vermeiden lässt. Dafür wird automatisch eine *“package.json”*-Datei generiert, die neben Metadaten des Projekts, wie Name oder Versionsnummer, auch die Dependencies

enthält, sowie Skripte, mit denen die Ausführung der App einfach über das CLI gestartet werden kann. [92]

- **Babel**

Bei Babel handelt es sich um einen JavaScript-Präprozessor, der ursprünglich dazu entwickelt wurde, aktuellen ECMAScript-Code in abwärtskompatible Syntax umzuwandeln, um diesen auch in älteren Browsern oder Umgebungen ausführen zu können. Im Zusammenhang mit React-Applikationen ist das Tool von Bedeutung um die typische JSX-Syntax vor dem Rendern der UI-Elemente im Browser in herkömmliches JavaScript zu konvertieren. [93]

- **Webpack**

Webpack ist ein sogenannter Modul-Packer, der dazu dient, unseren React-Code einschließlich statischer Assets wie Bilder oder CSS-Dateien zu bündeln und für die Ausführung im Browser bereitzustellen. Da man in der Regel keine unübersichtliche, monolithische JavaScript-Datei möchte, sondern den Code in mehrere Module aufsplittet, die über *“import”* und *“export”*-Statements miteinander verknüpft werden, kann Webpack genutzt werden, um die Abhängigkeiten untereinander zu analysieren und zu verbinden um schließlich eine ausführbare JS-Datei zu generieren. Dies kann hilfreich sein, da nativ von JavaScript kein Code-Splitting unterstützt wird. [94]

Komponenten

React ist für seinen komponentenbasierten Aufbau bekannt, der erlaubt, die Benutzeroberfläche in unabhängige, wiederverwendbare Teile aufzubrechen. Grundlegend sind Komponenten JavaScript-Funktionen, die sowohl die Logik, als auch die UI enthalten.

Die UI, mit der die jeweilige Komponente dargestellt werden soll, befindet sich dabei im *“return”*-Statement der Funktion und ist typischerweise in der HTML-artigen Syntax JSX geschrieben. Innerhalb des JSX-Codes kann man nun auf Variablen und JavaScript-Funktionen zugreifen. [95]

Verhalten und Aussehen lassen sich über zwei Wege beeinflussen:

- **Properties**

Die Eigenschaften, auch sogenannte Properties, werden beim Erzeugen der Komponente von außen als optional als Funktionsargument übergeben. Dabei handelt es sich um ein Objekt, das üblicherweise “*props*” genannt wird und den Namen und Wert des jeweiligen Properties enthält. Properties sind innerhalb der Komponente dann unveränderlich, können allerdings jederzeit extern neu gesetzt werden. [96]

```
2
3  const Component = (props) =>{
4      return(
5          <h1>Hello {props.name}</h1>
6      );
7  }
8
```

Abbildung 41: Grundlegende implementierung einer React-Komponente als Lambda-Funktion

- **State**

Des Weiteren gibt es noch den Zustand einer Komponente, der in React auch als State bezeichnet wird. Dieser lässt sich am besten als der aktuelle Ist-Zustand der Applikation oder der Komponente beschreiben. Hierbei handelt es sich um ein Objekt, das nur innerhalb der Komponente sichtbar ist, dafür allerdings, im Gegensatz zu den Properties von der Komponente selbst verändert werden kann. [96]

Sowohl Änderungen der Eigenschaften von außen, als auch in der Komponente am State, führen dazu, dass die Komponente im Browser neu gerendert wird.

Hooks

Während sich das “*State*”-Konzept und weitere Features früher nur mit Hilfe dem Schreiben von Klassen realisieren ließ und gerade für Anfänger recht schwer zu erlernen war, ist es ab Version 16.8 von React möglich diese Funktionen mit den sogenannten Hooks umzusetzen. Eine der größeren Schwierigkeiten der älteren React-Versionen war, die Umsetzung eines wiederverwendbaren Verhaltens an eine Komponente zu binden. Dafür mussten speziell Pattern wie “*higher order components*” eingesetzt werden, die eine Umstrukturierung der Komponenten unumgänglich machten und zu einer unübersichtlichen, inkonsistenten Syntax führten. [97]

Bei diesem Problem setzen die Hooks an. Sie machen den State wiederverwendbar, ohne dass die hierarchische Struktur der Komponente geändert werden muss. Inzwischen gibt es eine Vielzahl verschiedener Hooks, wobei sich für spezielle Anwendungsfälle auch eigene schreiben lassen. Im Folgenden werden die beiden Wichtigsten vorgestellt.

- **State Hook**

Der wohl in der Praxis am meisten genutzte Hook ist der *State Hook*. Wie der Name schon vermuten lässt, bietet dieser die Möglichkeit, die Kernfunktionalität von React, den State, in *Functional Components* zu benutzen. Dies war zuvor nur *Class Components* vorbehalten. [97]

```
2
3  const Counter = () =>{
4      //Declare new State "count", initial State 0
5      const [count, setCount] = useState(0);
6      return (
7          <div>
8              <p>Button wurde {count} mal geklickt</p>
9              <button onClick={() => setCount(count + 1)}>Hier klicken</button>
10          </div>
11      );
12  }
```

Abbildung 42: Verändern des States in React mit State Hooks

Die Funktion “*useState*” liefert hierfür einen Array zurück. Während der erste Wert die Repräsentation des aktuellen States darstellt, ist der zweite Wert eine Funktion, über die der State geändert werden kann. Als Argument wird “*useState*” der initiale Zustand, übergeben, der von jeglichem Datentyp sein kann, in diesem Beispiel eine Zahl mit dem Wert 0. In diesem Fall wird dann bei einem Button-Klick der State über “*setCount*” geändert und die Komponente neu gerendert.

- **Effekt Hook**

Ein weiterer Hook ist der sogenannte *Effect Hook*, mit der sich einer Komponente Seiteneffekte hinzufügen lassen. Diese werden zu einem bestimmten, definierten Zeitpunkt nachdem die UI gerendert wurde, ausgeführt. In der Praxis wird dies genutzt, um beispielsweise Daten aus einer Datenbank abzurufen oder Änderungen am DOM durchzuführen. [97]

```

6
7     useEffect(()=>{
8         //Show alert with the actual count
9         alert("Du hast "+count+" geklickt");
10    },[count]); //Run effect hook every time the state changes
11

```

Abbildung 43: Ausführen von Nebeneffekten mit der Effect Hook

In dieser Anwendung wird über eine Callbackfunktion eine Alert mit der aktuellen Anzahl an Klicks aus dem vorherigen Codebeispiel angezeigt. Auslöser ist eine Veränderung des States "count". Wird hier stattdessen kein Argument übergeben, wird der Effect Hook jedes mal bei einer Änderung der Komponente oder dem Neuladen der Seite ausgeführt.

3.3.1.2 Erweiterungen

Neben der Basisbibliothek von React wurden für die Umsetzung des Praxisprojekts weitere zusätzliche Module benötigt. Diese wurden zur Nutzung bei der Implementierung des Frontends über den NPM hinzugefügt.

Axios

Bei dem Modul Axios handelt es sich um einen HTTP-Client, mit dessen Hilfe beispielsweise *GET*- oder *POST*-Requests gesendet werden können und so später bei der Umsetzung des Projekts, Daten aus dem Frontend an die Datenbank des Backends weitergegeben oder deren Inhalte als Response im Frontend ausgewertet werden. In der unteren Anwendung wird ein *GET*-Request an eine beispielhafte URL eines Servers geschickt, der auf dieser Route alle Nutzer einer Datenbank als Response zurück sendet, woraufhin die Daten z.B. als JSON-Objekte verarbeitet werden. Axios ist dabei Promise-basiert, was die Möglichkeit bietet die Vorteile von *async* und *await* zu nutzen um einen lesbaren, asynchronen JavaScript Code zu erhalten. [98]

```

4
5     axios.get(`https://placeholder.com/readusers`) // Send Get-Request to a Route in Backend
6         .then((res) => { //Handle Success
7             const users = res.data; //Process the data from the response
8             console.log(users);
9         });
10    }

```

Abbildung 44: Absenden eines GET-Requests mit dem Axios-Modul

React-Hook-Form

Bei diesem Modul handelt es sich um eine praktische Erweiterung bei der Nutzung von Formularen in React. So werden dem Entwickler vor allem bei der Validierung der Input-Felder viele zusätzliche Zeilen an Code erspart, in dem über die API des Moduls die Daten der Eingaben überprüft und mit Meldungen auf mögliche Errors reagiert werden kann. Außerdem wird die Anzahl der Re-Renderings minimiert, wodurch die Performance der App verbessert werden kann. [99]

3.3.2 Backend

Für die Umsetzung des Backends der Webapp wird die in 2.4.3 behandelte Laufzeitumgebung Node.js genutzt, wodurch die Applikation als Fullstack-JavaScript-Anwendung auf einer einzigen Basis-Sprache aufbaut. Zur Vereinfachung der serverseitigen Entwicklung wird Node.js um das Framework Express, sowie einige externe, zusätzliche Module erweitert, die insbesondere für die Kommunikation mit dem Frontend und der Datenbank von Bedeutung sind.

3.3.2.1 Node und Express

Bei dem, bereits im 2. Kapitel kurz vorgestellten, Framework Express handelt es sich um eines der beliebtesten Zusatzmodule für die Node.js-Plattform. Bei dem Open-Source-Projekt liegt der Fokus bis heute auf Geschwindigkeit und einem überschaubaren Umfang, was Express nahezu unverzichtbar macht, wenn es um die Entwicklung von Webserveranwendungen mit Node geht. Im Folgenden soll daher kurz die grundlegende Funktionalität von Express vorgestellt werden und wie diese dabei hilft, gewisse Standardaufgaben unserer App mit einfacherer Syntax zu implementieren.

Aufbau

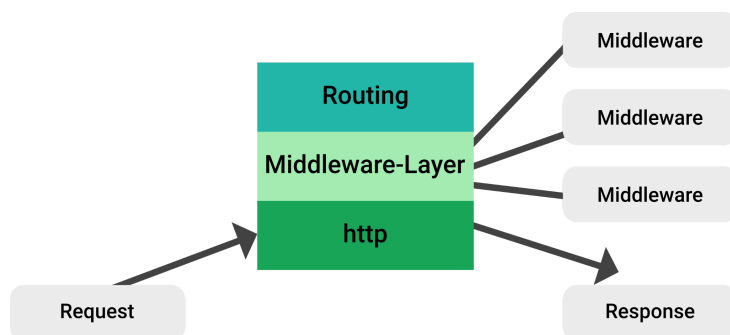


Abbildung 45: Architektur des Node-Frameworks Express

Wie sich in Abbildung 45 sehen lässt, bildet das **HTTP-Basis-Modul** von Node die Grundlage des Frameworks. Über dies lässt sich der Serverprozess erstellen und über die Request- und Response-Objekte auf Anfrageinformationen zugreifen und die Antworten an den Client verwalten. Die zweite Ebene bildet der **Middleware-Layer**, der zwischen der eingehenden Anfrage und der Antwort an den Client steht. Grundsätzlich werden Middleware-Komponenten eingesetzt um basierend auf der Anfrage des Clients definierte Funktionen auszuführen. Die dritte Schicht bildet der **Router**, über den gesteuert werden kann, welche Aktion abhängig von der aufgerufenen URL ausgelöst werden soll. [61]

Einrichtung

Nach der Installation von Express über den Node Package Manager kann das Modul, wie bereits in dem Abschnitt zu den Grundlagen von Node erläutert, über die *“require”*-Funktion importiert werden. Der Rückgabewert ist eine Funktion, die die Basis für die Konfiguration der Serveranwendung bereitstellt.

```
1
2  const express = require('express'); //Import Express
3  const app = express();
4
5  app.listen(3001, ()=>{ //Listen on Port + Callback
6    |   console.log("Server is listening on Port 3001")
7    | });
```

Abbildung 46: Grundlegendes Setup eines Server mit Hilfe von Express

Der Aufbau einer Express-App ist in der Regel immer der Gleiche. Bei einer eingehenden Anfrage wird anhand der gewählten HTTP-Methode und der passenden Route eine Callback-Funktion ausgeführt, in der man sowohl auf das Response- als auch auf das Request-Objekt zugreifen kann. [61]

- **Request**

Das Request-Objekt ist das erste Argument der Callback-Funktionen, steht für Anfragen des Benutzers und wird meistens mit *“req”* abgekürzt. Über das Objekt lassen sich so eingehende Informationen des Nutzers verarbeiten und beispielsweise eingehende Nutzerdaten oder Formulareingaben erfassen und in eine Datenbank schreiben. Von Bedeutung sind hierbei vor allem die Eigenschaften *“query”* zum Lesen variabler URL-Anteile, *“body”* für den Zugriff auf

String oder JSON-Objekte in POST-Requests und “*params*” für die Verarbeitung von URL-Query-Strings. [100]

- **Response**

Das Response-Objekt ist das zweite Argument der Callback-Funktion und steht für die Antwort an den Client. Während auf das Request-Objekt hauptsächlich lesend zugegriffen wurde, sind die Interaktionen mit dem Response-Objekt überwiegend schreibend. Für unsere Anwendung ist dies insbesondere von Bedeutung, um Inhalte der Datenbank zur Darstellung im Frontend weiterzugeben. [61]

- **Routing**

Bei eingehenden Anfragen des Users wird diese mit Hilfe des Routers und anhand des definierten URL-Pfads an eine bestimmte Funktion weitergeleitet, die für die Bearbeitung zuständig ist. Der Express-Router bietet außerdem die Möglichkeit neben der URL, auch anhand der gewählten HTTP-Methode zu entscheiden, welche Funktion ausgeführt wird. In der Praxis spielen dabei meist *GET*, *POST*, *PUT* und *DELETE* eine Rolle. [61]

```
4
5  app.post("/addCoupon", (req, res)=>{
6    //Appliation Logic ...
7    res.send("Added new Coupon");
8  });
9
10 app.delete("/deleteCoupon", (req, res)=>{
11   //Appliation Logic ...
12   res.send("Deleted Coupon");
13 });
```

Abbildung 47: Beispielhafte Routen um auf verschiedene URLs und HTTP Methoden zu reagieren

3.3.2.2 MongoDB

Die weitere wichtige Komponente im Backend des MERN-Stacks ist neben Node.js die Datenbank MongoDB. Hierbei handelt es sich um eine **dokumentenorientierte No-SQL Datenbasis**, deren grundlegendes Konzept bereits in 2.4.5.2 kurz vorgestellt wurde. MongoDB zeichnet sich dabei durch ein flexibles Schema aus. Das bedeutet, dass Felder in der Collection von Dokument zu Dokument variieren können und Datenmodelle im Laufe der Zeit auf sich ändernde Anwendungsanforderungen weiterentwickelt und angepasst werden können.

Die Daten werden hierbei nativ in JSON-Variante “*BSON*” gespeichert, wobei B für Binary steht. Für die Abfrage an sich kann man allerdings auf die JSON-Syntax zurückgreifen, was das Modell der Datenbank ideal für die Fullstack-Entwicklung mit JavaScript macht. So werden die Daten beispielsweise für das Frontend über JSON-Objekte ausgetauscht und über einfache Operationen Dokumente erstellt, geändert, durchsucht oder gelöscht. [101]

Für das Fullstack-Praxisprojekt “*Skimp*” wurde ein Cluster von **MongoDB Atlas**, einer globalen und verteilten Cloud-Datenbank genutzt, die in Microsoft Azure, Amazon AWS und Googles Cloud Platform zur Verfügung steht. Ein Cluster mit, für Skimp ausreichenden Basisfunktionen, kann dabei kostenlos erstellt werden und für die Anwendung mit dem Backend verknüpft werden. Zur Unterstützung lässt sich das Programm MongoDB Compass installieren, das als grafisches User Interface einen aktuellen Überblick über die Inhalte der Datenbasis gibt. [102]

3.3.2.2 Zusätzliche Module

Zu dem Grundgerüst aus Node.js im Zusammenspiel mit Express wurden für die Implementierung des Backends ebenfalls einige Erweiterungen als NPM Module installiert.

CORS

CORS steht für Cross-Origin Resource Sharing. Dabei handelt es sich um ein Konzept, das dem Frontend erlaubt, auf Daten des Backends zuzugreifen und in anderer Richtung zu senden. Da Client und Server unter verschiedenen Domains laufen, wird die Berechtigung normalerweise aus sicherheitsgründen blockiert. Technisch gesehen handelt es sich bei dem Modul um eine Express-Middleware, das der App nach dem Einbinden als Funktion zur Verfügung steht. [103]

Mongoose

Mongoose ist ein sogenannter Document Object Wrapper, über den sich Mongoose-Objekte nach einem typisierten Schema definieren lassen und auf einem Dokument in der Mongo-Datenbank abbilden lassen. Mongoose bietet dafür eine Auswahl an verschiedenen “*SchemaTypes*” mit denen eine Eigenschaft gespeichert werden kann und weitere Optionen um beispielsweise einen Default-Wert zu bestimmen oder anzugeben, dass ein Feld erforderlich ist. Zudem lässt sich über das Modul mit wenigen Codezeilen eine Verbindung zum Datenbank-Cluster herstellen. [104]

```

1
2  const mongoose = require("mongoose"); //Import mongoose
3  //Object as the representation of the data schmemma
4  const UserSchema = new mongoose.Schema({
5      userName: {
6          type: String,
7          required: true,
8      },
9      userAge: {
10         type: Number,
11         required: false,
12     },
13 });

```

Abbildung 48: Definieren eines Objekt-Schemas mit Mongoose

Express.json

Express.json ist eine eingebaute Middleware-Funktion, bei der die Daten eines eingehenden POST-Requests als JSON-Objekte interpretiert werden und so besser als Dokumente in der Datenbank verarbeitet werden können. [105]

Nodemon

Dieses Modul hat zwar keinen direkten Einfluss auf die Funktionalität des Backends, bietet aber die Möglichkeit in der *“Package.json”*-Datei ein Skript zu definieren, durch das die Änderungen im Code von der Server-Applikation automatisch erkannt und im laufenden Programm übernommen werden, ohne dass dies jedes mal über die Kommandozeile neu gestartet werden muss. [106]

3.4 Konzeption

3.4.1 Clientseite

Das Frontend der Anwendung soll für eine bessere Übersichtlichkeit bei der Implementierung in einzelne Bestandteile zerlegt werden. Grundlegende Komponenten der Applikation sind dabei eine Coupon-Liste, die einzelnen Coupons selbst, ein Formular und ein Button zum Öffnen des Formulars. Zudem werden auch der Header und der Footer der Seite als Components in React implementiert um das Programm strukturierter zu halten, auch wenn diese ebenfalls als statischer Code in einer HTML-Datei geschrieben hätten werden können.

Beim Aufbau des Frontends wird dabei die bereits erläuterte komponentenbasierte Struktur von der Library React deutlich. So wird hier die Möglichkeit genutzt, einzelnen Komponenten ineinander schachteln zu können, wie beispielsweise die *CouponList*, die mehrere, einzelne Coupon-Komponenten enthält.

Gebündelt werden die verschiedenen Einzelteile dann in der App-Komponente, die schließlich über den React-DOM in ein definiertes Div-Element in den realen HTML-Baum gerendert und somit dem Nutzer als User Interface dargestellt wird und dieser clientseitig damit interagieren kann.

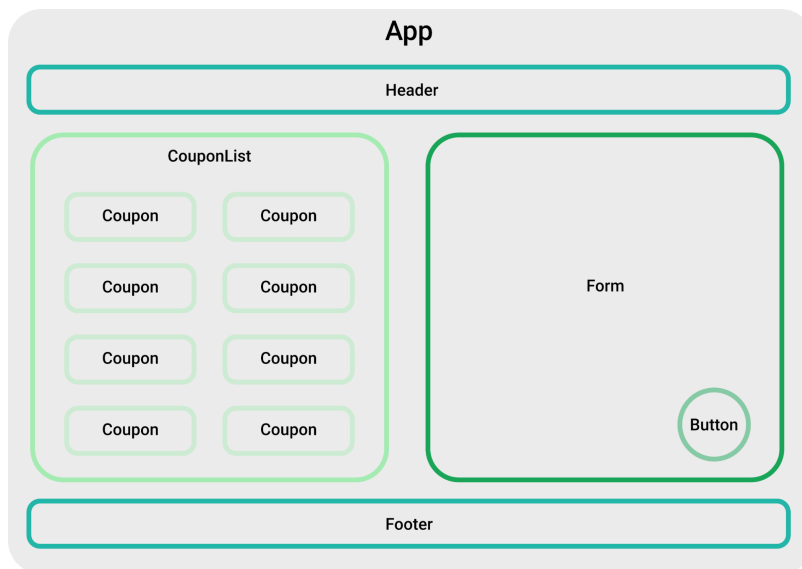


Abbildung 49: Einzelne React-Komponenten des Applikations-Frontends

3.4.2 Serverseite und Datenbank

Da der Codeumfang des Backends der Applikation deutlich geringer ist, als der des Frontends, kann hierfür eine einzige JavaScript-Datei genutzt werden, ohne dass der Code zu unstrukturiert und unübersichtlich wird.

Um auf die verschiedenen Anfragen reagieren zu können, müssen verschiedene Routen definiert werden, die die unterschiedlichen Requests verarbeiten und je nach Typ eine entsprechende Aktion ausführen um beispielsweise Nutzerdaten zu verarbeiten oder Dokumente aus der Datenbank abzurufen.

Für die Webanwendung Skimp werden dabei konkret vier Routen benötigt:

- Um Eingabedaten des Formulars aus einem **POST-Request** zu empfangen und als Mongoose-Schema da die Datenbank weiterzugeben
- Um bei einem **GET-Request** die einzelnen Coupons aus der Mongo-Database abzurufen und an das Frontend auszuliefern
- Um bei einer **HTTP-DELETE-Anfrage** nur einmal verwendbare Coupons aus der Datenbank zu entfernen
- Um bei einem **PUT-Request** bei einem einzelnen, bestimmten Dokument den Zähler der Anzahl der Einlösungen zu erhöhen.

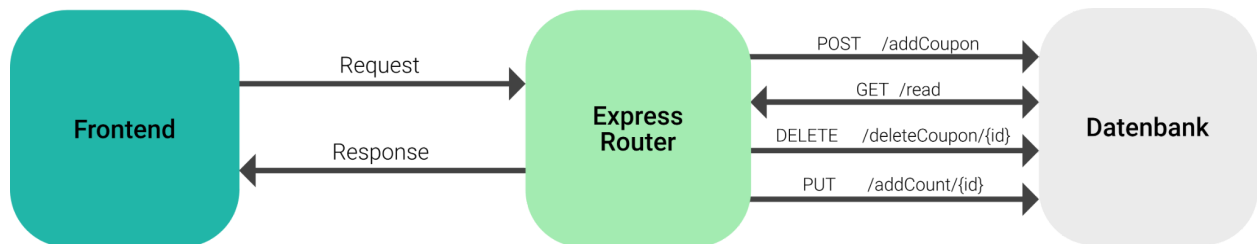


Abbildung 50: Benötigte Routen für die Funktionalität des Backends

Während sich Funktionalität der Serverseite in einer einzelnen Datei befindet, wird das mit Hilfe von Mongoose definierte Schema für die Objekte in einem separaten Modul ausgelagert, das dann mit Hilfe der "require"-Funktion importiert wird und zur Anwendung zur Verfügung steht. Das Datenmodell enthält dabei verschiedene Datentypen, die dann in dem späteren Dokument als Field-Value-Einträge in der Datenbasis zur eingetragen sind.

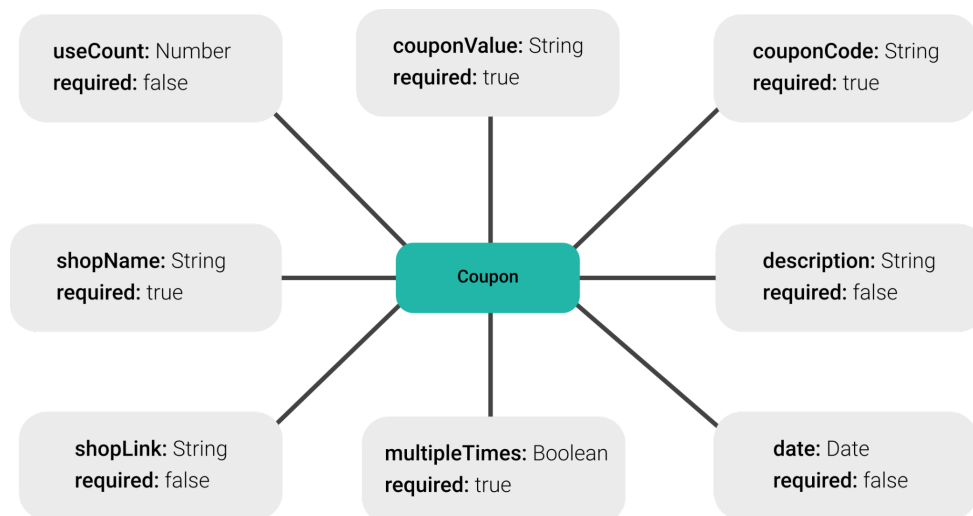


Abbildung 51: Datenmodell eines Coupon-Dokuments

3.4.3 User Interface Design

Die grafische Gestaltung des User Interfaces soll grundlegend relativ schlicht und minimalistisch sein, durch leichte Farbverläufe, abgerundete Formen der Elemente und *“Blur”*-Effekte der Hintergründe jedoch dem Nutzer einen hochwertigen, modernen Eindruck vermitteln und aktuelle Web- und App-Designtrends aufgreifen.

Skimp hat zwei primäre Akzentfarben in einem hellen und dunklen Türkis, die im Logo oder auch bei Buttons eingesetzt werden. In den Teilen der restlichen Anwendung werden diese mit schlichten, neutralen Farben wie Weiß und Grau kombiniert. Die Schrift wird in Schwarz dargestellt, um einen optimalen Kontrast zum Hintergrund zu geben. Befindet sich der Nutzer in der *“Darkmode”*-Ansicht, wird der weiße Hintergrund durch ein sehr dunkles Grün ersetzt und die Texte zur optimalen Lesbarkeit nun in Weiß statt Schwarz gestaltet.

Als Schrift wird die serifenlose Font *“Open Sans”* eingesetzt, die das schlichte, moderne User Interface optimal ergänzt, über Google Fonts lizenzfrei zur Verfügung steht und ohne Probleme in das Projekt importiert werden kann.



Abbildung 52: Logo, Farbpalette und Font-Family der Webanwendung

3.5 Implementierung

Zu Beginn der Implementierung wurden die Ordnerstrukturen für das Projekt angelegt. Die Gesamtanwendung wird in *“server”* und *“client”* unterteilt. Mit Hilfe des Terminals der Entwicklungsumgebung Visual Studio Code wurde zunächst auf dem Client mittels *“create-react-app”* die Basisanwendung erstellt und anschließend die in 3.3 vorgestellten Erweiterungen über den Node Package Manager zum Projekt hinzugefügt. Da Frontend und Backend auf verschiedenen Ports laufen, müssen auch beide separat über die Kommandozeile durch *“npm start”* bzw. *“node index.js”* gestartet werden.

3.5.1 Coupons hinzufügen

Um später Coupons zur Datenbank hinzufügen zu können wurden zunächst in der übergeordneten App-Komponente die einzelnen States als *“State Hook”* erstellt, damit diese global verfügbar sind. Um in der ausgelagerten Formular-Komponente mit Funktionen darauf zugegriffen zu können, werden die einzelnen States, sowie die Methoden, um diesen zu ändern als Properties der Form-Component übergeben. Man spricht hierbei auch davon, den State anzuheben, sodass dieser später von mehreren Komponenten genutzt werden kann.

Das Formular selber besteht grundlegend aus einem `<form>`-Tag, sowie den einzelnen Input-Feldern, die die Benutzereingaben für die Daten des Coupons wie der Shop, der Code, die Kategorie oder die Möglichkeit diesen mehrfach zu verwenden, entgegen nehmen.

Im Vergleich zu herkömmlichen HTML-Formularen, bei denen die Input-Felder einen natürlichen, internen Wert besitzen, der durch die Nutzereingabe bestimmt wird, ist es in React von Vorteil sogenannte kontrollierte Input-Felder zu implementieren. Hier aktualisiert ein Update am React-State den Wert des Formular-Elements und ebenso aktualisiert ein Update am jeweiligen Formular-Element den React-State. Diese **Controlled Components** sind etwas aufwändiger in der Umsetzung, sind zugleich jedoch auch die sicherere Variante, da keine Gefahr besteht, dass beide States bzw. Werte voneinander abweichen. [106] Hierfür wird mit einem Event-Listener gearbeitet, der immer auf Änderungen direkt State entsprechend dem Inhalt des Input-Feldes aktualisiert.

```
19
20     <input type="text" placeholder="Shop" value={shopName}
21         onChange={(event)=>{setShopName(event.target.value)}} />
22
23     <input type="text" placeholder="Betrag (€/%)" value={couponValue}
24         onChange={(event)=>{setCouponValue(event.target.value)}} />
25
```

Abbildung 53: Input-Felder als *“Controlled-Components”*

Ein weiterer Event-Listener überwacht außerdem, ob der Nutzer auf den *“Sende”*-Button klickt. Ist dies der Fall, wird in einer, vom Modul *“React-Hook-Form”* zur Verfügung gestellten, Funktion überprüft, ob die Pflichtfelder für den Shopnamen, den Code und den Wert des Gutscheins vom User ausgefüllt wurden. Falls keine Fehlermeldungen vorliegen, wird die Funktion *“addToList”* ausgeführt. Diese sendet die Inhalte der States als JSON-Objekt über das Axios-Modul per POST-Request an die Route *“/add”* im Backend. Damit der neue Coupon, ohne die Seite neu zu laden direkt angezeigt wird, wird außerdem asynchron definiert, dass nach erfolgreichem Senden die Eingabedaten in der *“CouponList”* als neuer Coupon gerendert werden sollen.

```

11 //Add Coupon to the Database in the backend
12 const addToList = ()=>{
13     Axios.post("http://localhost:3001/add", //Send post request with JSON Object to the route in backend
14     {shopName: shopName, couponCode: couponCode, description: description, multipleTimes: multipleTimes,
15     couponValue: couponValue, couponLink: couponLink, date:date, useCount: 0}).then((response)=>{
16     setCouponList([...couponList, {_id: response.data._id, shopName: shopName, couponCode: couponCode,
17     description: description, multipleTimes: multipleTimes, couponValue: couponValue,
18     couponLink: couponLink, date: new Date(null)}}]);
19     });
20 };

```

Abbildung 54: POST-Request mit Eingabedaten senden und direktes Aktualisieren

Im Backend werden die JSON-Daten über das `"request.body"`-Objekt verarbeitet und als neues Mongoose-Objekt an die Datenbank weitergegeben.

```

26 //Represents the value from form input fields
27 const shopName = req.body.shopName;
28 const couponValue = req.body.couponValue;
29 const couponCode = req.body.couponCode;

```

Abbildung 55: Verarbeiten der Daten des Request-Bodys



```

_id: ObjectId("60ec2a1bf4c09e3fbceb9a2e")
shopName: "Zalando"
couponValue: "10%"
couponCode: "FINAL-10"
description: "Mode"
multipleTimes: true
couponLink: "https://www.zalando.de/"
date: 2021-08-15T00:00:00.000+00:00
useCount: 0
__v: 0

```

Abbildung 56: Dokument im Datenbank-Cluster

3.5.2 Coupons abrufen

Um die einzelnen Coupons aus der Datenbank bei Start der Anwendung direkt zu anzeigen wird eine `"useEffect"`-Hook geschrieben, die immer beim Laden der Seite einen GET-Request an die Route `"/read"` sendet.

```

23 //Effect Hook which is executed at page load
24 useEffect(()=>{
25     Axios.get("http://localhost:3001/read").then((response)=>{ //Send request to route in backend
26     setCouponList(response.data); //Handle data from the response
27     });
28 },[]); //Start useEffect Hook as soon as page loads

```

Abbildung 57: GET-Request und Verarbeiten der Antwort durch aktualisieren des States

Im Backend werden mit Hilfe der *“find”*-Funktion bestimmte Dokumente aus der Datenbank geladen. Da hier kein spezieller Parameter übergeben wird, werden alle Inhalte abgerufen und es wird nicht gefiltert.

```
48 //Read all Coupons from the Database
49 app.get("/read", async (req, res)=>{
50   CouponModel.find({}, (err, result)=>{ //No paramter -> All Coupons from DB
51     if(err){
52       res.send(err);
53     }
54     res.send(result); //Send the Documents to the Frontend
55   });
56 });
```

Abbildung 58: Coupons der Datenbank abrufen

Die Antwort vom Backend wird clientseitig im State *“couponList”* als Array mit den verschiedenen Coupon-Objekten festgehalten. Die JavaScript-Funktion *“map”* ruft dabei jedes einzelne Element des Arrays nacheinander auf und gibt die jeweiligen Objekte als Property *“val”* an die Coupon-Komponente weiter. So kann schließlich über die typische Punkt-Notation auf die einzelnen Werte des Objekts zugegriffen werden und diese im Frontend für die unterschiedlichen Coupons dargestellt werden.

```
32 {couponList.map((val, key)=>{
33   return(
34     <Coupon val={val} key={key} couponList={couponList} setCouponList={setCouponList}/>
35   );
36 });
37
```

Abbildung 59: Erstellen der *“CouponList”*-Komponente aus dem Objekt-Array

3.5.3 Coupons einlösen

Die Coupons können vom Nutzer über einen Klick auf den entsprechend beschrifteten Button eingelöst werden. Dieser wird anschließend über das Hinzufügen einer bestimmten CSS-Klasse aus- und der eigentliche Code auf die gleiche Weise eingeblendet.

Zeitgleich wird nach dem Auslösen des Events eine weitere Funktion *“cashCoupon”* ausgeführt. Diese überprüft in einem *“if”*-Statement, ob es möglich ist, den jeweiligen Coupon mehrfach einzulösen. Ist dies nicht der Fall, wird über Axios ein Delete-Request an eine weitere, im Backend definierte Route *“/delete”* gesendet. Besonders hierbei ist, dass zusätzlich als Variable die eindeutige ID des Coupons per URL-Parameter mitgegeben wird. Die ID bekommen die Coupon-Dokumente beim Eintrag in die Datenbank automatisch zugewiesen.

```

42 //Use Coupon and delete it in the DB if you can't use it more than once
43 const cashCoupon = (id) =>{
44     if(val.multipleTimes === false){
45         Axios.delete(`http://localhost:3001/delete/${id}`);
46     } else{
47         Axios.put(`http://localhost:3001/countAdd/${id}`);
48     }
49 };

```

Abbildung 60: Übergeben der ID als URL-Parameter

Auf der Serverseite wird die Anfrage vom Express-Router entgegen genommen. Mit Hilfe des Request-Objekts kann die ID verarbeitet werden. Im Gegensatz zum Hinzufügen eines Coupons stehen die entscheidenden Informationen dieses mal nicht im Body des Requests, sondern in der URL. Daher wird hier das Objekt *“params”* genutzt, um die ID zu extrahieren. In einer asynchronen Callback-Funktion wird schließlich über die praktische, von Mongoose zur Verfügung gestellten Methode, direkt in einem Aufruf nach dem Identifier gefiltert, sowie das Dokument anschließend aus der Datenbank gelöscht.

```

58 //Delete a Coupon from the Database, which can only be used one time
59 app.delete("/delete/:id", async (req,res)=>{
60     const id = req.params.id; //Parameter from the URL Argument
61     await CouponModel.findByIdAndRemove(id).exec();
62     res.send("Coupon deleted");
63 });

```

Abbildung 61: Einfach einlösbare Coupons löschen

Das *“else”*-Statement ist für die Gutscheine, die mehrfach eingelöst werden können. Der Ablauf dieser Verzweigung ist ähnlich wie die des Löschens aus der Datenbank. Hier wird jedoch auf der Serverseite der Coupon mit der jeweiligen ID nicht gelöscht, sondern dessen *“useCount”*, d.h. die Anzahl der Einlösungen um eins erhöht.

3.5.4 Suchfunktion

Die Suchfunktion wurde vorerst rein clientseitig umgesetzt und dafür in die *“CouponList”*-Komponente ausgelagert, in der auch die Suche als einfaches Input-Feld enthalten ist.

Wie bereits die Felder des Formulars, wurde auch hier die Suche als *“Controlled Component”* implementiert, sodass bei Nutzereingaben immer direkt der entsprechende State aktualisiert wird. Zur Darstellung auf dem User Interface macht man sich das React-Konzept des **Conditional Renderings** zu Nutze. Dabei werden in Abhängigkeit des States entweder alle Gutscheine oder nur bestimmte Coupon-Komponenten zur Darstellung gerendert.

Bei leerem Input-Feld erscheinen im Frontend alle Coupons des Arrays. Bei einer Texteingabe des Benutzers wird dagegen mit der in JavaScript verfügbaren *“includes”*-Funktion überprüft ob Teile des Strings im Eingabefeld auf die Shopnamen der Coupon-Objekte im Array zutreffen und anschließend nur diese zurückgegeben. Damit es zu keinen Komplikationen bei Groß- und Kleinschreibung kommt, wird zum Abgleichen sowohl der String des Shop-Namens, als auch der Inhalt des Suchfelds mittels der Methode *“toLowerCase”* in Kleinbuchstaben umgewandelt.

```
19
20     {couponList.filter((val)=>{
21         if(searchTerm==""){
22             return val; //Return all Coupons with empty String in Search Field
23         } else if(val.shopName.toLowerCase().includes(searchTerm.toLowerCase())){
24             return val; //Return all Coupons with specific String from the Search Input
25         }
26     })}
```

Abbildung 62: Bedingte Darstellung in Abhängigkeit des States *“searchTerm”*

3.6 Deployment

Beim Deployment (*deploy = bereitstellen*) geht es um das Bereitstellen der Anwendung auf einem öffentlich verfügbaren Internet-Server.

Die Anwendung läuft nun auf 3 Servern:

1. Die Datenbank auf dem **MongoDB Atlas** Server
2. Der Node-Server bei **Heroku.com**.
3. Die React-Clientseite bei **Netlify.com**.

Server

Die MongoDB-Atlas Datenbank läuft bereits auf einem externen Server. Hier muss man für den öffentlichen Betrieb der Anwendung einstellen, dass Anfragen von allen IP-Adressen akzeptiert werden. Vor dem Deployment des Node-Servers sind noch ein paar kleine Änderungen daran vorzunehmen. Im Server-Directory muss *“Dotenv”* installiert werden und in der *„index.js“*-Datei (bzw. in der Ausgangsdatei des Skripts) eingebunden werden. Durch Dotenv kann auf eine *„env“*-Datei mit Umgebungsvariablen zugegriffen werden, welche der Server später erstellt. Aus dieser Datei erhält der Node-Server die ihm zugewiesene Portnummer und kann auf diesen Port hören.

```

71
72 app.listen(process.env.PORT || 3001, ()=>{
73   console.log("You're connected");
74 });
75

```

Abbildung 63: Zugriff auf die Portnummer aus den Server-Umgebungsvariablen.

Vor dem Deployment ist auch zu prüfen, ob der Programm-Startpunkt in der “*Package.json*”-Datei definiert ist:

```

  ▸ Debug
2   "scripts": {
3     "start": "node index.js",
4     "devStart": "nodemon index.js"
5   },

```

Abbildung 64: Definition des Server-Startpunkts in der Datei *package.json*.

Die Serverseite unserer Anwendung wird bei **Heroku.com** bereitgestellt. Heroku ist eine Cloud-Plattform als Service, bei der das Bereitstellen von Cloud-Anwendungen bis zu 100GB kostenlos ist. Hier muss man sich zuerst einen Account erstellen und kann dann eine neue App anlegen. Danach erhält man eine Anleitung, wie man durch Terminal-Befehle die Anwendung bereitstellt. Zunächst wird dafür ein Git-Repository initialisiert, um dann den gesamten Serverteil der Anwendung dorthin zu committen und dann auf Heroku bereitzustellen. Der gesamte Prozess besteht aus 6 Terminal-Befehlen und geht dementsprechend sehr schnell und mit wenig Aufwand. Wenn Heroku mit dem Deployment fertig ist, weist es der App eine entsprechende URL zu, über welche unsere Anwendung später erreichbar sein wird. Danach kann man auf Heroku.com die App starten und testen. Hierfür kann man beispielsweise eine Get-Route ausprobieren: in unserem Fall die zugewiesene URL + „/read“. Wenn alles funktioniert hat, werden nun die entsprechenden Datenbank-Inhalte in JSON-Form im Browserfenster angezeigt. Updates können auch nach dem Veröffentlichen schnell und mit wenig Aufwand durchgeführt werden. Hierfür muss man sich wieder bei Heroku einloggen und einfach wie von Git gewohnt die Änderungen committen.

Client

Auf der Client-Seite müssen vor dem Deployment auch kleine Veränderungen vorgenommen werden. Bislang waren alle Request-URLs noch Localhost-Adressen. Um mit der auf Heroku veröffentlichten Anwendung kommunizieren zu können, müssen die Adressen nun durch die von Heroku zugewiesene URL ersetzt werden.

```
24
25     useEffect(()=>{
26         Axios.get("https://coupon-test-app.herokuapp.com/read").then((response)=>{
27             setCouponList(response.data); //Handle data from the response
28         });
29     },[]); //Start useEffect Hook as soon as page loads
```

Abbildung 65: Einfügen der Server-URL anstelle des Localhost

Dann kann man eine Build-Version der Anwendung erstellen, welche für das Ausführen auf dem Server optimiert ist. Dieser Schritt ist optional. Danach erstellt man ein GitHub-Repository, wohin man den gesamten Client-Teil der Anwendung committed.

Bereitgestellt wird die Anwendung bei **Netlify**. Hier muss man ebenfalls erst einen Account erstellen. Dann wählt man die Option „New Site from Git“ aus und verbindet Netlify mit dem zuvor erstellten GitHub Repository. Dann muss man den Build-Command durch „*CI= npm build*“ und anschließend den „Deploy Site“-Button drücken. Wenn Netlify mit dem Deployment der Seite fertig ist, erhält man einen Link über den die Seite jetzt erreichbar ist. Zudem kann man hier auch direkt weitere Schritte durchführen, wie beispielsweise das Erwerben der passenden Domain oder das Anpassen von Sicherheitsoptionen. Auch hier geht das nachträgliche Updaten mit wenig Aufwand, in dem einfach die Änderungen in das GitHub-Repository committed werden.

4. ERFAHRUNGEN UND ERGEBNISSE

In diesem Kapitel werden die Erfahrungen und Ergebnisse des Praxisprojekts zusammengefasst und bewertet.

4.1 Aufgetretene Probleme

Der Umfang dessen, was komplett neu erarbeitet werden musste, war sehr groß. Im Studium wurde JavaScript zuvor selten und nur die Grundlagen behandelt. Dazu kamen noch gänzlich neue Themen wie Node.JS, React.JS, MongoDB und die gesamte Kommunikation zwischen den einzelnen Teilsystemen.

Die verschiedenen Syntax-Formen bestimmter Ausdrücke in JavaScript, die mit ES6 einzug hielten, haben den Lernprozess zudem erschwert und machten auch das Beheben von Fehlern oft nicht leichter. Es stellte ebenfalls eine Herausforderung dar, die Ordner- und Dateistruktur übersichtlich zu halten, dass noch nachvollziehbar ist, welche Wege die Daten gehen.

Für die React-Anwendung wäre es vorteilhaft gewesen, Redux für das State-Management zu nutzen, da es den Code übersichtlicher gemacht hätte, die einzelnen States getrennt und global verwaltet hätten werden können und zur Nutzung nicht als Properties jeweils den einzelnen Komponenten zur Verfügung gestellt und ausgetauscht werden hätten müssen. Eine Einarbeitung in dieses gänzlich neue Modul hätte jedoch einen größeren zeitlichen Aufwand mit sich gebracht, bis dieses sicher und voll funktionsfähig in die Anwendung integriert werden hätte können.

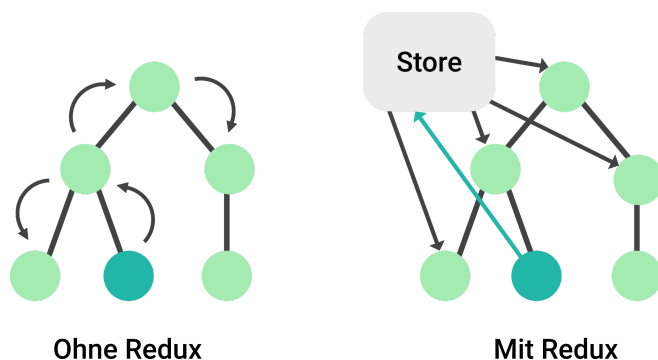


Abbildung 66: Globales State-Management mit Redux

Gewisse Schwierigkeiten bereitete auch das Speichern, Formatieren und die letztendliche Darstellung des Ablaufdatums für den User. Da dies im Input-Feld als *“Date”*-Objekt an das Backend gesendet wird, ist der Wert im Dokument in der Datenbank in der wenig anschaulichen ISO-Darstellung gespeichert (siehe Abbildung 56). Nachdem zunächst versucht wurde, das Datum bereits als String im Datenbank-Cluster zu speichern, dieser Ansatz im Hinblick auf das Überprüfen nach abgelaufenen Daten wieder verworfen wurde, konnte schließlich mit der Library *“moment.js”* eine Lösung gefunden werden, die auf der Clientseite vor dem Rendern die Objekte in Strings umwandelt und gleichzeitig die unhandliche ISO-Darstellung in ein lesbares Datum nach europäischem Format transformiert.

In Node trat oft das Problem auf, dass die Fehlercodes nicht hilfreich für die Behebung des Problems waren und es wenig Hilfe bei der Fehlerbehandlung gab. Errormeldungen in der Konsole verweisen oft auf Dateien und Module im *“node_modules”*-Ordner, welche man nicht selbst geschrieben hat, sondern extern bereitgestellt werden. Oft wird auch auf Code-Zeilen in der *„index.js“*-Datei verwiesen, welche nicht existieren: z.B. besteht die Datei aus 200 Zeilen Code, aber der Fehlercode verweist auf Zeile 500 in dieser Datei.

In der Server-Anwendung wurde für mehr Übersichtlichkeit und Struktur mehrfach versucht Routen und die Funktionen für die Interaktion mit der Datenbank in separate Dateien aufzusplitten und in der *“index.js”* wieder zusammen zu führen, da diese Struktur gerade bei größeren Anwendungen als *“State of the Art”* von vielen Entwicklern genutzt wird. Es gelang im Praxisprojekt jedoch nicht, die verschiedenen Module korrekt einzubinden und miteinander zu verknüpfen, um so einen lauffähigen Server zu starten.

Während der Entwicklung selbst traten keine weiteren, größeren Probleme oder Bugs auf, die den Stand des Projekts weit zurück warfen oder längere Zeit aufhielten. Lediglich eine vergessene Klammer im Code des Frontends führte bei dem Start der Anwendung zu einem wiederholten, unkontrollierten Aufruf der Funktion *“addToList”*. Weil zu diesem Zeitpunkt noch keine Formularvalidierung implementiert war, wurden so mehrere Tausend leere Coupon-Dokumente der Datenbank hinzugefügt. Da in dieser Phase nur einige Test-Gutscheine angelegt waren und es zeitlich nicht möglich wäre, die vielen Coupons eigenhändig zu löschen, konnte einfach das Datenbank-Cluster auf dem Mongo-DB Server gelöscht werden und ein Neues angelegt werden.

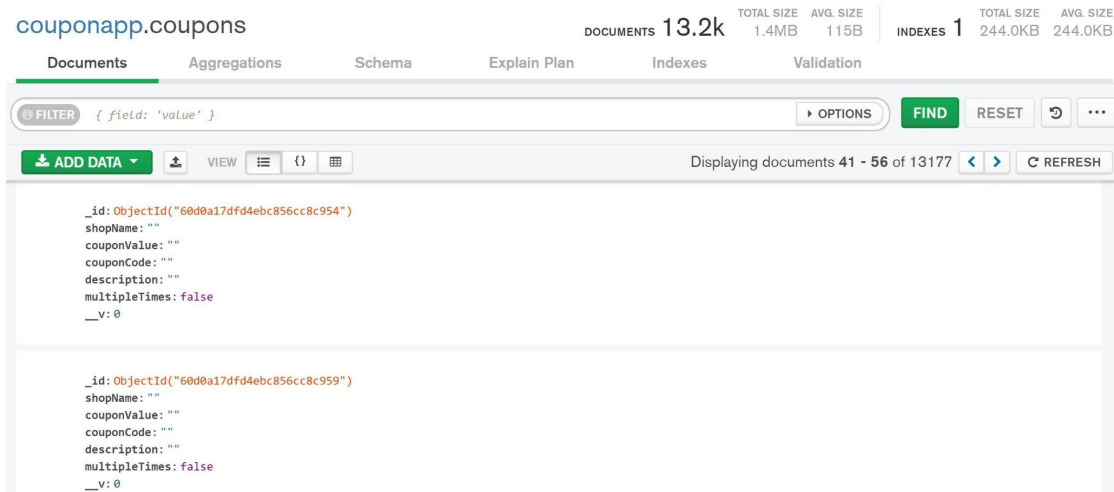


Abbildung 67: Leere Dokumente in der Datenbank

4.2 Evaluation

In diesem Abschnitt wird die Realisierung unseres Praxisprojekts bezüglich dessen Funktionalitäten evaluiert. Dabei wird überprüft, ob der erstellte Prototyp funktioniert und die festgelegten Anforderungen erfüllt.



Abbildung 68: Mockup der Anwendung auf verschiedenen Endgeräten

Erfüllung der Pflichtenforderungen

- *Als Nutzer der Webapp möchte ich beim Aufruf alle verfügbaren Coupons angezeigt bekommen, um eine Übersicht der verschiedenen Angebote zu erhalten.*

Beim Seitenaufruf werden durch einen GET-Request alle in der Datenbank enthaltenen Coupons abgerufen und dem Nutzer angezeigt. Die Anforderung wurde somit erfüllt. Dies funktioniert gut, solange noch nicht all zu viele Coupons in der Datenbank sind. Doch wenn die Datenmenge wächst und hunderte Coupons in der Datenbank sind, wäre es suboptimal, alle Daten auf einmal zu laden. In diesem Fall könnte es sinnvoll sein, dies im weiteren Verlauf so zu implementieren, dass vorerst nur eine beschränkte Anzahl Coupons geladen wird und auf weitere Anfrage dann weitere Coupons nachgeladen werden (*Progressives Laden*).

- *Als Nutzer von Skimp möchte ich eigene Codes, die ich nicht mehr benötige, hinzufügen können, um diese mit anderen Nutzern zu teilen.*

Durch das Klicken eines Button im unteren rechten Bildschirmbereich öffnet sich eine Formularseite, genauer gesagt ein PopUp-Fenster mit einem Formular. Das Formular enthält Eingabefelder für alle Daten eines Coupons. Wenn man das Formular ausgefüllt hat und auf den „Hinzufügen“-Button klickt, dann wird das Formular validiert. Wenn alle benötigten Angaben eingefügt wurden, dann wird das wird der Coupon der Datenbank hinzugefügt und im Interface angezeigt. Die Anforderung wurde somit erfüllt.

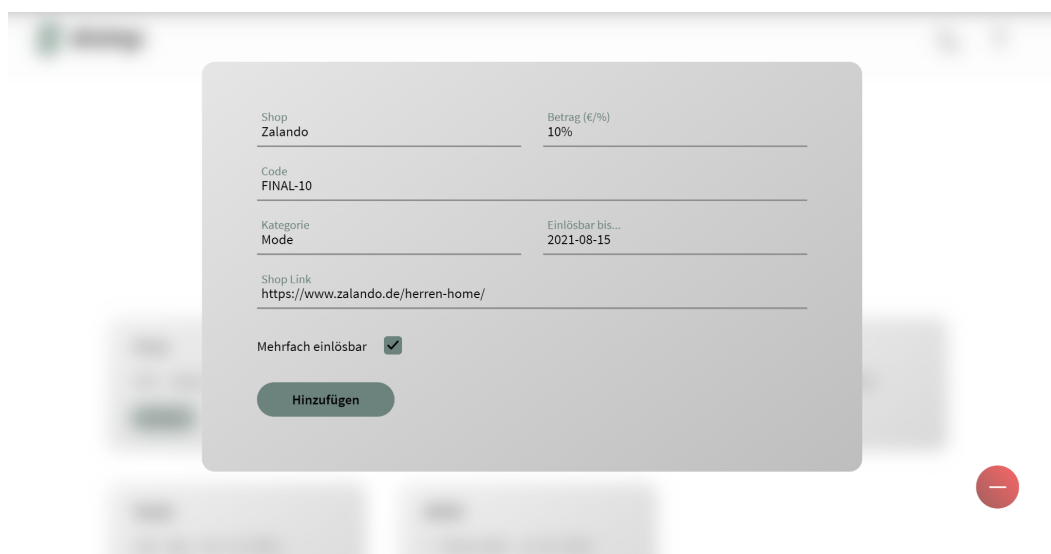
The image shows a light gray pop-up form with rounded corners. It contains several input fields with labels and values. The first row has 'Shop' labeled 'Zalando' and 'Betrag (€/%)' labeled '10%'. The second row has 'Code' labeled 'FINAL-10'. The third row has 'Kategorie' labeled 'Mode' and 'Einlösbar bis...' labeled '2021-08-15'. The fourth row has 'Shop Link' labeled 'https://www.zalando.de/herren-home/'. Below these fields is a checkbox labeled 'Mehrfach einlösbar' which is checked. At the bottom is a green button labeled 'Hinzufügen'. To the right of the form is a red circular button with a white minus sign.

Abbildung 69: Geöffnetes und vollständig ausgefülltes Nutzer-Formular

- *Als Verwalter der Anwendung möchte ich, dass Formulareingaben vor dem Absenden überprüft werden, damit die Coupons korrekt in der Datenbank gespeichert werden.*

Es müssen nicht alle Felder des Formulars ausgefüllt werden, um einen Coupon hinzuzufügen, sondern es werden nur die Daten benötigt, welche im Datenmodell als „required“, also als Pflicht-Wert, deklariert sind. Vor dem Absenden wird in der React-Anwendung überprüft, ob alle Pflichtfelder ausgefüllt sind. Wenn dies nicht der Fall ist, dann erhält der Nutzer einen Hinweis an allen fehlenden Formularfeldern. Wenn alle benötigten Felder ausgefüllt sind, dann werden die Daten übertragen und der erstellte Coupon angezeigt. Die Anforderung wurde somit erfüllt.

The screenshot shows a form for creating a coupon. It has several input fields: 'Shop', 'Betrag (€/%)', 'Code', 'Kategorie', 'Einlösbar bis...', and 'Shop Link'. Below the 'Shop' and 'Betrag' fields, there are red error messages: 'Bitte geben Sie den Shop ein' and 'Bitte geben Sie einen Betrag in € o. % ein'. Below the 'Code' field, there is a red error message: 'Bitte geben Sie einen Couponcode ein'. At the bottom, there is a checkbox labeled 'Mehrfach einlösbar' and a green button labeled 'Hinzufügen'.

Abbildung 70: Validierung der Pflichtfelder

- *Als Benutzer der Seite möchte ich alle Inhalte auf kleineren Displays ohne zu zoomen erkennen können, damit ich auch auf meinem Smartphone die Anwendung bedienen kann.*

Bei der Erstellung wurde in jedem Schritt beachtet, dass die Bedienung auf mobilen Endgeräten wie Tablets und Smartphones auch funktioniert und die Darstellung aller Elemente entsprechend angepasst wird. Dies wird in CSS durch Media Queries umgesetzt. Die Anforderung wurde somit erfüllt.

Finde dein Schnäppchen...

 Suchen...

Yoox

10% Mode Oct 10, 2022

Einlösen

Zalando

10% Mode Aug 15, 2021

Einlösen



Abbildung 71: Responsive Darstellung auf einem mobilen Endgerät

- Als Nutzer der Seite möchte ich die Gutscheine einlösen können, um bei einem Einkauf in einem Online-Shop den Code nutzen zu können.

Die Coupons werden im Interface als Karten dargestellt. Sie enthalten die wichtigsten Daten des Coupons und einen „Einlösen“-Button. Wenn man diesen Button anklickt, dann verschwindet dieser, der Coupon-Code und ein Kopier-Button erscheinen. Durch das Anklicken des Kopier-Buttons wird der Coupon-Code in die Zwischenablage des Geräts kopiert und kann dann im Shop eingelöst werden. Die Anforderung ist somit erfüllt.

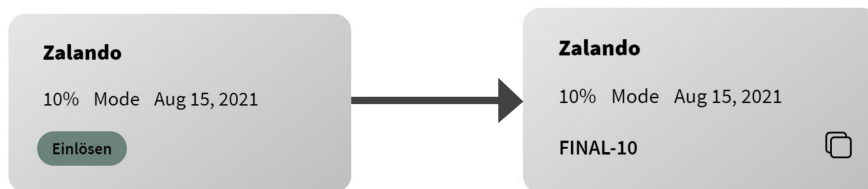


Abbildung 72: Coupon-Verhalten bei Nutzerinteraktion

- Als Nutzer der Seite erwarte ich, dass bereits eingelöste Coupons gelöscht werden, damit ich nur noch gültige Codes in der App vorfinde.

Wenn ein Nutzer den „Einlösen“-Button anklickt, dann wird geprüft, ob es sich bei dem Coupon um einen mehrfach einlösbaren Coupon handelt oder ob er nur einfach einlösbar ist. Wenn dieser nur einmal einlösbar ist, dann wird er im Hintergrund direkt aus der Datenbank gelöscht. Dem Nutzer wird der Code trotzdem noch angezeigt und er kann ihn so einlösen. Dem nächsten Besucher der Seite wird der Coupon nicht mehr angezeigt. Die Anforderung ist somit erfüllt.

- *Als Seitenbetreiber möchte ich, dass das User Interface der Anwendung ansprechend gestaltet ist, um den Besuchern ein positives Nutzererlebnis zu bieten.*

Das User Interface ist unaufdringlich, simpel und übersichtlich gestaltet. Dem Nutzer werden alle relevanten Informationen auf einen Blick geboten. Die Bedienung ist durch das One-Page-Design sehr intuitiv. Übergänge sind stets animiert und die komplette Darstellung ist vektorbasiert und bietet somit auf allen Geräten die bestmögliche Darstellung bei kleinstmöglicher Datenmenge. Die Anforderung ist somit erfüllt.

- *Als Nutzer möchte ich schnell und ohne Verzögerungen durch die Webanwendung navigieren können.*

Durch die One-Pager-Darstellung sind alle Elemente der Anwendung immer schnell erreichbar. In der Anwendung finden keine URL-Wechsel statt, so entstehen keine Verzögerungen durch das Neuladen von Informationen. Zudem ist die Datenmenge gering und der Node-Server sehr leistungsfähig, wodurch keine Verzögerungen stattfinden. Die Anforderung ist somit erfüllt.

- *Als Nutzer der Anwendung möchte ich auf der Seite nach verschiedenen Shops suchen können, um schnell einen passenden Coupon zu finden.*

Die Suche nach Shops funktioniert und wurde im Frontend umgesetzt. Die bereits im Frontend empfangenen Daten werden nach dem gesuchten Inhalt gefiltert. Im weiteren Verlauf wäre es sinnvoller, wenn man die Suchfunktion so umsetzt, dass die Anwendung die Suche in der Datenbank durchführt und die Ergebnisse dann an das Frontend sendet. Die Anforderung wurde somit dennoch erfüllt.

Finde dein Schnäppchen...

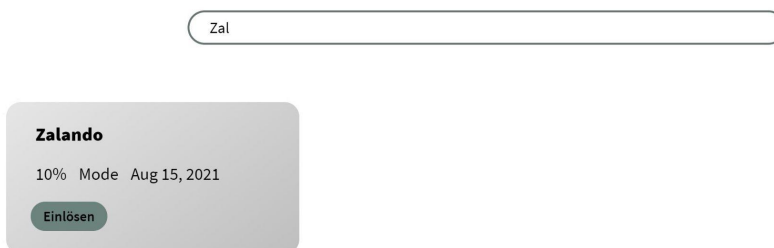


Abbildung 73: Suchfunktion der Anwendung

- *Als Verwalter der Seite erwarte ich, dass die Eingabefelder des Formulars zurückgesetzt werden, nachdem ein Coupon hinzugefügt wird, damit der Kunde die Eingaben nicht selbst löschen muss.*

Wenn man dies nicht explizit implementiert, dann werden die Daten zwar korrekt übertragen und der Coupon wird erstellt und angezeigt, aber man erhält als Nutzer nicht ausreichend Feedback. So denkt man, dass es nicht funktioniert hat, man klickt den Button mehrmals und dadurch ist der Coupon danach vielfach in der Datenbank vorhanden. Zudem musste man für das erneute Einfügen eines Coupons entweder alle Felder manuell leeren oder die Seite neu laden. Um dies zu beheben wurde diese Funktion so implementiert, dass sich das Formular nach dem Hinzufügen leert und schließt und die States zurückgesetzt werden. Somit wurde die Anforderung erfüllt.

Erfüllung der optionalen Anforderungen

- *Als Nutzer der Seite möchte ich, dass die Nutzer den Code eines eingelösten Gutscheins per Buttonklick in die Zwischenablage kopieren können, damit beim späteren Einfügen keine Fehler entstehen.*

Wie bereits beschrieben wurde auch diese Anforderung erfüllt. Der Kopier-Button erscheint auf Anfrage zusammen mit dem Coupon-Code. Wenn man den Button anklickt, dann wird der Code mittels einer JavaScript-Schnittstelle in die Zwischenablage des Geräts kopiert.

- *Als User erwarte ich eine Option für manuellen Dark Mode, damit ich die UI nach meinen Displaypräferenzen anpassen kann.*

Dies funktioniert nicht nur manuell, sondern die Anwendung erkennt automatisch die Geräteeinstellung des Nutzers und wählt dementsprechend entweder die helle oder die dunkle Darstellung (siehe Abbildung 68). Dies funktioniert über ein CSS Feature Query. Somit ist auch diese Anforderung erfüllt.

- *Als Seitenbetreiber möchte ich, dass abgelaufene Coupons automatisch aus der Datenbank gelöscht werden, damit den Nutzer keine ungültigen Gutscheincodes angezeigt werden.*

Diese Anforderung wurde zuletzt umgesetzt, funktioniert nun jedoch einwandfrei und ermöglicht, dass bei jedem neuen Seitenaufruf zuvor in der Datenbank die Ablaufdaten der einzelnen Coupons überprüft und ältere, nicht mehr gültige Coupons gelöscht werden.

Ergebnis der Evaluation

Unter Verwendung von JavaScript als einzige Programmiersprache sowohl im Bereich der clientseitigen Webentwicklung, als auch der serverseitigen Webentwicklung, konnte mit Unterstützung des Frameworks React und der Laufzeitumgebung Node die voll funktionsfähige Webanwendung Skimp umgesetzt werden.

Hilfreich waren dabei mit Sicherheit die guten, offiziellen Dokumentationen der Frameworks, sowie die zahlreichen verfügbaren Tutorial-Reihen oder Fachbücher, die die Einarbeitung in ein neues, komplexes Fachgebiet erleichterten und dabei unterstützen, weitgehend unbekannte Konzepte, wie die der asynchronen Programmierung, der Anbindung einer nicht-relationalen Datenbank oder dem Implementieren einer dynamischen UI zu erlernen. Insbesondere bei Schwierigkeiten in der Umsetzung, Bugs oder auf den ersten Blick unerklärlichen Fehlern wurde der Vorteil der weltweiten Beliebtheit von JavaScript und der großen Community um die, inzwischen weit verbreiteten, Frameworks Node und React deutlich. So konnten meist nach kurzer Recherche zahlreiche Lösungsvorschläge zu gleichen oder ähnlichen Problemen gefunden werden.

Die Applikation erfüllt somit abschließend die zuvor definierten Pflichtanforderungen und alle grundsätzlichen Funktionalitäten werden einwandfrei ausgeführt. Es konnten sogar bereits einige optionale Anforderungen im Prototyp umgesetzt werden, doch für eine finale, skalierbare Anwendung, die noch mit deutlich mehr Coupons in der Datenbank umgehen können sollte, sind einige weitere Features zu implementieren, sowie neue Möglichkeiten und Konzepte zu erlernen, um beispielsweise eine Benutzerauthentifikation, ein progressives Nachladen der Inhalte oder eine übersichtlichere Verwaltung der States zu realisieren.

5. FAZIT

5.1 Zusammenfassung

Ziel der Projektarbeit war es, den aktuellen Stand des Full Stack Developments mit JavaScript zu erfassen, sowie client- und serverseitige Frameworks für die Webentwicklung zu untersuchen und miteinander zu vergleichen. Zuletzt sollte mit diesem Wissen dann eine eigene Webanwendung realisiert werden.

Zu Beginn wird eine allgemeine Übersicht zu JavaScript gegeben, um einen Einstieg in dieses umfassende Thema zu ermöglichen. Neben der Entstehung wird hier unter anderem auch auf die JavaScript-Engine, welche die Ausführung von Code beschleunigen soll, sowie auf die Möglichkeit der asynchronen Programmierung in JavaScript eingegangen, welche es ermöglicht, Aufgaben parallel auszuführen. Außerdem wurde neben dem Datenformat JSON auch die Obermenge TypeScript behandelt. Das erste Kapitel wurde dann mit einem Überblick über die heutigen Einsatzgebiete von JavaScript abgeschlossen. Hier sollte deutlich gemacht werden, dass bei dem breiten Feld der Webentwicklung kein Weg mehr an JavaScript vorbeiführt.

In Kapitel Zwei dieser Projektarbeit wird das Thema Full Stack Development ausführlich vorgestellt. Zu Beginn zeigen einige Beispiele aus der Praxis, dass Full Stack JavaScript längst eine ernstzunehmende Alternative für Webanwendungen geworden ist, welche auch Internetriesen wie Netflix oder PayPal nutzen. Anschließend wird kurz das Client-Server-Modell beschrieben, um danach genauer die clientseitige Webentwicklung zu untersuchen. Neben den grundlegenden Bestandteilen (JavaScript, HTML und CSS) ist zudem einen umfassender Überblick über die drei populärsten Frameworks React, Angular sowie Vue gegeben. Schwerpunkt lag dabei auf den individuellen Eigenschaften, Stärken und Schwächen und es wurde aufgezeigt, wo und wieso jedes dieser Frameworks unterschiedliche Einsatzgebiete hat, die ihre Daseinsberechtigung unterstreichen. Danach folgt die serverseitige Webentwicklung, welche es ermöglicht dynamische Websites zu erstellen. Node.js, was genutzt wird, um JavaScript-Code auch außerhalb vom Browser auszuführen, sowie die drei verbreitetsten Node-Frameworks werden hier detaillierter beschrieben, sowie deren Vor- und Nachteile erläutert. Hiernach ging es weiter mit dem Thema Datenbanken. Neben einer kurzen Übersicht inklusive möglicher Einsatzgebiete wurde hier auch ein kurzer Einblick zu SQL- und No-SQL-Lösungen gegeben. Abgeschlossen wurde das Zweite Kapitel mit einer Auswahl aus den beliebtesten Stack-Varianten. Neben einer kurzen Vorstellung dieser drei Ansätze sollte diese Gegenüberstellung auch dabei helfen, einen geeigneten Stack für die in Kapitel Drei folgende Planung der Web-Anwendung zu finden.

Das dritte Kapitel *FULL STACK PRAXISPROJEKT "SKIMP"* diente dazu, das in den vorherigen Kapiteln erlangte Wissen in der Praxis umzusetzen. Die Aufgabe war es, eine Anwendung zu entwickeln, die aus einer dynamischen Clientseite, welche mit einem Server mit Datenbank kommuniziert, besteht. Nachdem eine Menge unterschiedlicher Ideen zur Umsetzung gesammelt wurde, fiel die Entscheidung eine Webanwendung namens „Skimp“ zu realisieren, die es möglich macht, Rabatt- und Gutscheincodes mit anderen zu teilen. Um eine Idee zu bekommen ob und wie bereits ähnliche Anwendungen umgesetzt wurden, recherchierten man im Internet, mit dem Ergebnis, dass es zwar eine Reihe Vergleichbares gibt, jedoch bisher niemand das Teilen von personalisierten Rabattcodes möglich macht. Mit diesem Wissen wurde daraufhin eine Liste von pflicht bzw. optionalen Anforderungen an „Skimp“ angelegt, welche eine effiziente und zugleich angenehme Bedienung durch die Nutzer ermöglichen sollten. Aufgrund der in dem vorherigen Kapitel gesammelten Eindrücke, sollte zur Entwicklung der Anwendung der MERN-Stack (MongoDB, Express, React, Node) eingesetzt werden. Anschließend wurden dann alle wichtigen Technologien vorgestellt und anhand ihrer grundlegenden Funktionen beschrieben. Im Unterkapitel Konzeption wird unter anderem erläutert, dass auf der Seite des Servers eine einzige JavaScript-Datei genutzt wird, die Clientseite jedoch aufgrund der Übersichtlichkeit in einzelne Bestandteile zerlegt worden ist. Zudem wird zusätzlich kurz das Design der UI erläutert, welches möglichst schlicht und minimalistisch sein sollte, um so einen hochwertigen Eindruck zu vermitteln und eine gute User Experience zu bieten. Danach wurden die einzelnen Funktionen der Webapp beschrieben und schließlich erklärt wie das Deployment der Anwendung funktioniert und auf welchen öffentlichen Servern Skimp bereitgestellt wird.

Das letzten Kapitel dieser Projektarbeit behandelt die Probleme, die während der Umsetzung des Praxisprojekts auftraten und beurteilt danach die finale Anwendung bezüglich dessen, ob die zuvor gestellten Anforderungen erfüllt wurden. Eine Schwierigkeit war, dass zu Beginn der Projektarbeit nur Grundwissen über das Thema vorhanden war, was eine umfangreiche Recherche erforderlich machte. Neue Syntax-Formen in JavaScript erschwerten den Lernprozess zusätzlich. Neben ein paar kleineren Problemen, wie unübersichtlichem Code, was zur Behebung noch mehr Zeit in Anspruch genommen hätte, sowie wenig hilfreichen Fehlercodes, gab es nur vereinzelte Bugs, sodass das Praxisprojekt trotzdem erfolgreich abgeschlossen werden konnte. Als sehr hilfreich stellten sich vor allem die zahlreichen Tutorials, Foren sowie Fachbücher heraus. Die Menge an Beiträgen im Web unterstreicht zudem den Eindruck einer weltweit sehr aktiven Community, die den Fortschritt der Entwicklung mit JavaScript weiter vorantreibt. Dass das Praxisprojekt „Skimp“ erfolgreich beendet werden konnte, war auch das Ergebnis der Evaluation, welche zeigt, dass alle Pflichtanforderungen erfüllt wurden. Darüber hinaus konnten ebenfalls einige optionale Anforderungen, wie das Kopieren eines Codes in die Zwischenablage oder einen optionalen Dark Mode umgesetzt werden.

5.2 Ausblick

Um den Prototypen letztlich zu einer vollumfänglichen, skalierbaren Anwendung werden zu lassen, fehlen jedoch noch einige Features. Dazu gehört vor allem das progressive Laden der Coupons, da bei einer Vielzahl von Coupons das Laden der Seite ungemein länger dauert und somit zu einer schlechteren Nutzererfahrung führen würde. Eine große Menge an Gutscheincodes führt dementsprechend zu einer hohen Anzahl an Datenbankeinträgen, wobei es wichtig wäre, dass Codes die abgelaufen sind, automatisch aus der Datenbank gelöscht werden. Darüber hinaus wäre es sinnvoll dem User die Möglichkeit zu geben, einen Account anzulegen, um so beispielsweise Coupons speichern zu können. Eine Progressive Web App (PWA) um die Anwendung auf seinem Endgerät zu installieren wäre ebenfalls eine Idee Wert.

Mit Blick auf die Zukunft des Full Stack Developments mit JavaScript kann man von einer noch weiter steigenden Verbreitung ausgehen. Dass das Interesse zu diesem Thema ungebrochen ist, zeigen auch die Google Search Trend aus Kapitel 2 (siehe Abbildung 19). Der Trend der Full Stack Entwicklung birgt einiges an Potential. So ist die Entwicklung von vollständigen Anwendungen mit heutigen JavaScript-Technologien und Frameworks zur Unterstützung bereits problemlos möglich und könnte sich durch die aktive und wachsende Community in den nächsten Jahren noch weiter entwickeln. Auch wenn ein Entwickler nur im Frontend oder Backend tätig ist, erleichtert ihm das Wissen des Full Stacks die Umsetzung und gibt einen hilfreichen Überblick über die Kommunikation zwischen den beiden Seiten.

6. ANHANG

6.1 Eigenständigkeitserklärung

Hiermit versichern wir, dass wir die vorliegende Projektarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt haben. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Die Projektarbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.



Nils Mattmüller,



Tim Peters,



Eric Meyer

Offenburg, den 20.Juli.2021

6.2 Abbildungsverzeichnis

Abbildung 1: Vorgang des Compiling und der Ausführung von JavaScript	6
Abbildung 2: Objektsyntax in JavaScript	6
Abbildung 3: Syntax einer Callback-Funktion in JavaScript	8
Abbildung 4: Ergebnis der Ausführung der synchronen Funktion	8
Abbildung 5: Syntax einer asynchronen Callback-Funktion in der Lambda-Schreibweise	8
Abbildung 6: Ergebnis der Ausführung der asynchronen Funktion	9
Abbildung 7: Erstellen und auflösen eines Promise-Objekts	10
Abbildung 8: Callback-Funktion wird an Promise gebunden	10
Abbildung 9: Lebenszyklus eines Promise-Objekts	10
Abbildung 10: Definieren einer asynchronen Funktion mit den Schlüsselwörtern “async” und “await”	11
Abbildung 11: Deklaration des Datentyps einer Variablen in TypeScript	12
Abbildung 12: Deklarieren der Datentypen der Argumente in einer Funktion	12
Abbildung 13: Fehlerbehandlung in TypeScript vor der Kompilierung	13
Abbildung 14: Darstellung eines JSON-Objekts	14
Abbildung 15: JSON Datentypen.....	15
Abbildung 16: Erstellen eines Objekts in JavaScript	15
Abbildung 17: Zugriff auf Objekteigenschaften in JavaScript	16
Abbildung 18: Desktopanwendung des Prototyping- u. Designtools Figma	17
Abbildung 19: Google Search Trends für den Begriff “Full Stack Development”	19
Abbildung 20: Frontend und User Interface der Webanwendung von Netflix	20
Abbildung 21: Neues Frontend von LinkedIn Mobile	21
Abbildung 22: Client-Server-Architektur	22
Abbildung 23: Synchrone und asynchrone Client-Server-Kommunikation	24
Abbildung 24: Grundlegende Bestandteile und Technologien einer Webseite	25
Abbildung 25: Beispielhafte Darstellung der einzelnen Knoten des Document Object Models	26
Abbildung 26: Vergleich verschiedener Frontend Frameworks in den StackOverflow-Trends	27
Abbildung 27: JSX-Syntax in React, die ermöglicht HTML-Elemente in JavaScript zu schreiben	28
Abbildung 28: Beispielhafter Aufbau einer komponentenbasierten Anwendung	29

Abbildung 29 : Unterschied zwischen einem Framework und einer Library	30
Abbildung 30 : Vergleich zwischen unidirektionaler und bidirektionale Datenbindung	31
Abbildung 31 : Vue.js Template-Syntax	32
Abbildung 32 : Rendern der UI-Elemente mit Hilfe des virtuellen DOM	33
Abbildung 33 : Schichtenarchitektur des LAMP-Stack	36
Abbildung 34 : Auslagern von I/O-Operationen in die Eventschleife	41
Abbildung 35 : Einbinden des HTTP-Moduls über die “require”-Funktion und erstellen eines Servers	43
Abbildung 36 : Einrichten des Servers und Absenden einer Response	44
Abbildung 37 : Relationsschema	50
Abbildung 38 : Graphische Darstellung des Aufbaus einer MERN-Anwendung	55
Abbildung 39 : Graphische Darstellung des Aufbaus einer MEAN-Anwendung	56
Abbildung 40 : Graphische Darstellung des Aufbaus einer PERN-Anwendung	57
Abbildung 41 : Grundlegende implementierung einer React-Komponente als Lambda-Funktion	63
Abbildung 42 : Verändern des States in React mit State Hooks	64
Abbildung 43 : Ausführen von Nebeneffekten mit der Efect Hook	65
Abbildung 44 : Absenden eines GET-Requests mit dem Axios-Modul	65
Abbildung 45 : Architektur des Node-Frameworks Express	66
Abbildung 46 : Grundlegendes Setup eines Server mit Hilfe von Express	67
Abbildung 47 : Beispielhafte Routen um auf verschiedene URL´s und HTTP Methoden zu reagieren	68
Abbildung 48 : Definieren eines Objekt-Schemas mit Mongoose	70
Abbildung 49 : Einzelne React-Komponenten des Applikations-Frontends	71
Abbildung 50 : Benötigte Routen für die Funktionalität des Backends	72
Abbildung 51 : Datenmodell eines Coupon-Dokuments	72
Abbildung 52 : Logo, Farbpalette und Font-Family der Webanwendung	73
Abbildung 53 : Input-Felder als “Controlled-Components”	74
Abbildung 54 : POST-Request mit Eingabedaten senden und direktes Aktualisieren	75
Abbildung 55 : Verarbeiten der Daten des Request-Bodys	75
Abbildung 56 : Dokument im Datenbank-Cluster	75
Abbildung 57 : GET-Request und Verarbeiten der Antwort durch aktualisieren des States	75
Abbildung 58 : Coupons der Datenbank abrufen	76

Abbildung 59 : Erstellen der "CouponList"-Komponente aus dem Objekt-Array.....	76
Abbildung 60 : Übergeben der ID als URL-Parameter	77
Abbildung 61 : Einfach einlösbare Coupons löschen	77
Abbildung 62 : Bedingte Darstellung in Abhängigkeit des States "searchTerm".....	78
Abbildung 63 : Zugriff auf die Portnummer aus den Server-Umgebungsvariablen.....	79
Abbildung 64 : Definition des Server-Startpunkts in der Datei package.json.....	79
Abbildung 65 : Einfügen der Server-URL anstelle des Localhost	80
Abbildung 66 : Globales State-Management mit Redux	81
Abbildung 67 : Leere Dokumente in der Datenbank	83
Abbildung 68 : Mockup der Anwendung auf verschiedenen Endgeräten	83
Abbildung 69 : Geöffnetes und vollständig ausgefülltes Nutzer-Formular	84
Abbildung 70 : Validierung der Pflichtfelder.....	85
Abbildung 71 : Responsive Darstellung auf einem mobilen Endgerät.....	86
Abbildung 72 : Coupon-Verhalten bei Nutzerinteraktion	86
Abbildung 73 : Suchfunktion der Anwendung	87

6.3 Quellenverzeichnis

- [1] Wikipedia-Autoren (2002): JavaScript, in: Wikipedia, [online] <https://de.wikipedia.org/wiki/JavaScript> [13.05.2021]
- [2] Mathias Schäfer (2015): Eine kurze Geschichte von JavaScript, in: Webkrauts, [online] <http://webkrauts.de/artikel/2015/eine-kurze-geschichte-von-javascript> [13.05.2021]
- [3] Rainer Hahnenkamp (2019): JavaScript Engines: Performance-Steigerung der Browser, in: Heise, [online] <https://m.heise.de/developer/artikel/JavaScript-Engines-Performance-Steigerung-der-Browser-4264792.html?seite=all> [15.05.2021]
- [4] Mathias Schäfer: JavaScript/Ajax – SELFHTML-Wiki, in: selfhtml, [online]
- [5] Wikipedia-Autoren (2002): JavaScript, in: Wikipedia, [online] <https://en.wikipedia.org/wiki/JavaScript> [14.05.2021]
- [6] Wikipedia contributors (2021): ECMAScript, in: Wikipedia, [online] <https://en.wikipedia.org/wiki/ECMAScript#ES2015> [14.05.2021]
- [7] Benedikt Meurer, Mathias Bynens (2018): JavaScript-Engines: Die Turbolader für den Browser, in: t3n, [online] <https://t3n.de/magazin/javascript-engines-turbolader-fuer-browser-247188/> [23.05.2021].
- [8] Golo Roden (24.07.2015): Einführung in die asynchrone JavaScript-Programmierung, in: Heise, [online] <https://www.heise.de/developer/artikel/Einfuehrung-in-die-asynchrone-JavaScript-Programmierung-2752531.html?seite=all> [07.05.2021]
- [9] Golo Roden (16.03.2020): Node.js - Tutorial, Folge 2: Erste Schritte, in: YouTube, [Video] https://www.youtube.com/watch?v=APwRg37ShSY&list=PL6QrD7_cU23kaZ05MvixcoJ5vctRD1qgC&index=3 [08.05.2021]
- [10] Stack Abuse (24.04.2020): So schreiben Sie asynchronen Code in Node.js, in: Digital Ocean, [online] <https://www.digitalocean.com/community/tutorials/how-to-write-asynchronous-code-in-node-js-de> [12.05.2021]
- [11] Promises benutzen, in: MDN Webdocs, [online] https://developer.mozilla.org/de/docs/Web/JavaScript/Guide/Using_promises [18.04.2021]
- [12] Understanding JavaScript Promises, in: NodeJS, [online] <https://nodejs.dev/learn/understanding-javascript-promises> [18.04.2021]
- [13] async function, in: MDN Webdocs, [online] https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Statements/async_function [18.04.2021]
- [14] Olena Bochkor, Veikko Krypczyk (2019): Webapp-Tutorial 4: Typsicheres JavaScript mit TypeScript, in: Entwickler.de, [online] <https://entwickler.de/online/javascript/web-app-tutorial-typescript-javascript-579888415.html> [05.04.2021]
- [15] Jana Tillmann (25.09.2017): Was ist eigentlich TypeScript?, in: t3n, [online] <https://t3n.de/news/eigentlich-typescript-859869/> [05.04.2021]

[16] David Choi: *Fullstack React, TypeScript and Node: Build cloudy-ready web applications using React 17 with Hooks and GraphQL*, 1. Aufl. Birmingham, United Kingdom: Packt Publishing, 2020.

[4] (2021): Modules, in: TypeScript [online] <https://www.typescriptlang.org/docs/handbook/modules.html> [07.04.2021]

[17] Holger Schwichtenberg, Jörg Krause: *Moderne Webanwendungen für .NET-Entwickler: Server-Anwendungen, Web APIs, SPAs & HTML-Cross-Platform-Anwendungen mit ASP.NET, ASP.NET Core, JavaScript, TypeScript & Angular*, 3. Aufl. Boston, United States: O'Reilly dpunkt, 2018

[18] (2021): Modules, in: TypeScript [online] <https://www.typescriptlang.org/docs/handbook/modules.html> [07.04.2021]

[19] Wikipedia-Autoren (2017): JavaScript Object Notation, in: Wikipedia, [online] [https://de.wikipedia.org/wiki/JavaScript_Object_Notation#:~:text=Die%20JavaScript%20Object%20Notation%20\(JSON,existieren%20in%20allen%20verbreiteten%20Sprachen](https://de.wikipedia.org/wiki/JavaScript_Object_Notation#:~:text=Die%20JavaScript%20Object%20Notation%20(JSON,existieren%20in%20allen%20verbreiteten%20Sprachen) [12.04.2021]

[20] JSON Data Types, in: w3 schools, [online] https://www.w3schools.com/js/js_json_datatypes.asp [12.04.2021]

[21] JavaScript JSON, in: w3 schools, [online] https://www.w3schools.com/js/js_json.asp [12.04.2021]

[22] Monika Steinberg (27.06.2012): 10 JavaScript Anwendungen für mehr als Websites, in: t3n, [online] <https://t3n.de/news/10-javascript-anwendungen-mehr-397964/> [10.04.2021]

[23] Andreas Domin (25.04.2018): Was ist eigentlich eine Single-Page-Webanwendung, in: t3n, [online] <https://t3n.de/news/single-page-webanwendung-1023623/> [13.04.2021]

[24] Hartmut Schlosser (2018): JavaScript-Anwendungen für den Desktop: Electron 3.0 ist da, in: Entwickler.de, [online] <https://entwickler.de/online/javascript/electron-3-579860600.html> [13.04.2021]

[25] (2021): Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML und CSS entwickeln, in: Electron.js, [online] <https://www.electronjs.org/> [14.04.2021]

[26] Alexander Sinsel: *Das Internet der Dinge in der Produktion: Smart Manufacturing für Anwender und Lösungsanbieter*, 1 Aufl. Berlin, Deutschland: Springer Vieweg, 2019

[27] Volker Andelfinger, Till Hänisch: *Internet der Dinge: Technik, Trends und Geschäftsmodelle*, 1. Aufl. Wiesbaden, Deutschland: Springer Gabler, 2014

[28] Alexander Neumann (15.04.2016): Studie: Java, C und JavaScript sind die Top-Sprachen im Internet der Dinge, in: Heise, [online] <https://www.heise.de/developer/meldung/Studie-Java-C-und-JavaScript-sind-die-Top-Sprachen-im-Internet-der-Dinge-3175329.html> [15.04.2021]

[29] Olaf Göllner (09.09.2013): Espruino: Quelloffenes JavaScript-Board, in: Heise, [online] <https://www.heise.de/make/meldung/Espruino-Quelloffenes-JavaScript-Board-1952427.html> [02.05.2021]

[30] Serverseitige Webseitenprogrammierung, in: MDN Webdocs, [online] <https://developer.mozilla.org/de/docs/Learn/Server-side> [16.04.2021]

[31] Google Trends, [online] <https://trends.google.de/trends/explore?geo=DE&q=Full%20Stack%20Development> [14.06.2021]

[32] Anke Töpfer (04.03.2020): 10 Unternehmen, die Node.js für ihren Erfolg nutzen, in: Flycats [online] <https://www.flyacts.com/10-unternehmen-die-node-js-fuer-ihren-erfolg-nutzen> [18.05.2021]

- [33] (04.04.2019): *Top-Companies that used Node.js in Production*, in: YouTeam [online]
<https://youteam.io/blog/top-companies-that-used-node-js-in-production/> [18.05.2021]
- [34] Kristofer Baxter (05.08.2015): *Making Netflix.com faster*, in: Netflix Technology Blog [online]
<https://netflixtechblog.com/making-netflix-com-faster-f95d15f2e972> [21.05.2021]
- [35] Stefan Luber (2020): *Was ist das Client-Server-Modell?*, in: IP-Insider, [online]
<https://www.ip-insider.de/was-ist-das-client-server-modell-a-940627/> [29.05.2021]
- [36] Abts, Dietmar (2015): *Masterkurs Client/Server-Programmierung mit Java*, [Ausschnitt online]
https://bilder.buecher.de/zusatz/22/22613/22613774_lese_1.pdf.
- [37] (28.08.2016): *Was ist Frontend - was Backend*, in: Frontend GmbH [online]
<https://www.frontend-gmbh.de/blog/frontend-und-backend/> [05.05.2021]
- [38] Azat Mardan: *Full Stack JavaScript: Learn Backbone.js, Node.js and MongoDB*, 1. Aufl. New York, USA: Apress, 2015
- [39] Chris Northwood: *The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer*, 1. Aufl. New York, USA: Apress, 2018
- [40] (20.03.2020): *Was ist das Document Object Model*, in: Ionos [online]
<https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-das-document-object-model-dom/>
 [08.05.2021]
- [41] (09.04.2021): *Introduction to client-side frameworks*, in: MDN Webdocs [online]:
https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction [15.05.2021]
- [42] Isaak Tappert (01.01.2020): *Angular vs. React vs. Vue.js – JavaScript Frontend Frameworks im Vergleich*, in: Tappert IT [online] <https://tappert-it.de/2020/01/01/angular-react-vue-javascript-frontend-frameworks-vergleich/> [25.05.2021]
- [43] *Stack Overflow Trends*, in: Stack Overflow [online]
<https://insights.stackoverflow.com/trends?tags=reactjs%2Cangular%2Cvue.js> [10.05.2021]
- [44] Jana Tillmann (12.08.2017): *React vs. Angular – wann ist was besser?*, in: t3n [online]
<https://t3n.de/news/react-vs-angular-besser-845241/> [12.05.2021]
- [45] Oliver Sturm (2019): *React - Mal was anderes als Angular*, in: Entwickler.de [online]
<https://entwickler.de/online/javascript/react-angular-579862960.html> [16.05.2021]
- [46] *Components and Props*, in: React Docs [online] <https://reactjs.org/docs/components-and-props.html>
 [16.05.2021]
- [47] Ferdinand Malcher, Johannes Hoppe, Danny Koppenhagen: *Angular: Grundlagen, fortgeschrittene Themen und Best Practices*, 2. Aufl. Heidelberg, Deutschland: dpunkt, 2019
- [48] Veikko Krypczyk, Olena Bochkor (2019): *Web-App-Tutorial: Eine Anwendung mit Angular bauen*, in: Entwickler.de [online] <https://entwickler.de/online/javascript/web-app-tutorial-angular-579898837.html>
 [20.05.2021]
- [49] *Databinding - einfach/bidirektional*, in: Angular [online]
<https://angular.de/buecher/angularjs-buch/databinding/> [20.05.2021]

- [50] Robin Böhm (19.10.2020): Angular-Tutorial für Einsteiger, in: Angular [online] <https://angular.de/artikel/angular-tutorial-deutsch/> [21.05.2021]
- [51] Lars Peterke: Vue.js kurz und gut, 1. Aufl. Heidelberg, Deutschland: dpunkt, 2019
- [52] Jana Tillmann (01.08.2018): Was ist eigentlich Vue.js?, in: t3n [online] <https://t3n.de/news/vuejs-1097879/> [24.05.2021]
- [53] Ralph Steyer: Webanwendungen erstellen mit Vue.js: MVVM-Muster für konventionelle und Single-Page-Webseiten, 1. Aufl. Wiesbaden, Deutschland: Springer Vieweg, 2019
- [54] GeeksforGeeks (2020): Difference between Server Side Scripting and Client Side Scripting, in: GeeksforGeeks, [online] <https://www.geeksforgeeks.org/difference-between-server-side-scripting-and-client-side-scripting/> [02.06.2021]
- [55] Server-Side vs. Client-Side: Die beiden Scripting-Techniken im Vergleich (2020): in: Ionos, [online] <https://www.ionos.de/digitalguide/websites/web-entwicklung/server-side-und-client-side-scripting-die-unter-schiede/> [02.06.2021]
- [56] Mikhail Mitra (12.03.2018): LAMP/MEAN Stack: Business and Developer Perspective, in: Matra Labs, [online] <https://www.mantralabsglobal.com/blog/lamp-vs-mean-stack-business-and-developer-perspective/> [05.05.2021]
- [57] IBM Cloud Education (09.05.2019): LAMP Stack, in: IBM, [online] <https://www.ibm.com/cloud/learn/lamp-stack-explained> [05.05.2021]
- [58] What is Full Stack?, in: w3 schools, [online] https://www.w3schools.com/whatis/whatis_fullstack.asp [05.05.2021]
- [59] Stephan Augsten (08.12.2020): Was ist Node.js?, in: Dev Inside, [online] <https://www.dev-insider.de/was-ist-nodejs-a-972703/> [23.04.2021]
- [60] Sebastian Springer: The State of Node, in Entwickler.de, [online] <https://entwickler.de/leseproben/state-node-js-239361.html> [23.04.2021]
- [61] Sebastian Springer: Node.js: Das Praxisbuch, 2. Aufl. Bonn, Deutschland: Rheinwerk Verlag, 2016
- [62] A brief history of Node.js, in NodeJS, [online] <https://nodejs.dev/learn/a-brief-history-of-nodejs> [23.04.2021]
- [63] Node.js-Fachbegriffe einfach erklärt, [online] <https://www.arocom.de/fachbegriffe/webentwicklung/nodejs> [25.04.2021]
- [64] Tom Hughes-Croucher, Mike Wilson: Einführung in Node.js, 1. Aufl. Köln, Deutschland: O'Reilly, 2012
- [65] Fernando Doglio (12.08.2020): The JIT in JavaScript: Just in Time Compiler, in: Bits and Pieces, [online] <https://blog.bitsrc.io/the-jit-in-javascript-just-in-time-compiler-798b66e44143> [26.04.2021]
- [66] (31.03.2013): Node.js: Das JavaScript-Framework im Überblick, in: t3n [online] <https://t3n.de/magazin/serverseitige-javascript-entwicklung-nodejs-einsatz-231152/> [26.04.2021]
- [67] (11.09.2018): Node.js - Einführung: Die JavaScript-Laufzeitumgebung im Überblick, in: Ionos, [online] <https://www.ionos.de/digitalguide/websites/web-entwicklung/einfuehrung-in-nodejs/> [22.04.2021]

- [68] Golo Roden: *Node.js & Co: skalierbare, hochperformante und echtzeitfähige Webanwendungen professionell in JavaScript entwickeln*, 1. Aufl. Heidelberg, Deutschland: dpunkt, 2012
- [69] Nat Friedman (16.03.2020): *NPM is joining GitHub*, in: *The GitHub Blog* [online] <https://github.blog/2020-03-16-npm-is-joining-github/> [15.05.2021]
- [70] MDN Contributors (07.04.2021): *Express/Node introduction*, in: *MDN Web Docs*, [online] https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction [06.05.2021]
- [71] Tejas Kaneriya (10.09.2021): *Top 3 Most Popular Nodejs Frameworks and When to Use Them*, in: *Dzone*, [online] <https://dzone.com/articles/top-3-most-popular-nodejs-frameworks-and-when-to-u> [06.05.2021]
- [72] Liz Parody (22.03.2019): *Choosing the right Node.js Framework: Express, Koa, or Hapi?*, in: *Nodesource*, <https://nodesource.com/blog/Express-Koa-Hapi> [08.05.2021]
- [73] Doug Wade (18.05.2021): *Who is using Koa? #22*, in: *Github*, [online] <https://github.com/koajs/discussions/issues/22> [06.05.2021]
- [74] *Introduction*, in: *Socket.io*, [online] <https://socket.io/docs/v4/index.html> [08.05.2021]
- [75] *Socket.IO*, in: *Stackshare*, [online] <https://stackshare.io/socket-io> [08.05.2021]
- [76] *What is Database*, in: *Oracle*, [online] <https://www.oracle.com/de/database/what-is-database/> [03.06.2021]
- [77] (2019): *Datenbanken: Wozu man sie braucht und welche Arten es gibt*, in: *Ionos*, [online] <https://www.ionos.de/digitalguide/hosting/hosting-technik/datenbanken/> [04.06.2021]
- [78] Volker Sanger: *Datenspeicherung • Datenbanken*, in: *MI-Learning*, [online] <https://dbcontent.mi.hs-offenburg.de/DB/datenspeicherung/datenmanagement/fakten/index.html> [04.06.2021]
- [79] Wikipedia-Autoren: *SQL*, in: *Wikipedia*, [online] <https://de.wikipedia.org/wiki/SQL> [08.06.2021]
- [80] TenMedia: *SQL Datenbank*, in: *TenMedia*, [online] <https://www.tenmedia.de/de/glossar/sql-datenbank> [11.06.2021]
- [81] Stefan Luber (2017): *Was ist SQL?*, in: *Dev-Insider*, [online] <https://www.dev-insider.de/was-ist-sql-a-586264/> [06.06.2021]
- [82] Wikipedia-Autoren: *SQL*, in: *Wikipedia*, [online] <https://de.wikipedia.org/wiki/SQL#Sprachelemente%20und%20Beispiele> [08.06.2021]
- [83] Volker Sanger: *SQL • Datenbanken*, in: *MI-Learning*, [online] <https://dbcontent.mi.hs-offenburg.de/DB/sql/grundoperationen/fakten/index.html> [13.06.2021]
- [84] Volker Sanger: *NoSQL • Datenbanken*, in: *MI-Learning*, [online] <https://dbcontent.mi.hs-offenburg.de/DB/nicht-relationaleddb/no-sql/fakten/index.html> [13.06.2021]
- [85] *NoSQL Datenbankenerklrung*, in: *MongoDB*, [online] <https://www.mongodb.com/de-de/nosql-explained> [14.06.2021]
- [86] *MERN Stack*, in: *MongoDB*, [online] <https://www.mongodb.com/mern-stack> [21.04.2021]

- [87] Dr. Faith Gedikli (28.01.2016): *DER MEAN STACK*, in: adesso, [online] <https://www.adesso.de/de/news/blog/der-mean-stack.jsp> [26.04.2021]
- [89] pcpiyush1106 (05.09.2021): *What is PERN Stack?*, in: GeeksforGeeks, [online] <https://www.geeksforgeeks.org/what-is-pern-stack/> [21.04.2021]
- [90] PostgreSQL, in: Wikipedia, [online] <https://de.wikipedia.org/wiki/PostgreSQL> abgerufen: [23.04.2021]
- [91] Vorteile von PostreSQL, in: Cybertec, [online] <https://www.cybertec-postgresql.com/de/postgresql-uebersicht/vorteile-von-postgresql/> [21.04.2021]
- [92] Joe Morgen (12.05.2020): *Einrichten eines React-Projekts mit Create React App*, in: Digital Ocean, [online] <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-react-project-with-create-react-app-de> [01.06.2021]
- [93] *What is Babel?*, in: Babel.js Docs [online] <https://babeljs.io/docs/en/> [01.06.2021]
- [94] Aswin G (11.12.2019): *Webpack and Babel - What are they, and how to use them with React*, in: Aswin G Medium [online] <https://medium.com/@agzuniverse/webpack-and-babel-what-are-they-and-how-to-use-them-with-react-5807afc82ca8> [01.06.2019]
- [95] Stoyan Stefanov: *React - Up and Running: Building Web Applications*, 2. Aufl., Sebastopol, USA: O'Reilly, 2021
- [96] Nils Hartmann, Oliver Zeigermann: *React: Grundlagen, fortgeschrittene Techniken und Praxistipps – mit TypeScript und Redux*, 2. Aufl., Heidelberg, Deutschland: dpunkt, 2019
- [97] *Einführung in Hooks*, in: React Docs [online] <https://de.reactjs.org/docs/hooks-intro.html> [03.06.2021]
- [98] Paul Halliday (12.11.2020): *Verwenden von Axios mit React*, in: Digital Ocean [online] <https://www.digitalocean.com/community/tutorials/react-axios-react-de> [06.06.2021]
- [99] *React Hook Form - Performant, flexible and extensible forms with easy-to-use validation*, in: React Hook Form [online] <https://react-hook-form.com/> [06.06.2021]
- [100] William Le (24.12.2020): *How To Use the req Object in Express*, in: Digital Ocean [online] <https://www.digitalocean.com/community/tutorials/nodejs-req-object-in-expressjs> [10.06.2021]
- [101] Vasan Subramanian: *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node*, 2. Aufl., New York, USA: Apress, 2019
- [102] *Die Datenbank für moderne Anwendungen*, in: MongoDB [online] <https://www.mongodb.com/de-de> [11.06.2021]
- [103] *Cross-Origin Resource Sharing (CORS)*, in MDN Webdocs [online] <https://developer.mozilla.org/de/docs/Web/HTTP/CORS> [11.06.2020]
- [104] *Eine Einführung in Mongoose für MongoDB und Node.js*, in: Accentsconagua [online] <https://www.accentconagua.com/articles/code/an-introduction-to-mongoose-for-mongodb-and-node-js.html> [12.06.2021]
- [105] *4.x-API*, in: Express [online] <https://expressjs.com/de/api.html> [12.06.2021]
- [106] Remy Sharp: *Nodemon*, in: npm [online] <https://www.npmjs.com/package/nodemon> [14.06.2021]

[107] Manuel Bieh (2019): *Formulare*, in: *React - lernen und verstehen* [online]
<https://lernen.react-js.dev/die-grundlagen/formulare> [11.07.2021]