Ausführlicher Vergleich zwischen einem Node.js-basierten Softwarestack und anderen serverseitigen Stacks für die Umsetzung von Webapplikationen

von

Jannis Günsche

Eine besondere Lernleistung im Fach Informatik
bei Herr Lohrbächer

Alfred-Delp Schule
Dieburg
Schuljahr 2020/21

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

In dieser Ausarbeitung werde ich ausführlich Vor- und Nachteile, sowie grundlegende Unterschiede, von verschiedenen *Softwarestacks* für die Webentwicklung hervorheben. Einer der Softwarestacks wird auf Node.js-basieren und steht bei diesem Vergleich im Mittelpunkt. Ich werde außerdem auf die verschiedenen Funktionsweisen der Stacks eingehen und welche Möglichkeiten die einzelnen Technologien bieten. Dafür habe ich zeitgleich eigene Webanwendungen in Verbindung mit verschiedenen Softwarelösungen entwickelt, damit ich an diesen Unterschiede direkt veranschaulichen kann.

1.2 Hintergrund

Vor der Umsetzung einer Webapplikation muss man sich für einen Softwarestack entscheiden. Unter einem Softwarestack versteht man den Stapel (engl. stack) an aufeinander aufbauende Softwarekomponenten. Diese bilden gemeinsam die serverseitige Plattform, das sogenannte *Backend*, auf welcher zum Schluss die Webapplikation laufen wird.[Wik20c] Dabei muss sich das Backend nicht nur auf einen Softwarestack beschränken, sondern kann auch aus mehreren Stacks bestehen, welche dann gemeinsam einen größeren Stack bilden.

Je nachdem für welchen Stack man sich entscheidet, können einzelne Funktionen und Teile der Webapplikation besser oder schlechter umgesetzt werden.[Upw15] Wenn man einen ungeeigneten Softwarestack gewählt haben sollte, ist manche Umsetzung umständlich und aufwendig. Im schlimmsten Falle können Funktionen vorerst gar nicht umgesetzt[1] werden, da der für den Server ausgewählten Stack nicht die benötigten Eigenschaften bietet. Die Entscheidung für einen Softwarestack ist auch meistens endgültig, weil eine spätere Konvertierung gleichfalls sehr aufwendig ist. Also kann der für das Projekt richtig ausgewählte Stack schon vor der eigentlichen Entwicklung mögliche Hürden umgehen.

1.3 Struktur der Ausarbeitung

Zuerst werde ich die Softwarestacks und die Projekte, welche ich parallel für diese Ausarbeitung entwickelt habe, vorstellen. Somit kann ich im darauffolgenden Vergleich auf einzelne Hürden der jeweiligen Projekte eingehen und die Vor- und Nachteile einer Softwarelösung direkt ansprechen.

Dabei sollte man beachten, dass ich bei der Umsetzung dieser Projekte entweder auf (für mich persönlich) bewährte Methoden zurückgegriffen habe, oder mir die gängigsten Methoden der Softwareentwicklung in den einzelnen Bereichen angeschaut und benutzt habe. Es sei auch gesagt, dass man Funktionen auf verschiedene Weisen umsetzen kann, welche unter Umständen die von mir erteilte Bewertung positiv aber auch negativ beeinflussen hätte können. Alle Umsetzungen der Projekte sind auch als GitHub-Repository

[1] Mit „vorzeitig nicht umgesetzt“ meine ich beispielsweise die Implementierung von WebSockets, ohne zuvor eine eigene Bibliothek für WebSockets zu schreiben oder eine dritte Bibliothek zu verwenden. Im späteren Verlauf werde ich noch einmal diesen Aspekt aufgreifen.

über die jeweils angegebene URL aufzufinden. Die beigelegte *Readme*-Datei beinhaltet nochmal die vollständige Zusammensetzung des Softwarestacks und Screenshots der Projekte.

Diese schriftliche Ausarbeitung ist auch digital unter dem GitHub-Repository zu finden:

https://github.com/jgteam/bell--paper

Um nur die Abbildungen digital einzusehen, sind diese unter „Alle Abbildungen in der Übersicht“ verfügbar.

2 Vorstellung der drei Softwarestacks

2.1 Node.js-basierender Stack

2.1.1 Die Softwarekomponenten

Ein häufig genutzter Software Stack, welcher Node.js beinhaltet, heißt *MEAN*.[IBM19; Wik20b] Die einzelnen Buchstaben stehen hier für die benutzen Softwarekomponenten. Diese sind aber meistens im Namen nicht nach Funktion geordnet, sondern so, dass man sich den Namen besser behalten und aussprechen kann. Hier sind nochmal alle Teilkomponenten in abweichender Reihenfolge aufgeführt:[2]

Buchstabe	Softwarename	Beschreibung
N	Node.js	Node.js in die Komponente, welche es ermöglicht JavaScript-Quelltext auf dem Server auszuführen. Es handelt sich hierbei um eine Laufzeitumgebung, welche die sogenannte *V8-Engine* von Google für das Ausführen von JavaScript verwendet.[aro] Somit wird das Backend in JavaScript[3] geschrieben.
E	Express	Das *Express-Framework* bietet uns eine Reihe an vorgefertigten Funktionen, welche für das Bauen eines Webservers benötigt werden. Funktionen die sich zum Beispiel um HTTP-Anfragen kümmern. Ohne ein solches Framework müsste man sich selbst um die Implementierung der Verarbeitung von Nutzeranfragen kümmern, unter Zuhilfenahme von den Node.js bereitgestellten Funktionen.

[2] Hierbei handelt es sich nur um die Hauptkomponenten. Es wurden neben der in der Tabelle aufgeführten Software auch noch andere Module und Bibliotheken für die Umsetzung genutzt. Eine ausführliche Auflistung der genutzten Software ist jeweils im Abschnitt oder auf der GitHub-Seite der Projekte zu finden.

M	MariaDB	Bei *MariaDB* handelt es sich um die Datenbank. Hier können neben Nutzerdaten auch andere wichtige Informationen gespeichert, manipuliert und abgerufen werden. Diese Softwarekomponente ist die einzige, welche wir in diesem Vergleich in allen Softwarestacks[4] vorfinden werden.
A	AngularJS	Bei unserer letzten Komponente haben wir ein JavaScript-*Frontend*-Framework. Dieses kann für die Umsetzung einer Webapplikation sehr vieles erleichtern. Hauptsächlich werden solche Frameworks genutzt um Daten aus der Datenbank dynamisch anzeigen zu können. So aktualisieren sie zum Beispiel Inhalte bei Änderungen in der Datenbank automatisch im Browser, ohne das der Endnutzer die Seite neu laden, oder der Entwickler explizit diese Funktion implementieren muss.

Wir werden aber kein Frontend-Framework mit dieser umfangreichen Funktionsweise benötigen, weshalb wir aus dem eigentlichen *MEAN*-Stack das *A*, und somit *AngularJS*, weglassen werden. Folglich wird unser auf Node.js-basierender Softwarestack nur *MEN* heißen.

2.1.2 Praktische Umsetzung einer MEN Webapplikation

Nachdem man auf seiner Maschine Node.js und *NPM*[5] installiert hat[6], ist der Aufwand der Vorbereitung einer Webapplikation welche auf Node.js basiert sehr gering. Das liegt unter anderem daran, dass man für sein Projekt nur einen einzigen Projektordner anlegt, in welchem dann alle benötigten Dateien im Laufe der Entwicklung abgelegt werden. Nicht wie bei anderen Softwarelösungen, gibt es bei Node.js keine Konfigurationsdateien oder ähnliche Einstellungsmöglichkeiten außerhalb der Projektordners. So findet man in dem Projektordner seine JavaScript-Dateien, welche die Serverseite ausmachen und standardmäßig mit `node serverfile.js` ausgeführt werden können. Auch die statischen Dateien für die Clientseite sind im Projektordner zu finden. Dabei könnte es sich um HTML und CSS, sowie Bilder oder ähnliche Ressourcen wie JavaScript handeln. Währenddessen generiert aber auch NPM seine eigenen Dateien. So wird nach dem Initialisierungsbefehl `npm init` die *package.json* im Hauptverzeichnis erzeugt. Bei dem Hinzufügen von NPM-Modulen werden diese in der generierten Datei vermerkt. Das ist die Konfigurationsdatei der Applikation und ist im Stammverzeichnis des Projektordners zu finden.

[3] Es besteht aber auch die Möglichkeit sein Backend-Quellcode in TypeScript zu verfassen, wird aber nicht standardmäßig von Node.js unterstützt und muss erst nachgerüstet werden.

[4] Trotzdem wird eine Datenbank nicht in allen Projekten vorkommen.

[5] Kurz für Node package manager, auf welchen ich später im Vergleich noch eingehen werde

[6] Ich werde für diesen und den folgenden Stacks nicht auf die Aufsetzung der Datenbank eingehen, da dieser Schritt bei allen Stacks identisch ist und nichts zum Vergleich beitragen würde.

Projekt 3D Koordinatensystem: /package.json (Vereinfacht[7]):

```
{
  "name": "bell--3d-coordinate-system--nodejs",
  "version": "1.0.0",
  "main": "main.js",
  "scripts": {
    "start": "node main.js",
    "dev": "nodemon main.js"
  },
  "author": "Jannis Guensche",
  "dependencies": {
    "express": "^4.17.1",
    "nodemon": "^2.0.6",
    "socket.io": "^3.0.1"
  }
}
```

Man sieht, dass in der *package.json* viele allgemeine Informationen über das Projekt stehen. Unter *scripts* steht zum Beispiel der *start*-Eintrag, welcher `npm main.js` ausführt, wenn wir `npm start` als Befehl abschicken. Die verschiedenen Einträge sind praktisch, wenn man schnell auf einen Produktionsstart oder einen Entwicklungsstart, beispielsweise mit *Debugging*-Ausgaben, zugreifen will, da es nicht selten ist, dass die verschiedenen Startbefehle durch zusätzliche Argumente länger werden können. Unter *dependencies* sehen wir noch die vermerkten Module, welche die Applikation benötigt. Diese werden direkt mit der Versionsnummer versehen, welche man auch noch manuell anpassen kann, falls man eine feste Version benutzen muss oder entweder nur *Patch*-, *Minor*- oder doch alle und somit *Major*-Updates von Modulen zulassen möchte.[NPM] Die benötigten Module sind im Projektordner im Unterordner *node_modules* zu finden, nachdem sie von NPM heruntergeladen wurden.

2.2 Deno-basierender Stack

2.2.1 Die Softwarekomponenten

Der *MOD*[8]-Softwarestack hat sehr viele Parallelen zu dem vorherigen *MEN*-Softwarestack und besteht auch aus drei Hauptkomponenten:

[7] Vereinfacht: Für den Kontext irrelevante Zeilen entfernt. In weiteren Codeausschnitten auch Kommentare und Quellcode für ein vereinfachtes Verständnis umgeschrieben.

[8] Bei „MOD“ handelt es sich um keinen etablierten Stacknamen, weil Deno unter anderem noch eine relative neue Softwarelösung ist.

Buchstabe	Softwarename	Beschreibung
D	Deno	Deno erfüllt denselben Zweck wie Node.js und die Benutzung ist fast identisch, auf Grund dessen, dass Deno sowie Node.js beides von Ryan Dahl entwickelt wurde und Deno im Jahr 2020 in gewisser Weise als ein Nachfolger von Node.js angesehen werden kann. Deno baut auch auf der *V8-Engine* für JavaScript auf, kann aber neben JavaScript auch nativ mit TypeScript verwendet werden.[Cie19]
O	Oak	*Oak* ist genauso wie *Express* ein Framework, welches sich auch um HTTP-Anfragen kümmert. Bei der Benutzung von *Oak* fallen Unterschiede zur Implementierung zwar auf, aber der grundlegende Aufbau beim Bearbeiten von verschiedenen HTTP-Anfragen ist fast derselbe wie bei *Express*.
M	MariaDB	**siehe Node.js Tabelle*

2.2.2 Praktische Umsetzung einer MOD Webapplikation

Um eine Deno Applikation bauen zu können, muss man nur Deno auf seiner Maschine installiert haben. Ähnlich wie bei Node.js hat man seinen Projektordner, in welchem alle Dateien drin sind. Was aber fehlt ist die *package.json* und der *node_modules*-Ordner. Somit generiert Deno keine Dateien in dem Projektordner. Das hält aber nicht davon ab beispielsweise selbst einen *deno_modules*-Ordner anzulegen, um in diesem Deno-Module von Drittanbietern zu speichern. Um nun ein Skript auszuführen, benutzt man einfach den Befehl `deno run serverfile.ts` .

2.3 Apache2/PHP-basierender Stack

2.3.1 Die Softwarekomponenten

Dieser Softwarestack unterschiedet sich am stärksten von den anderen zwei. Es handelt sich hier um eine Zusammensetzung von Apache2 und PHP, welche die Grundlage des *XAMP*-Stacks bilden. Auch in diesem Stack gibt es nur drei Hauptkomponenten, da es sich bei dem „X“ nur um die Kennzeichnung *Cross-Platform* handelt, während beispielsweise der *LAMP*-Stack explizit auf Linux-Rechner laufen würde.

Buchstabe	Softwarename	Beschreibung
A	Apache2	Bei Apache2 handelt es sich um ein HTTP-Webserver, welcher in den letzten 24 Jahren der populärste Webserver war und noch den größten Marktanteil von 24,6% (Stand: Dezember 2020) hat.[net20] Das kann sich aber bald ändern, da immer mehr Betreiber auf alternative Software wie zum Beispiel Node.js umsteigen und somit der Marktanteil von Apache2 seit etwa 2012 nahezu konstant abnimmt, wenn man sich die top eine Millionen meistgenutzten Seiten anschaut.[Abb. V]
P	PHP	PHP ist die Programmiersprache, welche statische HTML-Dokumente ergänzt und diese durch die Verarbeitung von PHP als Präprozessor mit Inhalten, beispielsweise aus Datenbanken oder Aufgrund von vorausgehenden Nutzereingaben, füllen kann. Somit ist es möglich dynamische HTML-Dokumente zu generieren, aber auch serverseitige Operationen über die PHP-Skripte laufen zu lassen.
M	MariaDB	**siehe Node.js Tabelle*

2.3.2 Praktische Umsetzung einer XAMP Webapplikation

Bei einem *XAMP*-Stack gibt es keine Projektordner, sondern feste Verzeichnisse für Log- und Konfigurationsdateien und ein Verzeichnis für die Serverdateien, auf die der Nutzer später über den Apache2-Webserver zugreifen kann. Dieses Verzeichnis wird in der Standardkonfigurationsdatei *000-default.conf* festgelegt.

Apache2-Installation unter Ubuntu: /etc/apache2/sites-available/000-default.conf (Vereinfacht):

```
<VirtualHost *:80>

  ServerAdmin webmaster@localhost
  DocumentRoot /var/www/html

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Hier wird durch das Definieren eines *VirtualHost* mit `*:80` jede eingehende Verbindung unter dem Port 80 nun unter diesen Einstellungen verarbeitet. *DocumentRoot* legt das

Stammverzeichnis fest, also in diesem Falle `/var/www/html`. Man kann weitere „Seiten“ anlegen, welche unter anderen Ports oder unterschiedlichen Domains und/oder Subdomains erreichbar sind, indem man neben der „000-default.conf“ weitere Konfigurationsdateien anlegt (Bsp.: *anotherpage.conf*) und diese dann mit `a2ensite anotherpage` aktiviert. Danach muss aber Apache2 neu gestartet oder neu geladen werden.

Neben den Apache2-Konfigurationsdateien ist die *php.ini*-Konfigurationsdatei auch wichtig. Diese bestimmt beispielsweise ob Dateiuploads erlaubt sind, und falls dies der Fall ist, welche Einschränkungen gesetzt sind.

PHP-Installation unter Ubuntu: /etc/php/7.4/apache2/php.ini (Vereinfacht):

```
; Maximum size of POST data that PHP will accept.
post_max_size=40M

; Whether to allow HTTP file uploads.
file_uploads=On

; Maximum allowed file upload size.
upload_max_filesize=40M
```

Bei dieser Konfiguration wären Dateiuploads, welche dem Größenlimit bis 40 Megabyte entsprechen, möglich.

3 Vorstellung der drei Webapplikationen

Für diesen Vergleich habe ich drei verschiedene Webapplikationen entwickelt, welche in verschiedenen Softwarestacks umgesetzt wurden.

Das Projekt *3D-Koordinatensystem* ist das einzige welches nur im Node.js-Stack umgesetzt wurde. Zugleich ist es auch das umfangreichste Projekt. Es veranschaulicht die Vorteile des Node.js-Stacks und wie diese eingesetzt werden können.

Die beiden Projekte *File-Manager* und *Markdown-CMS* habe ich neben dem Node.js-Stack auch in anderen Softwarelösungen umgesetzt, um im späteren Verlauf dieser Ausarbeitung einen direkten Vergleich zwischen einzelnen Umsetzungen ziehen zu können.

3.1 3D-Koordinatensystem

3.1.1 Über das Projekt

3D-Koordinatensystem ist eine Webapplikation, welche dem Nutzer ein dreidimensionales Koordinatensystem bereitstellt, worin er Punkte, Strecken und Ebenen einzeichnen und manipulieren kann. Es soll eine vereinfachte Alternative zu dem bekannten Programm *Geogebra* darstellen, wenn es um den dreidimensionalen Raum geht. Meine Applikation

soll leicht zugänglich sein und eine Zusammenarbeit von Gruppen ermöglichen. Auch wenn der Funktionsumfang von *3D-Koordinatensystem* geringer ist, gegenüber professioneller Software, bietet die Nutzung trotzdem einige Vorteile:

3.1.1.1 Webapplikation Da es sich bei *3D-Koordinatensystem* um eine Webapplikation handelt, kann man diese einfach über den Internetbrowser aufrufen. Die Applikation muss also nicht installiert werden.

3.1.1.2 Einfache Bedienung Die Benutzeroberfläche wurde so konzipiert, dass sich jeder Nutzer direkt zurechtfindet. Man ist nicht von ausufernden Menüs überwältigt und kann direkt mit der Erstellung von Punkten, Strecken und Ebenen im Koordinatensystem beginnen. Eingebaute *Tooltips* zeigen dem Benutzer passende Erklärungen, wenn diese mit ihrem Cursor über Schaltflächen fahren. Ein Benachrichtigungssystem zeigt allen Nutzern zu jedem Zeitpunkt was passiert.

3.1.1.3 Zusammenarbeit In der Zeit von Corona ist die persönliche Zusammenarbeit stark eingeschränkt. So kann man mit *3D-Koordinatensystem* sein Koordinatensystem mit mehreren Nutzern teilen und zeitgleich an diesem arbeiten. Dies ermöglicht einfaches Präsentieren und Erklären von Lösungen über das Internet in Echtzeit. Dazu wurde ein Raumsystem umgesetzt, welches Nutzer in virtuellen Räumen gruppiert und wichtige Informationen unter diesen bereitstellt. Der Ersteller eines Raumes, der sogenannte *Host*, hat außerdem die Möglichkeit den Bearbeitungsumfang seiner Raumgäste, also seiner *Guests*, temporär oder permanent einzuschränken und seine Ansicht direkt über die *Webapp*[9] zu spiegeln. Es werden bei der Spiegelung die Kamera- und Maus-Position übertragen, so dass alle Gäste das Koordinatensystem nicht nur aus demselben Blickwinkel sehen, sondern auch nachverfolgen können, worauf der Host in diesem Moment mit seinem Cursor zeigt.

Auch eine Zusammenarbeit in Nicht-Echtzeit ist möglich. Alle Nutzer haben die Möglichkeit den aktuellen Stand zu exportieren, sowie auch wieder zu importieren. So können auch mehrere Teile zu einer Datei zusammengeführt werden. Die Handhabung der Dateien ist dem Nutzer überlassen. Somit kann er selbst entscheiden, über welche Plattformen und/oder Speichermedien er seine Koordinatensysteme teilt und verbreitet.

3.1.2 Technische Hürden bei der Umsetzung

Die größte Hürde an diesem Projekt ist die Ermöglichung der Zusammenarbeit in Echtzeit über das Internet. Ein solcher Informationsaustausch zwischen mehreren Browsern und dem Server wird meistens mit WebSockets umgesetzt. Man muss sicherstellen, dass alle Nutzer eines Raumes zu jedem beliebigen Zeitpunkt genau denselben Stand haben. Am besten löst man dieses Problem, indem man immer nur einen Stand beziehungsweise eine Instanz des Koordinatensystems speichert und bearbeitet, welchen dann alle Nutzer einsehen können. Die Handhabung dieses Koordinatensystems wird in der Praxis entweder dem Server überlassen oder einem einzelnen Nutzer. Der Vorteil einer serverseitigen Handhabung ist die, dass wir von keinem Nutzer abhängig sind, erfordert aber eine

[9] Kurzform für Webapplikation.

komplexere Implementierung einer serverseitigen Handhabung der Datenstruktur des Koordinatensystems. Deswegen habe ich mich für die einfachere Lösung entschieden, dem Raumersteller, also dem Host, diese Aufgabe zu erteilen. Der Vorteil: Er als Nutzer hat schließlich schon ein funktionierendes Modell der Koordinatensystemhandhabung. Der Nachteil ist aber die damit einhergehende Abhängigkeit von dem einzelnen Nutzer, da er für die Kernfunktion der Applikation nun benötigt wird. Im Anhang habe ich eine visuelle Darstellung[Abb. III] eines Ausschnitts der Kommunikation zwischen Server und den Nutzern hinzugefügt. Es verdeutlicht die Rolle des Hosts und wie gewährleistet wird, dass alle Clients einheitliche Änderungen im Koordinatensystem vornehmen können.

3.1.3 Quellcode

Dieses Projekt habe ich ausschließlich in dem Node.js-Softwarestack umgesetzt und der Quellcode kann unter folgende URL auf GitHub eingesehen werden:

https://github.com/jgteam/bell--3d-coordinate-system--nodejs

3.1.3.1 Projektspezifische Software Für dieses Projekt habe ich mich an den beiden JavaScript-Bibliotheken *ThreeJS* und *jQuery* bedient. *ThreeJS* ermöglicht mir *DOM*[10]-Elemente in einer Szene im Raum darzustellen, welche dann mit einer vom Nutzer gesteuerten Kamera aus verschiedenen Blickwinkeln betrachtet werden kann. Die *jQuery*-Bibliothek hilft mir DOM-Elemente zu manipulieren und mit diesen zu interagieren. Außerdem benutzt das Projekt *socket.io*, sowohl auf der Serverseite (als NPM-modul), als auch auf der Clientseite, um zwischen beiden Instanzen einen Austausch von Nachrichten über WebSockets ohne große zeitliche Verzögerungen zu ermöglichen.

3.2 File-Manager

3.2.1 Über das Projekt

Der *File-Manager* ist ein kleineres Projekt. Es handelt sich hauptsächlich um eine *REST-API* mit einer minimalen/optionalen Weboberfläche. Die API erlaubt es Nutzerdateien auf den Server hochzuladen, welche dann wiederum über einen bereitgestellten Link von einem beliebigen Nutzer heruntergeladen werden kann. Das Projekt soll eine einfache Möglichkeit des *Filesharings* bieten oder für eine andere Applikation eventuell der Grundbaustein einer solchen Funktionalität sein. So könnte auch mein vorheriges Projekt *3D-Koordinatensystem* von einer Einbindung einer solchen Funktion profitieren, um Projektdateien ohne externen Datei-/Clouddienste zu teilen.

3.2.2 Technische Hürden bei der Umsetzung

Ich musste schauen, wie ich in den einzelnen Softwarestacks eine REST-API umsetze und über diese einen Datei-Upload, sowie -Download, ermöglichen kann. Dabei wird die Dateihandhabung die wahrscheinlich interessantere Aufgabe sein, da die verschiedenen Softwarelösungen jeweils auf eine andere Weise den Zugriff auf das Dateisystem ermöglichen.

[10] Abkürzung für „Document Object Model“. U. a. die Struktur HTML-Dokumente.

3.2.3 Quellcode

Da dieses Projekt einen kleineren Umfang hat, war es optimal dieses in allen drei Softwarestacks umzusetzen.

Alle Projektvarianten setzten *MariaDB* als Datenbankmanagementsystem ein und besitzen dasselbe Frontend, weshalb auch überall[11] wieder *jQuery* für die Manipulierung von DOM-Elementen zum Einsatz kommt, aber hauptsächlich auch um die Anfragen an die REST-API über JavaScript zu formen.

3.2.3.1 Umsetzung in Node.js *https://github.com/jgteam/bell--file-manager--nodejs*

Für die Umsetzung in Node.js habe ich die NPM-Module *body-parser*, *cookie-parser*, *express-fileupload*, *express-sessions*, *mysql* und *uuid* auf der Seite des Servers genutzt. Diese helfen mir mit der Dateihandhabung, Sessions, Session-Cookies, sowie mit der Kommunikation zur Datenbank.

3.2.3.2 Umsetzung in Deno *https://github.com/jgteam/bell--file-manager--deno*

Im Deno-Stack habe ich die Deno-Module *oak_upload_middleware*, *mysql*, *hash* und *sessions* über die jeweilige URL direkt geladen. Genauso wie bei Node.js helfen diese mir mit den Dateiuploads und mit der Datenbankmanipulation. Die Implementierung von Sessions und Session-Cookies ist aktuell im Deno-Stack zwar umgesetzt aber deaktiviert, da diese nicht wie gewünscht funktioniert haben.

3.2.3.3 Umsetzung in PHP (Apache2) *https://github.com/jgteam/bell--file-manager--php*

Bei der Umsetzung in PHP mit Apache2 als Webserver musste ich nur bei der Einrichtung darauf achten, dass Apache2 die benötigten Module für PHP und PHP die Module für *MySQLi* aktiviert haben. Ansonsten musste keine weitere Drittsoftware installiert werden, um dieselben Funktionen anbieten zu können.

3.2.3.4 Umsetzung in PHP (Apache2) - Alternative Version ohne JavaScript *https://github.com/jgteam/bell--file-manager--php--nojs*

Diese Version verzichtet auf JavaScript und somit auch auf *jQuery*. So werden die clientseitigen Anfragen nicht über JavaScript getätigt, sondern durch einen Seitenaufruf, welcher die Anfrage verarbeitet und anschließend den Nutzer wieder zum Formular leitet. Ansonsten wird dieselbe Software wie im normalen PHP/Apache2-Stack verwendet.

[11] Außer bei der alternativen PHP-Version ohne JavaScript.

3.3 Markdown-CMS

Markdown-CMS ist ein Content-Management-System das für überschaubare Webauftritte genutzt werden kann. Die Textinhalte der Webseite werden ausschließlich in Markdown[12]-Dateien gespeichert. Eine Datenbank kommt nicht zum Einsatz.

3.3.1 Über das Projekt

Im Gegensatz zu anderen CM-Systemen wie *Wordpress* benutzt dieses keine Datenbank und hat auch keine Administration, welche man über den Webbrowser aufrufen könnte. Alle Textinhalte werden in Markdown-Dateien gespeichert. So können die Markdown-Auszeichnungen genutzt werden, um Inhalte zu formatieren und zu gestalten. Zusätzlich können auch applikationseigene *Markdown-CMS*-Elemente genutzt werden, um dem Autor über den Standardformatierungen hinaus noch mehr Gestaltungsmöglichkeiten und Features anzubieten. Vorteil dieses CMS ist es, dass es durch die nicht vorhandene Datenbank und Administration weniger Fläche für Angreifer bietet und das Sicherheitsrisiko allgemein verringert. Um Änderungen an der Seite vornehmen zu können reicht einzig und allein der Zugriff auf einen separaten Unterordner der Webapplikation, beispielsweise über *SFTP*.

3.3.2 Technische Hürden bei der Umsetzung

Für dieses Projekt musste ich mich mit *Embedded JavaScript* vertraut machen, damit ich mit Node.js HTML-Templates erstellen kann. Darüber hinaus musste ich mir Gedanken machen, wie ich eigene Metadaten in Markdown-Dateien speichere und wie ich die Dateien dann wieder korrekt auslese und darstelle.

3.3.3 Quellcode

Dieses Projekt habe ich in Node.js und in PHP/Apache2 umgesetzt. Es soll zeigen, wie man in JavaScript und in PHP *templating* und *Server Side Rendering* betreiben kann.

3.3.3.1 Umsetzung in Node.js *https://github.com/jgteam/bell--markdown-cms--nodejs*

Für die Umsetzung habe ich die NPM-Module *ejs*, *showdown*, *escape-string-regexp*, *body-parser*, *express* und *uuid* auf der Seite des Servers genutzt. Dabei sind die beiden ersten Module die wichtigsten. *EJS* ermöglicht die Benutzung von *Embedded JavaScript*. *Showdown* wandelt unsere Markdown-Dateien in HTML um.

3.3.3.2 Umsetzung in PHP (Apache2) *https://github.com/jgteam/bell--markdown-cms--php*

Für diesen Stack habe ich nur die Bibliothek *Parsedown* benötigt, auch um Markdown-Dateien in HTML umwandeln zu können.

[12]Markdown ist eine Auszeichnungssprache

4 Der Vergleich

4.1 Grundfunktionsweise

Der **Apache2**-Stack unterschiedet sich grundlegend vom Node.js- und Deno-Stack. Während diese als eine Laufzeitumgebung auf dem Server ausgeführt werden. Dabei benutzen sie meistens Drittsoftware wie *Express* oder *Oak*, um die HTTP-Anfragen wie gewünscht abarbeiten zu können. Dagegen steht bei dem Apache2-Stack Apache2 an erster Stelle und ist bereits ein fertiger Webserver. Apache2 allein würde für eine Webapplikation, welche nur statische Dateien bereitstellen muss und einen sehr eingeschränkten Funktionsumfang hat, ausreichen. Dieser ist aber nicht in der Lage beispielsweise gezielt Daten aus einer Datenbank auszulesen. Hier kommt dann PHP ins Spiel. Apache2 holt sich PHP zu Hilfe, um bei jeder HTTP-Anfrage die Möglichkeit zu haben ein PHP-Skript ausführen zu können, anstatt nur eine statische Datei zu übergeben. Das heißt der funktionale Aufbau des Apache2/PHP-Stacks ist genau umgekehrt im Vergleich zu den Node.js- und Deno-Stacks, wie ich im stark vereinfachten Schaubild[Abb. IV] zeige. So werden bei dem PHP-Stack HTML-Dateien als PHP-Dateien abgespeichert und beinhalten neben HTML Quellcode auch PHP Quellcode-Passagen oder besteht rein aus PHP. Bevor nun die angefragte Datei vom Server zurückgeschickt wird, wird diese von dem PHP-Präprozessor im Vorhinein verarbeitet und dann erst geschickt. Das ermöglicht es dynamische Seiteninhalte[13] direkt bei der ersten Anfrage bereitstellen zu können.

Projekt Markdown-CMS (PHP-Stack): Artikel-Vorlage /templates/article.php:

```
1 <main>
2     <div class="container">
3         <div class="wrapper">
4             <?php
5             printDocumentHere();
6             ?>
7         </div>
8     </div>
9 </main>
```

Hier in der Artikel-Vorlage sieht man von Zeile vier bis sechs eine PHP-Passage, welche zwischen HTML-Quellcode steht. Wenn diese PHP-Datei angefragt wird, wird die Methode `printDocumentHere()` aufgerufen und kann an dieser Stelle über den `echo`-Befehl dynamisch generierten HTML-Quellcode oder Text einfügen.

Währenddessen hat man bei **Node.js** und **Deno** die Möglichkeit direkt bei Anfragen JavaScript-Quellcode ausführen zu können. Je nach implementierter Logik können dann verschiedene Dateien oder Antworten zurückgeschickt werden, indem man die Antwort in JavaScript formt. Um aber dynamische Seiteninhalte dem Nutzer zeigen zu können,

[13] Damit sind Inhalte auf einer Seite einer Webapplikation gemeint, welche zum Beispiel nutzerspezifische Informationen darstellen.

muss entweder der Browser mindestens eine weitere Anfrage dem Server zurückschicken, um beispielsweise benutzerspezifische Inhalte zu erhalten[14], oder die Seite/Antwort muss, wie bei PHP, serverseitig geformt werden. Auf diese Umsetzung bauen viele Frontend-Frameworks auf. Diese benutzen JavaScript im Browser, um nachträglich Inhalte und Informationen zu laden. Da es sich aber um Frontend-Frameworks handelt, können diese auch in Apache2/PHP umgesetzt werden, benötigen aber meistens eine Node.js-Installation für das „bauen“ der Applikation. Die andere Lösung ist serverseitig die Antwort zu rendern (server-side-rendering). Mit *Embedded JavaScript* kann man beispielsweise server-side-rendering betreiben, dabei baut *EJS* auf einen ähnlichen Ansatz wie PHP auf und ermöglicht innerhalb HTML-Quellcode JavaScript-Passagen zu platzieren.

4.2 Clientkompatibilität

Wie schon im vorherigen Punkt angesprochen, hat man beim PHP-Stack die Möglichkeit größere Antworten, wie ganze Webseiten, serverseitig zu formen. Somit muss der Browser keine vom Client ausgehenden Folgeanfragen dem Server über JavaScript schicken. Das erhöht die Kompatibilität enorm.

Auch wenn man davon ausgeht, dass alle modernen Browser JavaScript unterstützen, gibt es einige Nutzer, welche durch Browser-Addons die Ausführung von JavaScript grundsätzlich blockieren und somit **Node.js**- und **Deno**-Webanwendungen, welche auf solche clientseitigen Anfragen aufbauen, unbenutzbar machen. Zum Beispiel haben *NoScript*, *ScriptSafe* und *ScriptBlock* laut Googles *chrome web store* zusammen 350.000+ bekannte Nutzer (Stand: 24. Dezember 2020).[chr20a; chr20c; chr20b] Es besteht auch die Möglichkeit, dass die Nutzerzahl solcher JavaScript-Blocker wachsen könnte, da Nutzer immer mehr auf ihre Privatsphäre achten und die meisten Tracker JavaScript benötigen. Unter anderem kann die Ausführung von JavaScript im Browser die eigene Sicherheit im schlechtesten Fall verringern.

Das heißt aber nicht, dass Webapplikationen, welche auf dem **PHP**-Stack basieren, von diesem Problem nicht betroffen sein können. Die meisten modernen Webapplikationen benutzen in irgendeiner Form clientseitigen JavaScript-Code, was auch diese Applikationen unnutzbar machen könnte. Wenn man aber auf clientseitige JavaScript Ausführung verzichten möchte, um die höchste Kompatibilität des Projekts sicherzustellen, dann hat man mit dem Apache2/PHP-Stack bei weitem mehr nutzbare Funktionen, da eine Node.js- oder Deno-Applikation einfach zu eingeschränkt wäre und erst durch Drittsoftware wie *EJS* mit PHP mithalten könnte.

So ist das *File-Manager* Projekt auch im PHP-Stack auf JavaScript angewiesen. Wenn man dennoch versucht eine Datei über das Formular hochzuladen während man das Ausführen von JavaScript blockiert, erkennt man, dass weder die Datei auf den Server hochgeladen wurde, noch eine Fehlermeldung angezeigt wird.[Abb. VI] Das liegt daran, dass für das Absenden, aber auch für das Empfangen, sowie für das Anzeigen von Antworten,

[14] Ausnahme sind hier REST-APIs, welche so kurze Antworten zurückliefern, dass man diese direkt vollständig, ohne viel Aufwand, erzeugen kann.

JavaScript eingesetzt wird. Bei der alternativen Umsetzung im PHP-Stack, welche komplett auf JavaScript verzichtet, kann auch mit einem JavaScript-Blocker wie *ScriptSafe* die Anwendung im vollen Funktionsumfang benutzt werden.[Abb. VII] Für das Anzeigen der Antworten beziehungsweise für die Generierung der Tabelle wird PHP benutzt, welches auf dem Server läuft. So wird die Tabelle mit den Antworten serverseitig generiert und erst danach an den Browser geschickt.

So wie es PHP-Webapplikationen gibt, welche clientseitiges JavaScript benutzen, gibt es auch Node.js- und Deno-Anwendungen, welche auf JavaScript beim Client verzichten können. Mehr dazu im nächsten Abschnitt.

4.3 Server-Side-Rendering auch für Node.js und Deno

Einige Node.js-Webapplikationen werden im Browser gerendert, weil viele Node.js-Applikationen auf Frameworks wie *React*, *Angular* und *Vue* aufbauen. Das Rendern im Browser muss aber nicht immer der Fall sein. Es besteht die Möglichkeit mit **Node.js** oder auch **Deno** Inhalte wie HTML-Seiten serverseitig zu generieren, und die fertige Datei an den Client zu senden. Dabei können wir zum Beispiel auf *EJS*, wie im Anfang schon vorgestellt, zurückgreifen. So kann man wie in PHP Dateien mit HTML-Inhalten verfassen, die Passagen von JavaScript-Code besitzen können.

Projekt Markdown-CMS (PHP-Stack): Startseiten-Vorlage /templates/home.php (Ausschnitt):

```
1 <?php if($meta["ShowLatestWork"] != "false"): ?>
2   <div class="right">
3     <h2><?php echo $meta["LatestWorkHeading"]; ?></h2>
4     <?php projectCard(getLatestProject()); ?>
5     <a class="anchor-button view-all" href="<?= ROOT ?>mywork">
          View all</a>
6   </div>
7 <?php endif; ?>
```

In **PHP** sind die Passagen von `<?php` und `?>` eingeschlossen, und falls es eine Ausgabe in den HTML-Quellcode geben soll, kann man `echo` benutzen oder auch als Kurzform Funktionen und Variablen innerhalb von `<?=` und `?>` schreiben. Methoden und eingebettete PHP-Dateien haben die Möglichkeit `echo` zu benutzen, auch wenn diese nicht über einen `echo`-Befehl aufgerufen wurden. Außerdem können Ressourcen wie Variablen dateiübergreifend genutzt werden.

Derselbe Ausschnitt der Vorlage sieht mit *EJS* in Node.js folgendermaßen aus:

Projekt Markdown-CMS (Node.js-Stack): Startseiten-Vorlage /views/templates/home.ejs (Ausschnitt):

```
1 <% if(meta["ShowLatestWork"] != "false"){ %>
2   <div class="right">
3     <h2><%- meta["LatestWorkHeading"]; -%></h2>
4     <%- projectCard(getLatestProject()); -%>
5     <a class="anchor-button view-all" href="<%- ROOT -%>mywork">
          View all</a>
6   </div>
7 <% } %>
```

Hier sind die JavaScript-Passagen innerhalb der Klammern `<%` und `%>`. Falls hier eine Ausgabe getätigt werden soll, dann müssen die Klammern noch mit einem Bindestrich versehen werden, wie in Zeile drei, vier und fünf. Methoden einfach aufzurufen und Dateien einzubinden reicht nicht mehr aus, wie es bei PHP noch der Fall war, denn falls diese irgendeine Ausgabe tätigen wollen, müssen diese direkt schon mit den Ausgabe-Klammern gekennzeichnet sein. Außerdem können die *EJS*-Dateien nicht auf Ressourcen anderer Dateien zugreifen. Das führt dazu, dass alle Variablen und Funktionen bereitgestellt werden müssen, bevor die erste *EJS*-Datei gerendert wird:

Projekt Markdown-CMS (Node.js-Stack): Bereitstellung von app.locals vor dem Rendern der index.ejs innerhalb einer Express-Route in /main.js (Vereinfacht):

```
1 app.get(['/', '/mywork', '/contact', '/privacy-policy', '/legal-
      notice', '/page/:name', '/project/:name'], function(req, res){
2
3   // [...] Quellcode fuer die Vorbereitung der app.locals
4   // res.app.locals bereitstellen:
5
6   // Meta-Daten werden ausgelesen
7   res.app.locals.pageType = openPage_output.meta.Type;
8   res.app.locals.pageTitle = openPage_output.meta.Title;
9   res.app.locals.pageTitleMeta = pageTitleMeta;
10
11  // Restlicher Output der openPage()-Funktion
12  res.app.locals.meta = openPage_output.meta;
13  res.app.locals.fileBody = openPage_output.fileBody;
14  res.app.locals.fileText = openPage_output.fileText;
15  res.app.locals.fileElements = openPage_output.fileElements;
16  res.app.locals.uniquePlaceholder =
        openPage_output.uniquePlaceholder;
17
18  // index.ejs rendern und an den Client senden
19  res.render('index');
20 });
```

Wie hier gezeigt werden die Variablen über `app.locals` bereitgestellt. Bei diesen handelt es sich um anfragenspezifische Variablen, da diese dem `res`-JavaScript-Objekt zugeordnet werden. Allgemeine Ressourcen, welche statisch sind, werden außerhalb der Route direkt mit `app.locals.MYFUNCTION` bereitgestellt.

Auch wenn beide Technologien identische Ergebnisse liefern, hat hier PHP einen klaren Vorteil. Es ist einfacher in der Handhabung und klarer strukturiert. Auch die verschiedenen Klammertypen in *EJS* können schnell unübersichtlich werden und die Umsetzung von Ausgaben erschweren. Das kann man im folgenden Ausschnitt gut erkennen:

Projekt Markdown-CMS (Node.js-Stack): Aufbau vom Body in /views/index.ejs (Ausschnitt):

```
<body class="page--<%- pageType -%>">

  <!-- Header-Template aufrufen -->
  <%- include("components/header"); -%>

  <!-- Body-Template aufrufen -->
  <%

  // Je nach Seitentyp, wird ein anderes Template aufgerufen

  if(pageType == "home") {%>
    <%- include("templates/home");-%>
  <%} else if(pageType == "article" || pageType == "legal") {%>
    <%- include("templates/article");-%>
  <%} else if(pageType == "mywork") {%>
    <%- include("templates/mywork");-%>
  <%} else if(pageType == "contact") {%>
    <%- include("templates/contact");-%>
  <%}
   %>

  <!-- Footer-Template aufrufen -->
  <%- include("components/footer"); -%>

</body>
```

Im Vergleich ist das äquivalente in PHP einfacher zu lesen:

Projekt Markdown-CMS (PHP-Stack): Aufbau vom Body in /index.php (Ausschnitt):

```
<body class="page--<?= $pageType ?>">

  <!-- Header-Template aufrufen -->
  <?php include_once("components/header.php"); ?>
```

```
5
6   <!-- Body-Template aufrufen -->
7   <?php
8
9   // Je nach Seitentyp, wird ein anderes Template aufgerufen
10
11  if($pageType == "home") {
12    include_once("templates/home.php");
13  } elseif ($pageType == "article" || $pageType == "legal") {
14    include_once("templates/article.php");
15  } elseif ($pageType == "mywork") {
16    include_once("templates/mywork.php");
17  } elseif ($pageType == "contact") {
18    include_once("templates/contact.php");
19  }
20  ?>
21
22  <!-- Footer-Template aufrufen -->
23  <?php include_once("components/footer.php"); ?>
24
25 </body>
```

4.4 Asynchrone Codeausführung

Da JavaScript von Haus aus nur *single-threaded* ist, ist die Umsetzung von Logik und Codeausführung sehr stark eingeschränkt. Codepassagen können nicht parallel ausgeführt werden und werden somit von konkurrierenden Funktionen blockiert, die noch nicht fertig ausgeführt wurden. Trotzdem wird JavaScript oft für asynchrone Codeausführung benutzt, denn JavaScript-*Engines*, wie die von **Node.js** und **Deno** benutzte *V8-Engine*, führen die sogenannte Eventschleife (Eventloop) ein. Diese ist dafür zuständig die vorhergesehene Warteschlange (Queue) abzuarbeiten.[Rob14] Dadurch kann weiter Code ausgeführt werden, auch wenn beispielsweise eine Datenbank- oder API-Abfrage länger Zeit in Anspruch nimmt. In PHP würde eine solche Abfrage die weitere Ausführung von Quellcode blockieren.

Um das Potential der Eventschleife vollständig auszuschöpfen, wurden die *promises*, welche schon aus anderen Programmiersprachen bekannt sind, Ende 2013 auch in JavaScript eingeführt. Diese agieren ähnlich wie die aus JavaScript schon vertretenen *eventhandler*. Diese „Versprechen“ können erfolgreich ausgeführt oder abgelehnt werden, und führen demnach anschließend ihre *callbacks* aus, und das ohne parallellaufenden Code zu blockieren. Zusätzlich ist es auch gut zu wissen, dass diese *callbacks* auch noch ausgeführt werden, wenn das *promise* vollendet ist, aber die Callback-Funktion an sich noch gar nicht geladen war. Dies ist bei normalen *eventlistener* beziehungsweise bei *eventhandler* nicht der Fall. Bei diesen muss man gewährleisten, dass diese schon geladen sind, bevor das eigentliche Event eintritt.[Arc13]

Auch wenn die Einführung von *promises* die Funktionalität von JavaScript erweitert hat, war dies nicht die perfekte Lösung für die Ausführung von asynchronem Code, da die *callbacks* und ihre Initialisierung zu unübersichtlichen und komplizierten Verzweigungen führen können:

JavaScript Promise-Syntax:

```
1 myFunction().then(() => {
2 /* Auszufuehrender Quellcode, nachdem das Promise entweder
     Aufgeloest (Resolved) oder Abgelehnt bzw. Verworfen (Rejected)
     wurde. */
3 })
```

Diese Problematik hat man schließlich mit der Einführung von asynchronen Funktionen, welche durch die Schlüsselwörter *async* und *await* gekennzeichnet sind, gelöst.[Kan20a; Bor19]

So kann man asynchrone Funktionen definieren, in welchen durch das Markieren von Funktionen mit *await* explizit auf die Antwort dieser Funktion gewartet wird und erst nach dem Erhalten der Antwort der Code weiter ausgeführt wird. Diese Blockade ist aber nur innerhalb von asynchronen Funktionen möglich, weil diese den restlichen Programmablauf nicht beeinflussen.

Asynchrone Funktionen habe ich auch in meinem *File-Manager* Projekt benutzt. So habe ich im Node.js-Stack die Funktion `getFileName(fileid)` welche den zu der *fileid* gehörenden Dateinamen aus einer Datenbank ausliest und das Ergebnis in Form eines *promises* zurückgibt:

Projekt File-Manager (Node.js-Stack): getFileName(fileid)-Funktion in /main.js (Vereinfacht):

```
1 // Gibt entweder "false" zurueck (falls die Fileid nicht in der
     Datenbank hinterlegt ist) oder den Dateinamen, welcher in der
     Datenbank hinterlegt ist.
2 function getFileName(fileid) {
3     // Neues Versprechen erstellen, da die Funktion mit await
        aufgerufen wird
4     return new Promise(function (resolve, reject) {
5         // SELECT-QUERY ausfuehren
6         con.query(
7           "SELECT * FROM files WHERE id = " + con.escape(fileid),
8           function (err, result, fields) {
9               // Ergebnis ueberpruefen
10              if(result.length < 1) {
11                // Fileid nicht in der Datenbank gefunden
12                // return false;
13                  resolve(false);
```

```
14                } else {
15                  // Fileid und somit den Dateinamen gefunden
16                  // return filename;
17                    resolve(result[0].filename);
18                }
19            }
20          );
21      });
22  }
```

Man sieht, es wird direkt in Zeile vier ein neues *promise* erstellt, welches direkt durch `return` zurückgegeben wird. In Zeile sechs fängt die SQL-Abfrage an, welche nach der Ausführung die Funktion im zweiten Parameter ausführt. Diese wertet dann das Ergebnis der Abfrage aus und definiert das endgültige Ergebnis über die `resolve()`-Funktion. Über die `reject()`-Funktion kann man auch das *promise* erfüllen, würde aber das nächstmögliche `catch` auslösen.[Kan20b] Die `reject()`-Methode wird für das Handhaben von Fehlern benutzt. Beide Funktionen setzten aber den Status des *promises*, von *pending* auf *resolved* oder *rejected*. Asynchrone Funktionen geben automatisch immer ein *promise* zurück, auch wenn die Rückgabe nur über `return` umgesetzt wurde.[Wal16]

Diese Funktion rufe ich in einer asynchronen Funktion mit dem Schlüsselwort *await* auf, wie man im folgenden Codeausschnitt sehen kann. Das führt dazu, dass wirklich auf das Beenden der Funktion gewartet wird. Dies ist essenziell, weil die Funktion Datenbankabfragen ausführt. Diese Abfragen können durch beispielsweise eine zu langsame Datenverbindung oder durch Überlastungen des Datenbankservers länger andauern, was perfekt die Nutzung von *await* demonstriert.

Projekt File-Manager (Node.js-Stack): asynchrone Funktion der „/download/{fileid}“-Route in /main.js (Vereinfacht):

```
1  app.get('/download/:fileid', async function(req, res){
2      // DOWNLOAD
3
4      // fileid-Parameter in Variable schreiben
5      var fileid = req.params.fileid;
6
7      // Pruefen ob Datei/fileid existiert und falls ja, den Namen
           der Datenbank entnehmen
8      var filename = await getFileName(fileid);
9      //console.log(filename);
10
11     if(filename === false) {
12         // Datei ist nicht in der Datenbank vermerkt
13         // Error-Antwort erstellen
14         /* Auszufuehrender Quellcode */
15     } else {
```

```
16          // Datei ist in der Datenbank vermerkt
17          // Dateidownload-Antwort erstellen
18          /* Auszufuehrender Quellcode */
19      }
20  });
```

In Zeile eins sieht man erstmal das *async* Schlüsselwort, somit handelt es sich hierbei um eine asynchrone Funktion und *await* kann innerhalb dieser benutzt werden. In der achten Zeile rufen wir jetzt die oben schon vorgestellte Funktion über *await* auf. Wenn wir testweise mal das *await* weglassen und die Kommentierung des Logging-Befehles in Zeile neun aufheben, dann bekommen wir ein *promise* zurück, welches noch den *pending* Status hat. Somit wird das Ergebnis der Funktion auch für die Fallunterscheidung in Zeile elf unbrauchbar. Im Projekt, welches mit Deno umgesetzt wurde, kann man die Problematik noch besser nachvollziehen, da in der Konsole auch der Verbindungsstatus zum Datenbankserver ausgegeben wird.[Abb. I]

Abbildung I: Projekt File-Manager (Deno-Stack): Ausgaben in der Konsole, während die Funktion getFilePath(fileid) ohne *await* in console.log() aufgerufen wird.

```
C:\Users\janni\PhpstormProjects\bell--file-manager--deno>deno run --allow-read
--allow-write --allow-net --unstable main.ts
Check file:///C:/Users/janni/PhpstormProjects/bell--file-manager--deno/main.ts
INFO connecting 81[redacted]:3306
Promise { <pending> }
INFO connected to 81[redacted]:3306
```

Man sieht, dass zum Zeitpunkt der Konsolenausgabe das *promise* noch aussteht, da die Datenbankabfrage von dem Datenbankserver noch nicht stattgefunden hat. Erst nach der Konsolenausgabe vom *promise* hat sich die Applikation mit der Datenbank erfolgreich verbunden und konnte somit noch keine Daten abgreifen. Also ist der Funktionsaufruf mit *await* zu beachten. Neben der normalen Unterstützung von asynchronen Funktionen, hat Deno noch einen großen Unterschied. Deno unterstützt nämlich sogenanntes „top-level“ *await*. Das heißt, dass wir auf dem höchsten Level, dem globalen *scope*, *await* benutzen können, da Deno als große asynchrone Funktion den geschriebenen Code ausführt.[Tor20]

Wenn wir uns aber den Präprozessor **PHP** auf der anderen Seite mal ansehen, fehlt die Implementierung einer solchen Funktionsweise. PHP führt jede Zeile von Quellcode immer nacheinander aus und niemals parallel. Im Falle einer Datenbankabfrage, wie oben im Node.js- und Deno-Stack gezeigt, wird keine besondere Implementierung benötigt, da die Codeausführung immer auf die Vollendung wartet und streng synchron abläuft. So kann im folgenden PHP-Codeausschnitt die in Zeile drei ausgeführte Abfrage direkt in Zeile fünf verarbeitet werden, da diese Zeile erst ausgeführt wird, wenn die SQL-Abfrage fertig ist:

Projekt File-Manager (PHP-Stack): Ausschnitt der /scripts/down.php (Vereinfacht):

```php
// SQL-QUERY: Sucht die Daten zur fileid raus
$sql = "SELECT * FROM files WHERE id = '" .
    mysqli_real_escape_string($con, $fileid) . "'";
$res = mysqli_query($con, $sql);

if (mysqli_num_rows($res) > 0) {
    // SQL-QUERY hat ein Ergebnis geliefert
    /* Auszufuehrender Quellcode */
} else {
    // SQL-QUERY hatte kein Treffer in der Datenbank
    /* Auszufuehrender Quellcode */
}
```

Trotz dieser synchronen Codeausführung können mehrere Nutzer zeitgleich die PHP-Webapp benutzen. Das liegt daran, dass Apache2, welcher die HTTP-Request verarbeitet, für jede Anfrage einen eigenen Prozess erstellt, die wiederum parallel laufen können. Diese Aufteilung von Request-Abarbeitung kann aber schnell die Ressourcen des Servers aufbrauchen, was zum Beispiel zu Auslagerung von Daten aus dem schnellen RAM-Speicher auf den langsameren Daten-Speicher führen kann. Diese Auslagerung kann die Leistung des Webservers erheblich beeinträchtigen. Es ist auch möglich mehrere Requests mit einzelnen Threads, anstatt von ganzen Prozessen zu abzuarbeiten, was einiges an Ressourcen sparen kann. Dafür muss aber gegeben sein, dass die laufende PHP-Applikation „thread-safe“ ist, da mehrere Requests über Threads laufen und ihr *Environment* teilen, was zu Speicherüberlagerungen führen kann.[Mos11; The]

Eine unter Umständen bessere Alternative für den Apache2-Webserver ist der *nginx*-Webserver, welcher mehrere sogenannter „worker“ benutzt. Diese laufen jeweils in einem eigenen single-threaded Prozess und können trotzdem mehrere Anfragen gleichzeitig asynchron abarbeiten, ohne ein Sicherheitsrisiko einzugehen. Auf diese Weise kann der Webserver nochmal um einiges effizienter laufen.[jan15; Row14]

Auch wenn in unserem Apache2/PHP-Stack sich nur Apache2 um die parallele Ausführung von PHP-Prozessen kümmert, gibt es trotzdem Wege in PHP asynchronen Code ausführen zu können. So kann man schließlich einfach seinen eigenen neuen PHP-Unterprozess starten, welcher dann parallel zum Hauptprozess arbeitet. So gibt es auch Bibliotheken wie *github.com/spatie/async*, welche sogar ähnlichen Syntax wie die aus JavaScript bekannten *promises* anbieten. Aber auch bei dieser Lösung gibt es ein Problem. Zu viele Prozesse können im schlimmsten Fall die Applikation oder sogar der Server zum Absturz bringen, was aber durch sogenannte „pools“ verhindert werden kann. Diese benutzen eine Warteschlange, um Prozesse nacheinander abarbeiten zu können, was die Serverlast verringert.[Roo20; Sai20] Nennenswert ist aber auch *ReactPHP*. Eine PHP-Bibliothek, welche stark von Node.js inspiriert ist, und Funktionen wie Eventbasiertes (Eventloop) und *non-blocking* I/O aber auch *promises* mit sich bringt.[Rea]

4.5 NPM - Node package manager

Durch den *Node package manager* (kurz NPM) kann man die Funktionalität seiner **Node.js** Applikation ganz einfach ergänzen. Befehle wie `npm install express --save` können Pakete (hier *Express*) als Abhängigkeiten direkt in das Projekt integrieren[Expa] und über die `require()`-JavaScript-Funktion einfach darauf zugreifen.

Projekt 3D Koordinatensystem: JavaScript require-Funktion und definieren der Express-App in /main.js:

```
1 const express = require('express');
2 const app = express();
```

Die NPM-Bibliothek hat Mitte 2019 über eine Million verfügbare Pakete indexiert[TM19], welche alle open-source und somit komplett einsehbar sind.

Der Vorteil eines Package-Managers ist, dass bei einer Weitergabe des Projektes zu anderen Entwicklern oder zu Kunden nur die Applikation und nicht alle damit verbundenen Abhängigkeiten übertragen werden müssen. Über `npm install` ist es möglich alle in der *package.json* vermerkten Abhängigkeiten zu laden, welche zuvor von dem Entwickler über den Node package manager hinzugefügt wurden.

Die zentrale Speicherung und Verfügbarkeit durch die *NPM registry* nimmt den Entwicklern der Module die Aufgabe des Hostings ab, und ermöglicht es Importe über den eindeutigen Paketnamen zu tätigen.

Deno nutzt keine zentrale Lösung für den Import von Modulen. Um Module anderer nutzen zu können, muss man auf die Benutzung der offiziellen *ES Modules* von JavaScript zurückgreifen. Diese Syntax erlauben nicht nur Module zu importieren, sondern auch zu exportieren.[MDN20]

So kann man mit der JavaScript `export`-Funktion

Projekt File-Manager (Deno-Stack): JavaScript export-Funktion in /config.ts bzw. /config.SAMPLE.ts:

```
1 export { config };
```

mühelos eigene Module bereitstellen, welche dann lokal oder über die jeweilige URL importiert werden können. Ein Modulimport würde dann so aussehen:

Projekt File-Manager (Deno-Stack): JavaScript import-Funktion (lokal) in /main.ts:

```
1 import { config } from "./config.ts";
```

Projekt File-Manager (Deno-Stack): JavaScript import-Funktion (über eine URL) in /main.ts:

```
1 import { Application , Router , send } from "https://deno.land/x/
    oak/mod.ts";
```

Auch wenn die Entwickler ihre eigenen Module durch eigenständiges Hosting bereitstellen, kann man unter der offiziellen Seite von Deno unter *deno.land/x* seine Module veröffentlichen und für alle zugänglich machen.[Tor20]

Außerdem wird bei der Nutzung von Deno im Vergleich zu Node.js weniger Speicher durch die Pakete aufgebraucht. Deno hat nämlich einen zentralen Ordner ($DENO_DIR)[Cas20] in welchem alle jemals genutzten Module abgelegt werden. Unabhängig wie viele Deno-Projekte die einzelnen Module nutzen, werden diese nur einmal gespeichert, außer die Projekte benutzen verschiedene Versionen desselben Modules. Node.js speichert seine Pakete immer nur im Projektordner, was dazu führt, dass jedes Projekt die Abhängigkeiten immer wieder neu herunterladen muss, was einiges mehr an Speicher verbraucht.[Tor20]

Außerdem stellt Deno offizielle *Standard-Libraries* zu Verfügung.[Tor20; Dah20b] Dazu zählen Bibliotheken wie *http* und *ws* (kurz für WebSockets), welche mit den Paketen *express* und *socket.io* von Drittanbietern unter Node.js vergleichbar sind.[15]

Bei einem **Apache2**-Webserver hat man auch die Möglichkeit Module zu aktivieren und zu deaktivieren. Dabei handelt es sich aber hauptsächlich um Apache2-eigene Konfigurationen und Funktionen, welche auch von Apache2 bereitgestellt werden. Zusätzlich gibt es noch ein paar Module und Softwarelösungen von größeren Anbietern.[Aur16]

Die Apache2-eigenen Module werden direkt mit Apache mitgeliefert, und müssen demnach nur noch aktiviert und nicht heruntergeladen beziehungsweise installiert werden. [Thea] Dazu gehört beispielsweise auch das Modul *mod_rewrite*, welches den Server ermöglicht die URL zu manipulieren. Diese Funktion ist nützlich für Weiterleitungen oder *user-friendly* URLs. Wie die anderen Softwarestacks die URL-Manipulation ermöglichen und welche Unterschiede es unter diesen gibt, werde ich im Abschnitt 4.8 erläutern.

Beispiele für Module von Drittanbietern könnten zum einen *Googles mod_pagespeed*-Modul sein, welches Verzögerungen und Ladezeiten verringert oder *mod_php* von *The*

[15] *Zu einem späteren Zeitpunkt werde ich auch noch auf die Implementierung von WebSockets, sowie die Handhabung von HTTP-Request, in den verschiedenen Softwarestacks eingehen.*

PHP Group, welches den Apache-Webserver so konfiguriert, dass er Anfragen auf PHP-Skripte richtig bearbeiten kann.[Wik20a]

Um ein solches Modul, beispielweise *mod_php* auf einem Linux-Server, zu nutzen, muss man zuerst das Apache-Modul auf dem System installieren und danach über den `a2enmod`-Befehl aktivieren.[Rai19]

```
1 apt install libapache2-mod-php
2 a2enmod php
```

Das Installationsverfahren der Module ist von Modul zu Modul, sowie von Betriebssystem zu Betriebssystem immer unterschiedlich und erschwert die Einrichtung und das Benutzen von bestimmten Paketen.

Bei Node.js und Deno ist diese Benutzung von Modulen plattformübergreifend einheitlich, aber die große Anzahl an verfügbaren Modulen kann zu ungewollten Nachteilen führen. Sie heißen nicht grundlos *dependencies* (*Abhängigkeiten*), da man sich schließlich von den installierten Paketen abhängig macht. Kommt es zu einer ungewollten Änderung eines Paketes, so kann es zu Inkompatibilität und nicht gewolltem Verhalten der Applikation führen. Außerdem kann es auch vorkommen, dass Entwickler ihre Pakete zurücknehmen und von der NPM-Plattform löschen.[Sch16] Dementsprechend kann unter Umständen eine Applikation ohne eigene Gegenmaßnahmen vorerst nicht mehr realisiert werden.

Man kann zwar die Anzahl der selbst verwendeten Abhängigkeiten reduzieren, doch oft ist die tatsächliche Anzahl an Abhängigkeiten viel größer, da einzelne Pakete auch von anderen Paketen abhängig sein können. So werden zum Beispiel durch die Benutzung von dem Paket *Express* Version *4.17.1* allein schon 84 weitere Pakete mit installiert.[Abb. VIII] Um einen vollständigen Abhängigkeitsbaum zu generieren, kann man mit dem Befehl `npm list` ein solches Diagramm erzeugen. Es werden alle Pakete rekursiv aufgelistet und zusammengeführt. Falls mehrere Pakete dieselben Abhängigkeiten haben sollten, werden diese nur einmal aufgelistet und mit dem Kennzeichen „*deduped*“(Kurzwort für engl. *deduplicated*, Duplikate entfernt) versehen.[Zap18]

4.6 Duck Typing / Type Juggling

Alle drei Stacks haben das Problem des *Duck Typings*. Das bedeutet das Variablen keinem Typ wie String, Integer oder Boolean zugeordnet bekommen. Damit aber Vergleichsoperationen und ähnliches trotzdem funktionieren können, werden Daten automatisch *gecasted*, also je nach Situation zum passenden Typ umgeformt. Auch *Type Juggling* genannt, da die verschiedenen Typen jongliert werden.[rip] Dies kann zu ungewollten Problemen führen, da durch manipulierte Nutzereingaben das Verhalten der Applikation bewusst negativ beeinflusst werden kann. Dieses Problem kann durch Typkontrollen umgangen werden, was aber zu einer komplexeren Umsetzung führt. **Node.js** und **Deno** unterstützen die Möglichkeit TypeScript zu verwenden, was es dem Entwickler ermöglicht, bei Variablen, Parametern und Funktionen einen Datentyp mit anzugeben. Diese Implementierungsmöglichkeiten stehen dem **PHP**-Stack nicht zur Verfügung.

Projekt File-Manager (Deno-Stack): getFileDownloadURL-Funktion in /main.ts:

```
// Gibt die URL fuer den Dateidownload zurueck
function getFileDownloadURL(fileid: String): String {
    return rootURL + "download/" + fileid;
}
```

Hier wird zum Beispiel nur ein Parameter vom Typ *String* akzeptiert und die Funktion hat ebenfalls einen Rückgabewert des Typs *String*. Wenn man trotzdem versucht eine Nummer als Parameter zu übergeben, gibt Deno schon vor der Ausführung des Skriptes einen Error zurück, und meldet, dass die Datentypen nicht übereinstimmen.[Abb. IX] Somit kann man mit der Verwendung von TypeScript dieses Problem vorbeugen, welche mit automatischen *Casts* (Typ-Umformungen) in Verbindung stehen.

Um TypeScript unter Node.js verwenden zu können, muss zuerst *typescript* über NPM installiert werden.

4.7 Fehlerhandhabung

Wenn eine Nutzeranfrage einen serverseitigen Fehler auslöst, welcher nicht *gecatcht* oder in irgendeiner anderen Art und Weise verarbeitet wird, stürzt bei dem **Node.js**- und **Deno**-Stack im schlimmsten Falle die gesamte Serverapplikation ab. Wenn zudem der Server nicht automatisch nach einem Absturz neu gestartet wird, werden nicht nur alle momentanen Nutzeranfragen nicht mehr beantwortet, sondern auch keine Anfragen mehr angenommen.

Einen großen Vorteil besitzt hier **PHP**, denn hier wirken sich Fehler anders auf den laufenden Serverprozess aus. Wenn ein Fehler auftreten sollte, stürzt nur die einzelne Anfrage ab und kann logischerweise nicht bearbeitet werden. Trotzdem werden alle parallellaufenden und zukünftigen Anfragen immer noch bearbeitet.

4.8 Semantische URLs und Routen

Semantic URLs oder *clean URLs* können nicht nur die Webapp ästhetisch aufwerten, sondern haben auch darüber hinaus einen großen Vorteil in der *Search Engine Optimization*, kurz *SEO*.[Nym07] Es handelt sich bei den beschriebenen URL-Adressen um besser lesbare und benutzerfreundliche Varianten, welche immer noch dieselben Informationen beinhalten. So wird beispielsweise bei meinem *File-Manager* Projekt die Adresse

`example.com/scripts/down.php/?fileid=1234abcd`

(welche schwer zu entziffern ist) zu

`example.com/download/1234abcd`.

Es sind nur die nötigsten Informationen enthalten und man sieht eine klare Hierarchie. Außerdem muss der Nutzer bei der Eingabe nicht auf überflüssige Verzeichnisse, Dateiendungen und die richtige Syntax und Namen von den *GET*-Parametern achten, wie es bei der ersten Adresse der Fall wäre. Dadurch wird auch die Adresse um einiges kürzer und bei der Eingabe weniger fehleranfällig. Auch bei meinem *3D-Koordinatensystem* Projekt benutze ich benutzerfreundliche URLs.[Abb. II]

Abbildung II: Projekt 3D-Koordinatensystem: Raum-Seite „Room 0815“ und Adressleiste mit leicht lesbarer URL

Es ist direkt an der URL erkennbar, dass man sich in einem Raum befindet, nämlich im Raum mit der ID „0815“. Diese Art von Internetadressen wird nicht nur gerne von Nutzern gesehen, sondern Suchmaschinen wie Google legen auch großen Wert auf *semantic URLs* und sortiert die Suchergebnisse unter anderem nach der URL Struktur. Unter anderem klicken Nutzer auf einfach lesbare und klar verständliche Webadressen häufiger als auf unverständliche und lange Webadressen.[Goo20]

Wenn man solche *semantic URLs* umsetzen will, dann hat man mit **Node.js** und **Deno** einen klaren Vorteil gegenüber Apache2/PHP. Denn bei diesen muss der Entwickler alle verfügbaren Adressen als sogenannte Routen von Grund auf anlegen und hat volle Kontrolle über die Gestaltung. Hier erstmal Node.js:

Projekt 3D-Koordinatensystem: index.html-Route für die Raum-Seite (Vereinfacht):

```
1 // Eine gueltige Raum-URL verweist auf die index.html der Raum-
    Seite (Base-Ordner)
2 app.get('/room/:id([0-9]{3,})', function(req, res){
3     res.sendFile(__dirname + '/base/index.html');
4 });
```

Hier wird die Index-Route für die Raum-Seite über die *Express*-App mit der Funktion `app.get()` definiert. Der erste Parameter ist unsere URL, also „/room/{roomid}“, da wir durch den Doppelpunkt einen Parameter in der URL einbauen können, welcher in diesem Falle „id“ heißt. Zusätzlich können wir einen regulären Ausdruck angeben, welcher der Parameter innerhalb der URL befolgen muss, damit diese Route greift. Hier ist der Ausdruck „[0-9]{3,}“, also muss es eine Ziffernfolge mit mindestens drei Ziffern sein. Über `res`

(*result*) können wir die Antwort formen. In dieser Route schicken wir einfach die Datei *index.html* zurück. Falls wir ganze Ordner mit statischem Inhalt unter einer bestimmten Adresse verfügbar machen wollen, benutzen wir einfach die `express.static`-Methode.

Projekt 3D-Koordinatensystem: Route der statischen Dateien für die Raum-Seite (Vereinfacht):

```
1 // Eine gueltige Raum-URL bzw. "/room/" verweist auf den Ordner
      der Raum-Seite (Base-Ordner)
2 app.use(['/room/:id([0-9]{3,})', '/room/'], express.static('base')
      );
```

Das heißt, wenn der Browser zum Beispiel die *main.js*-JavaScript-Datei vom Server beziehen will, da diese im *HEAD-Html-Tag* über die URL `src='js/main.js'` der *index.html* verlinkt wurde, wird das hierdurch möglich gemacht. Es wird auf die in dem „base"-Ordner liegende Datei unter „base/js/main.js" über die URL „/room/js/main.js" [16] Zugriff gewährleistet. So sieht man auch an der Route, dass man durch ein Array mehrere mögliche Pfade zu derselben Route haben kann.

Bei **Deno** nutzen wir statt *Express Oak*. *Oak* bietet uns ähnliche Funktionalität, hat aber nicht denselben Funktionsumfang. So kann man für seine Routen keine Arrays übergeben, falls man über mehrere Adressen auf dieselbe Route zugreifen möchte. Außerdem gibt es keine von *Oak* bereitgestellte Lösung nur einzelne Dateien zu übermitteln, wie es bei *Express* mit `res.sendFile(file);` der Fall ist. Somit muss man die Übermittlung einer einzelnen Datei selbst implementieren.

Projekt File-Manager (Deno-Stack): hist.html-Route für die statische Verlauf-Seite (Vereinfacht):

```
1 app.get("/history", async function (context) {
2     // Verlaufs-Seite
3
4     // hist.html einlesen
5     var fileContent = await Deno.readFile(Deno.cwd() + "/hist.html
          ");
6
7     // hist.html uebermitteln
8     context.response.status = 200;
9     context.response.body = fileContent;
10    context.response.headers.set('Content-Type', 'text/html');
11 });
```

[16] Oder „/room/{roomid}/js/main.js" , falls in der Adresszeile hinter der RoomID sich noch ein Schrägstrich befindet.

Bei einem Webserver wie **Apache2** hat man schon einen vordefinierten Ordner, welcher wie ein statischer Ordner bei Node.js oder Deno gesehen werden kann. Auf alle enthaltenden Dateien können über den korrespondierenden URL-Pfad zugegriffen werden. Nun muss man die schon gegebene Handhabung von Apache2 abändern. Das macht man mit den sogenannten *.htaccess*-Dateien, welche man mit in das Webverzeichnis legt. Diese Datei beinhaltet Apache2-Konfigurationsbefehle, sogenannte *Directives*.[Mur14] Man legt *RewriteRules* (Umschreibregeln) an, welche die semantische URL erkennen, und dann die Anfrage zu der von Apache2 und PHP benötigten Form umformen.

Projekt File-Manager (Apache2/PHP-Stack): Umschreibungsregeln /.htaccess-Datei (Vereinfacht):

```
1 RewriteEngine On
2 # ...
3 RewriteRule ^download/(.*)$ ./scripts/down.php?fileid=$1
4 RewriteRule upload ./scripts/up.php
```

So wird in Zeile drei die URL von

`example.com/download/1234abcd`

wieder zurück zu

`example.com/scripts/down.php/?fileid=1234abcd`

umgeformt, damit Apache2 auf die richtige Datei verweisen kann und damit in der PHP-Datei selbst später über die `$_GET`-Variable auf den *fileid*-Parameter, welcher bei der Umformung überführt worden ist, zugegriffen werden kann.

4.9 REST-API

Eine *REST-API* ist eine Schnittstelle für den Informationsaustausch, zum Beispiel zwischen Server und Client. Sie benutzt für die Anfragen die HTTP-Kommandos[Tor] wie beispielsweise `POST`, `GET` oder `PUT`.

Bei der Erstellung einer solchen API kann man unter dem **Node.js**- und **Deno**-Stack einige Vorteile genießen. Denn wir können Routen erstellen, wie schon im vorherigen Abschnitt gezeigt, welche für die Schnittstelle genutzt werden können. Diese werden direkt einem HTTP-Kommando zugeordnet. Beispielsweise würde `router.post` nur bei POST-Anfragen und `router.put` nur bei PUT-Anfragen ausgelöst werden. Egal auf welchem Kommando die Route aufbaut, sind die übergebenen Parameter immer in der `Request`- (*Express*) und `Context`-Variable (*Oak*) zu finden.[tuta]

Bei **Apache2/PHP** hat der Entwickler nicht die Möglichkeit Routen zu erstellen. Wenn eine HTTP-Anfrage dazu führt, dass ein PHP-Skript ausgeführt wird, wird nicht darauf geachtet über welches HTTP-Kommando die Anfrage gesendet wurde. Es muss innerhalb

des Skriptes geprüft werden, um welche Art der Anfrage es sich handelt. Dafür steht einem die `$_SERVER["REQUEST_METHOD"]`-Variable zur Verfügung. Falls es sich um eine GET- oder POST-Anfrage handelt, werden die Informationen der Anfrage jeweils in den `$_GET`- und `$_POST`-Variablen hinterlegt. Ansonsten müssen die Informationen der Anfrage umständlich aus dem Body über `file_get_contents("php://input")` herausgelesen werden.

4.10 WebSockets

Das WebSocket-Protokoll ist ein Protokoll für beidseitigen Nachrichtenverkehr, um mit geringer zeitlicher Verzögerung Daten zwischen dem Browser und dem Webserver auszutauschen.[Fet11]

Node.js in Verbindung mit *socket.io* ist ein bekannter Lösungsansatz, um WebSockets in Webapplikationen entwicklerfreundlich einzubauen. Obwohl man in allen Stacks WebSockets ohne Drittsoftware implementieren kann, da alle das für WebSockets vorausgesetzte TCP-Protokoll unterstützen, können Lösungen wie *socket.io* bei der Umsetzung einiges erleichtern. So kümmert sich *socket.io* beispielsweise um einen erneuten Aufbau der Verbindung, falls diese getrennt wurde. Eingebaute Funktionen wie Socket-IDs und Socket-Räume helfen ungemein bei der Implementierung von eigenen Funktionen.

Projekt 3D-Koordinatensystem: socket.io-EventHandler für genehmigte Änderungen am Koordinatensystem in /main.js (Vereinfacht):

```
1 // changes -> Beinhaltet die Aenderungen und den aktuellen Stand
      von den benoetigten Objektzaehlern
2
3 // WebSocket: eintreffendes grantChange-Event
4 socket.on('grantChange', function(changes) {
5   // Der Raum in welchem sich der Sender dieses Ereignisses
        befindet
6   var room = socket.roomId;
7
8   // Schickt an alle Clients im Raum ein ausgehendes
        broadcastChange-Event, mit der neuen Aenderung
9   io.in(room).emit('broadcastChange', changes);
10 });
```

Ähnliche Pakete unter **Deno** wie *websocket* oder das von Deno selbst bereitgestellte *ws* können ebenso gut wie *socket.io* sein, bieten aber im Vergleich nur wenige Funktionen für die Handhabung von Clients und das Senden von Nachrichten.

Bei **PHP** ist die Bibliothek *Ratchet* sehr bekannt und kann auch dem Entwickler einiges an Aufwand abnehmen. Trotzdem ist die Benutzung von WebSockets unter PHP im Gegensatz zu Node.js und Deno ein großer Mehrwaufwand, unter anderem da PHP nicht eventbasiert ist und/oder asynchron läuft. Das ist auch der Grund warum *Ratchet*

auf der schon weiter oben genannten Bibliothek *ReactPHP* basiert[Rea], um auf solche Funktionen zugreifen zu können.

Zusammengefasst kann man sagen, dass bei WebSockets Node.js die mit Abstand beste Lösung ist, da Node.js mit seiner Runtime die perfekte Grundlage bildet und in Kombination mit *socket.io* am meisten Features bieten kann.

4.11 File-Upload und -Download

In allen drei Stacks können sowohl Dateiuploads also auch Dateidownloads umgesetzt werden. Die Implementierung einer solchen Funktion ist aber in alles Stacks unterschiedlich aufgebaut. Ein Dateiupload besteht eigentlich nur aus der Validierung der Datei und, falls diese positiv war, die anschließende Verschiebung aus dem temporären Uploadverzeichnis in das gewünschte Zielverzeichnis.

Für **Node.js** gibt es das Modul *express-fileupload*. Dieses stattet das *Express*-Webserver-Modul mit zusätzlichen Funktionen für den Datei-**Upload** aus. Außerdem benutzt *express-fileupload* standardmäßig den RAM-Speicher für die temporäre Speicherung der hochgeladenen Dateien. Für die Speicherung der temporären Dateien in einem dafür vorgesehenen Ordner kann man diesen der *Express*-App mitteilen.[Gir20]

Modul express-fileupload: Konfiguration des temporären Ordners über die Express-App:

```
app.use(fileUpload({
    useTempFiles : true,
    tempFileDir : '/tempFolder/'
}));
```

PHP stellt schon alle benötigten Funktionen zur Verfügung. Sowohl *express-fileupload* für Node.js als auch PHP bieten durch die Bereitstellung einer File-Variable die Möglichkeit die hochgeladene Datei vollständig zu validieren. Bei *express-fileupload* kann man über `req.files.file` und bei PHP über `$_FILES['file']` auf diese Variable zugreifen. Die Verschiebung kann über einen einfachen Befehl ausgeführt werden:

Projekt File-Manager (Node.js): Das Verschieben von hochgeladenen Dateien in /main.js (Vereinfacht):

```
// fileid generieren
var fileid = uniqueID();

// Zielpfad fuer die hochgeladene Datei
var uploadfile = "./filestorage/" + fileid;

// Datei in dem/den Ordner filestorage mit der fileid als
    Dateinamen speichern/verschieben
req.files.file.mv(uploadfile);
```

Projekt File-Manager (Apache2/PHP): Das Verschieben von hochgeladenen Dateien in /scripts/up.php (Vereinfacht):

```php
// fileid generieren
$fileid = uniqueID();

// Absoluter Pfad zum Upload-Ordner
$uploaddir = dirname(dirname(__FILE__)) . '/filestorage/';
// Zielpfad fuer die hochgeladene Datei
$uploadfile = $uploaddir . $fileid;

// Datei in dem/den Ordner filestorage mit der fileid als
    Dateinamen speichern/verschieben
move_uploaded_file($_FILES['file']['tmp_name'], $uploadfile);
```

Beim **Deno**-Stack benutzen wir, wie bei Node.js auch, ein weiteres Modul für den Upload. Die `upload()`-Funktion von dem *oak_upload_middleware*-Modul kann direkt einer Route zugeordnet werden und kümmert sich um das komplette Validieren und Verschieben der Datei. Dabei benutzt es die vom Entwickler übergebenen Optionen. Optional kann man auch diese Prozesse selbst implementieren indem man auch wieder die bereitgestellte File-Variable `context.uploadedFiles.file` benutzt.

Projekt File-Manager (Deno): Die Validierung und Verschieben von hochgeladenen Dateien in /main.ts (Vereinfacht):

```ts
router.post(
  "/upload",
  upload(
    'filestorage',
    { extensions: ['txt'], maxFileSizeBytes: 10000000 }
  ),
  async function (context: any, next: any) {
    // ... restlicher Quellcode fuer den Upload (Datenbank-Eintrag
        und Antwort erzeugen)
});
```

Alle Stacks können auch einen Datei-**Download** ausführen, indem sie die Datei serverseitig auslesen, und den Inhalt in den *Body* der Antwort schreiben. **Node.js** hat mit *Express* sogar eine vorgefertigte Funktion für einen Dateidownload. Diese kann einen zweiten Parameter annehmen, der die Datei in den gewünschten Dateinamen umbenennt, ohne die benötigten *Header*-Werte separat setzen zu müssen.

Projekt File-Manager (Node.js): Dateidownload mit express-Funktion in /main.js (Vereinfacht):

```
// Dateidownload-Antwort erstellen
res.status(200).download("./filestorage/" + fileid, newFilename);
```

Vergleichsweise sieht die Umsetzung im **Deno**-Stack folgendermaßen aus:

Projekt File-Manager (Deno): Dateidownload durch separates Auslesen und Header setzen in /main.ts (Vereinfacht):

```
// Datei auslesen
var fileContent = await Deno.readFile(fullFilepath);

// Dateidownload-Antwort erstellen
context.response.status = 200;
context.response.body = fileContent;
context.response.headers.set('Content-Type', 'plain/text');
context.response.headers.set('Content-disposition', 'attachment;
    filename="' + filename + '"');
```

Im **PHP**-Stack müssen die Header immer noch manuell gesetzt werden, aber die gegebene `readfile()`-Methode schreibt den Dateiinhalt direkt in den *Body*:

Projekt File-Manager (Apache2/PHP): Dateidownload durch separates Auslesen und Header setzen in /scripts/down.php (Vereinfacht):

```
// Dateidownload-Antwort erstellen
http_response_code(200);
header("Content-Type: plain/text");
header("Content-Disposition: attachment; filename=$filename");

// Datei auslesen und in den Body schreiben
readfile($file);
```

4.12 Datenbanken

In allen von mir für diesen Vergleich untersuchten Stacks ist es möglich eine Datenbank über die eingebundene *MySQL*-Bibliotheken mit *SQL*-Befehlen anzusprechen. Jede Bibliothek bietet auch die Möglichkeit Zeichenketten zu „escapen“. Somit können sogenannte *SQL-Injection*-Attacken bekämpft werden, indem schädliche oder manipulierte Nutzereingaben durch einen Zwischenschritt entschärft werden. Die eingebaute Funktion maskiert die von den SQL-Befehlen reservierten Sonderzeichen entsprechend, um diese

von den zugelassenen Zeichen unterscheidbar zu machen.

Code-Beispiel aus dem **Node.js**-Stack:

Projekt File-Manager (Node.js-Stack): Datenbankabfrage ausführen in /main.js (Vereinfacht):

```
1 // SELECT-QUERY ausfuehren
2 con.query("SELECT * FROM files WHERE id = " + con.escape(fileid),
      function (err, result, fields) {
3   // Ergebnis ueber den result-Parameter auslesen und verarbeiten
4 });
```

Code-Beispiel aus dem **Deno**-Stack:

Projekt File-Manager (Deno-Stack): Datenbankabfrage ausführen in /main.ts (Vereinfacht):

```
1 // SELECT-QUERY ausfuehren
2 var result = await con.query("SELECT * FROM files WHERE id = ?", [
      fileid]);
3 // Ergebnis ueber die result-Variable auslesen und verarbeiten
```

Code-Beispiel aus dem **Apache2/PHP**-Stack:

Projekt File-Manager (Apache2/PHP-Stack): Datenbankabfrage ausführen in /scripts/down.php (Vereinfacht):

```
1 // SELECT-QUERY ausfuehren
2 $sql = "SELECT * FROM files WHERE id = '" .
      mysqli_real_escape_string($con, $fileid) . "'";
3 $res = mysqli_query($con, $sql);
4 // Ergebnis ueber die $res-Variable auslesen und verarbeiten
```

4.13 Sessions

Eine beliebte Möglichkeit Daten von Nutzern temporär zu speichern sind Cookies. Kleinere Informationen können auf der Clientseite gespeichert werden und später vom Server zum Beispiel bei weiteren Anfragen verarbeitet werden. Cookies sind aber vom Nutzer manipulierbar, da diese schließlich auf dem Endrechner des Nutzers liegen. Die Lösung sind Session-Cookies. Jeder Nutzer bekommt eine Session-ID zugeteilt, welche in einem Cookie auf dem Endgerät gespeichert wird. Diese ID wird dann bei jeder weiteren Anfrage

übermittelt. Dies ermöglicht dem Server den Nutzer mit den auf dem Server gespeicherten Inhalten zu verknüpfen.[17] Eine Session bietet den Vorteil, dass erstens nur der Server die Informationen setzen und bearbeiten kann, und dass die Erlaubnis der Nutzung von Session-Cookies keine Zustimmung des Nutzers benötigt, da diese einen „berechtigten Zweck“ haben, weil sie für die korrekten Ablauf der Webapp benötigt werden können.[Dat21]

Sessions und Session-Cookies werden von allen Technologien unterstützt. Die Nutzerdaten können problemlos über die jeweilige Session-Variable gespeichert und ausgelesen werden.

PHP ist ein Schritt voraus, denn die Session-Variable wird direkt von PHP selbst schon bereitgestellt. Nachdem mit `session_start()` die Session gestartet wurde, bekommt man Zugriff auf die `$_SESSION`-Variable:

Projekt File-Manager (PHP-Stack): Implementierung von Sessions in /functions.php (Ausschnitt):

```php
// Startet eine PHP-Session, sofern noch keine aktiv ist
function startSession() {
    if(session_status() != PHP_SESSION_ACTIVE) session_start();
}

// Gibt den Upload-Verlauf zurueck
function getUploadHistory() {
    startSession();
    if(!isset($_SESSION['uploads'])) return null;
    return $_SESSION['uploads'];
}
```

Bei **Node.js** und **Deno** hat man direkt über die Request- und Context-Variable Zugriff auf die Session, wenn man das passende Modul installiert hat:

Projekt File-Manager (Node.js-Stack): Implementierung von Sessions in /main.js (Vereinfacht):

```javascript
app.get('/getUploadHistory', async function(req, res) {
    // Upload-Verlauf uebermitteln
    if(req.session.uploads) {
        res.status(200).json(req.session.uploads);
    } else {
        // Verlauf ist leer
        res.status(404).json(null);
    }
});
```

[17] Man kann sich das wie „serverseitige Cookies“ vorstellen.

Unter Deno würde man über `context.state.session.set` und `.get` auf die Session zugreifen. Zum Zeitpunkt der Entwicklung von diesem Projekt hatte ich leider nicht die Möglichkeit Sessions im Deno-Stack zu benutzen, wie am anfangs erwähnt. Der Versuch auf die Variablen zuzugreifen hat einen *500 internal server error* verursacht. Da Deno noch im Anfang seiner Entwicklungsphase ist, muss man mit solchen „Überraschungen" rechnen.

4.14 Performance und Skalierbarkeit

Wenn man sich Performance-Unterschiede der Softwarestacks anschaut, wird man merken, dass Node.js und Deno schneller Anfragen abarbeiten und Aufgaben erledigen können als der Apache2/PHP-Stack, da die eventbasierte Architektur hierfür ein Vorteil ist.[emi20; Cho20]

Das heißt allerdings nicht, dass **PHP** in Verbindung mit einem Webserver wie **Apache2** langsam sein muss. Wie die Anwendung abschneiden wird, kommt größtenteils auf die Umsetzung der Entwickler an. Man sollte die Performance nicht durch einen Wechsel auf einen „schnelleren" Softwarestack verbessern, sondern sollte die implementierten Algorithmen und Serverstruktur an sich so anpassen, sodass die verfügbaren Serverressourcen bestmöglich genutzt werden können. Das gilt für alle Softwarestacks.

Auch **Node.js** hat seine Grenzen, beispielsweise das nur mit Aufwand umgehbare RAM-Limit von 1,5 GB, welches von der *V8-JavaScript-Engine* standardmäßig gesetzt wird.[Jua15] So muss man seine Serverstruktur anpassen, dass mehrere Node.js Instanzen parallel laufen können, um mehr Arbeitsspeicher nutzen zu können. Auch die verfügbaren Ressourcen des Servers sind zu beachten. Je nach Anwendung und Anzahl an Anfragen kann Leistung und Performance variieren.

4.15 Automation

Automatisierte Aufgaben auf Webservern können auf verschiedenste Weisen umgesetzt werden. Die bekannten *Cronjobs* unter Linux können zeitlich geplante Befehle ausführen, welche auch wiederkehrend sein können. Eine Anwendung könnte zum Beispiel sein, dass täglich eine Datenbank aufgeräumt, oder dass eine Rundmail verschickt werden soll. Solche Skripte können beispielsweise in *Python* geschrieben werden, müssen sie aber nicht, denn jeder der drei Stacks ermöglicht es serverseitig Skripte einzeln auszuführen.

Bei **Node.js** und **Deno** kann man JavaScript-Skripte über die jeweiligen Start-Befehle ausführen. Doch auch **PHP** kann genutzt werden um PHP-Skript über den `php` -Kommandobefehl einzeln auszuführen. Somit bieten die drei Technologien keine Grenzen, wenn man schon vorhandene Skripte für automatisierte Aufgaben verwenden will. Drittsoftware wie Python ist daher nicht unbedingt für Automationen notwendig.

4.16 Sicherheit

Im Grunde können alle Softwarestacks sicher sein. Letztendlich kommt es auf die Umsetzung und auf die eingebauten Sicherheitsmechanismen an. Unbeabsichtigte Sicherheitslücken können im Quellcode natürlich auftreten.

Deno ist ein Stück sicherer, denn beim Ausführen einer Datei müssen explizit Rechte als Argumente übergeben werden, falls das Skript auf diese zugreifen will. Darunter zählen zum Beispiel Lese- und Schreib-Rechte für das Dateisystem, aber auch Netzwerk-Rechte fallen darunter. Jedoch ist Deno unter diesen Stacks der neueste und hat die höchste Wahrscheinlichkeit unentdeckte Sicherheitslücken zu besitzen.

Falls man seinen Server auch bei der Benutzung von **Node.js** oder **Apache2** schützen möchte, kann man einfach durch separat angelegte Benutzer mit eingeschränkten Berechtigungen den Webserver sichern.

4.17 *Wordpress*, *React* und erweiterte Softwarestacks

Wenn man sich entschieden hat seine Webapplikation mit Softwarelösungen wie *Wordpress* oder *React* umzusetzen, dann wird die Entscheidung, welchen Softwarestack man benutzen soll oder kann, eingeschränkt.

Bei der Entscheidung für ein **PHP**-basierendes Content-Management-System wie *Wordpress*, gibt es keine Möglichkeit an PHP vorbeizukommen, weil *Wordpress* selbst in PHP verfasst wurde. An Apache2 ist man jedoch nicht gebunden, und könnte stattdessen Alternativen wie den *nginx*-Webserver benutzen.

Bei *React* haben wir dagegen mehr Freiheiten. Ein *React*-Frontend kann sowohl über **Node.js** mit *Express*, **Apache2** oder viele andere Webserver bereitgestellt werden. Für die Umsetzung des Backends ist man auch an keinen Softwarestack gebunden.

Falls eine Webapplikation aber mehrere Sprachen oder Technologien benötigt, kann man auch seinen Softwarestack übergreifend erweitern und beispielsweise Node.js/Deno und Apache2 parallel laufen lassen. So könnte man Apache2 als Proxy benutzen, welcher dann Anfragen entweder selbst bearbeitet oder eben an Node.js/Deno weitergibt.[Theb] So ergeben sich mehr Möglichkeiten seine Anwendungen umzusetzen.

Ein weiteres Beispiel für einen zusammengesetzten Softwarestack wäre eine *Wordpress*-Seite, welche auf Apache2 und PHP aufbaut, aber nun auch Echtzeitdaten über WebSockets übermittelt. Hier kann Node.js zusätzlich serverseitig eingesetzt werden. Die Node.js-Instanz würde sich dann um die WebSockets kümmern ohne dass man umständlich WebSockets in PHP implementieren muss. Auch für eine mögliche API-Schnittstelle zum Auslesen der Echtzeitdaten würde sich diese zusätzliche Node.js-Instanz anbieten.

4.18 Entwicklung einer Applikation

Betrachtet man die Entwicklung einer Applikation mit den drei Softwarestacks, gibt es doch schon einige Unterschiede.

Node.js und **Deno** haben hier einen großen Vorteil gegenüber Webservern wie Apache2, denn diese greifen auf keine Konfigurationsdateien außerhalb des einzelnen Projektes zu, wie es bei Apache2 und PHP der Fall ist. Somit ist es einfacher bestehende Node.js und Deno Projekte auf verschiedenen Rechnern und Plattformen aufzusetzen. Es werden auch keine Konfigurationsdateien projektübergreifend verwendet und Applikationen können problemlos gleichzeitig ausgeführt werden. Dieser ausschließlich projektbasierende Aufbau der Applikation macht die Integration eines *Version-Control-System* (kurz VCS) wie GitHub um einiges leichter.

Apache2 und **PHP** speichern ihre Konfigurationsdateien außerhalb des Projektordners beziehungsweise des Webverzeichnisses. Somit muss unter Umständen die ganze Apache2/PHP-Installation angepasst werden, bevor die gewünschte Applikation benutzt werden kann. Das erschwert den Austausch von Projekten zwischen Entwicklern erheblich. Mehrere Apache2/PHP-Applikationen unter einer Apache2-Installation laufen zu lassen kann zu Problemen führen, weil diese dieselbe Apache2-Instanz benutzen, und somit auch dieselben Konfigurationsdateien einlesen, obwohl die Applikationen wahrscheinlich verschiedene Konfigurationen benötigen. Es ist möglich mehrere Apache2-Instanzen/ Konfigurationen aufzusetzen, dies ist aber mit einem großen Mehraufwand verbunden.

Ein anderer Lösungsansatz wäre den Apache2/PHP-Stack in einem *Docker*[18]-Container ausführen zu lassen. Das ermöglicht mehrere Webserver-Instanzen problemlos zu benutzen, indem man für jede Instanz einen *Docker*-Container anlegt.

4.19 Plattformkompatibilität

Grundsätzlich sollte man sich Gedanken über den Server und die Entwicklungsumgebung bei der Umsetzung einer Webapplikation machen. Auch wenn alle Stacks theoretisch auf den bekanntesten Betriebssystemen wie Windows, Linux und macOS laufen und auch in einem *Docker*-Container benutzt werden können, fehlt bei **Deno** doch noch an einigen Stellen die Kompatibilität, Stabilität und Zuverlässigkeit, weil Deno erst Mitte Mai 2020[Dah20a] erschienen ist.

So fehlt Deno beispielsweise die offizielle Unterstützung von ARM-Prozessoren[DeM20] und kann aktuell auf Maschinen wie dem Raspberry Pi noch nicht ausgeführt werden. Für den im November 2020 erschienen Apple M1-Chip[App20], welcher auch ARM-basiert ist, gibt es schon offizielle experimentelle Unterstützung von Deno.[Iwa20] **Node.js** und **Apache2** können beide nativ auf M1 Rechnern installiert und ausgeführt werden, können aber auch noch kleine Probleme verursachen.[Gho20; lvo20]

[18]Mit der Software *Docker* kann man Software, zum Beispiel einen Webserver, in einer isolierten Umgebung, sogenannten *Container*, ausführen. Dabei sind Docker-Container voneinander und von dem *Host*-Rechner abgetrennt.

5 Fazit

Zu welcher Software sollte man für die Umsetzung seiner Webapplikation nun greifen? Zumal es neben den hier genannten noch eine Vielzahl an weiteren Softwarelösungen gibt. Im Grunde kann man erstmal festhalten, dass keine Stacks irgendeine völlige Einschränkung mit sich bringen. Es ist immer möglich ein gewünschtes Feature in einem Stack umzusetzen oder durch weitere Drittsoftware (darunter kann auch ein zweiter Stack zählen) zu verwirklichen. Das kann aber sehr umständlich, kompliziert und eventuell auch schlecht optimiert sein. Wie ich aber auch schon in der Einleitung erwähnt habe, kann man sich viel Aufwand sparen, wenn man sich im Vorhinein Gedanken über den Softwarestack macht.

So lohnt es sich für Echtzeitapplikationen ein Stack mit Node.js zu wählen. Für eine Applikation, welche mehr auf Inhalte wie Text und Bilder basiert, ist einen Stack mit PHP eine gute Wahl. Deno würde ich für eine fertige Produktions-Applikation noch nicht empfehlen, weil Deno noch nicht weit genug ausgereift ist.

Für mein *3D-Koordinatensystem*-Projekt ist Node.js die richtige Wahl, da WebSockets für eine Kernfunktion diese Applikation benötigt werden. Trotzdem könnte man als Abwandlung die statischen Dateien für das Frontend über Apache2, ohne PHP, bereitstellen lassen, um Node.js für die Echtzeitkommunikation zu entlasten.

Für meine Projekte *File-Manager* und *Markdown-CMS* war die Umsetzung im Apache2/PHP am effizientesten. Das liegt daran, dass der Softwarestack schon mit den meisten für die Projekte relevanten Features ausgestattet ist und am wenigsten Software von Dritten benötigt hat. Trotzdem haben alle von mir angewandten Stacks das gleiche Ergebnis liefern können, mit Ausnahme des Deno-Stacks, welcher im *File-Manager* Projekt nicht die Session-Cookies nutzen konnte.

5.1 Highlights

Hier ist noch eine kleine Tabelle der *Hightlights*:

Node.js	**Deno**	**Apache2/PHP**
3D-Koordinatensystem		
+: Schneller Datenaustausch in Echtzeit mit *socket.io* über WebSockets	*nicht umgesetzt*	*nicht umgesetzt*
File-Manager		
+: Einfacher Dateidownload über die `res.download()`-Methode von *Express*	+: Simples wegnehmen von Schreib- und/oder Leseberechtigungen über Startparameter +: Unterstützt direkt TypeScript -: Nicht gelungene Umsetzung von Sessions/Sessioncookies	+: Gut geeignet, um schon fertig generierte Webdokumente zu übermitteln -: Umständliche Umsetzung von semantischen URLs
Markdown-CMS		
-: Komplexe Implementierung von Templating mit *EJS*	*nicht umgesetzt*	+: Bietet die perfekte Umgebung für Templating -: Umständliche Umsetzung von semantischen URLs

+: positiver Aspekt -: negativer Aspekt

6 Schlusswort

Rückblickend kann ich sagen, dass diese besondere Lernleistung eine Herausforderung dargestellt hat, welche ich aber mit viel Motivation und Zuversicht angenommen habe. Während der Zusammenstellung meiner gesammelten Informationen hatte ich nicht nur viel Spaß, sondern konnte auch viele Erfahrungen bezüglich der Dokumentengestaltung mit LaTeX, aber auch der Webentwicklung, mitnehmen.

7 GitHub-Verzeichnis

7.1 Schriftliche Ausarbeitung:

- https://github.com/jgteam/bell--paper

7.2 Projekt *3D-Koordinatensystem*:

- https://github.com/jgteam/bell--3d-coordinate-system--nodejs

7.3 Projekt *File-Manager*:

- https://github.com/jgteam/bell--file-manager--nodejs
- https://github.com/jgteam/bell--file-manager--deno
- https://github.com/jgteam/bell--file-manager--php
- https://github.com/jgteam/bell--file-manager--php--nojs

7.4 Projekt *Markdown-CMS*:

- https://github.com/jgteam/bell--markdown-cms--nodejs
- https://github.com/jgteam/bell--markdown-cms--php

Abbildungsverzeichnis

9 Abbildungen

Abbildung III: Projekt 3D Koordinatensystem: Ausschnitt der Kommunikation über Web-Sockets

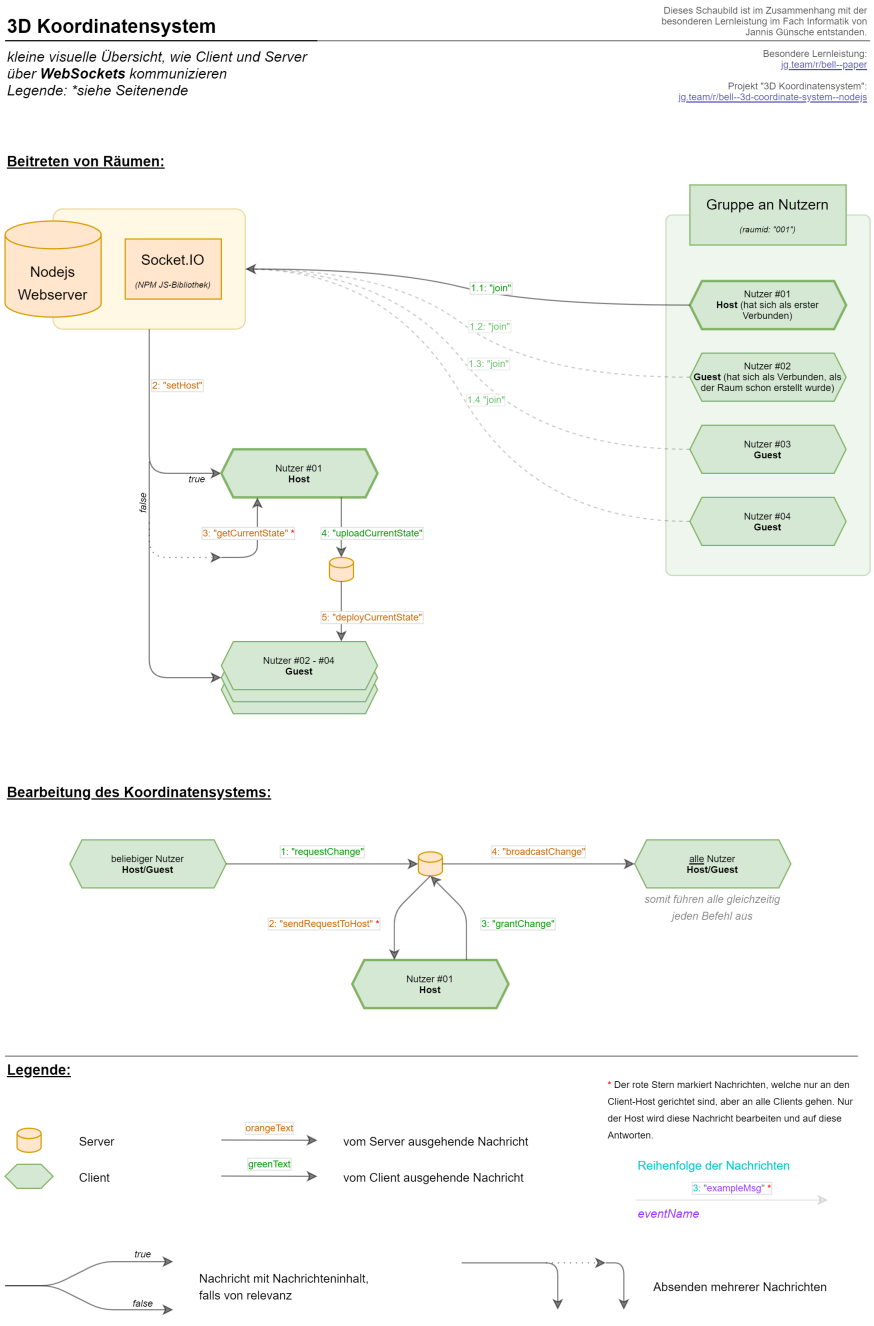


Abbildung IV: Stark vereinfachter Aufbau der Softwarestacks

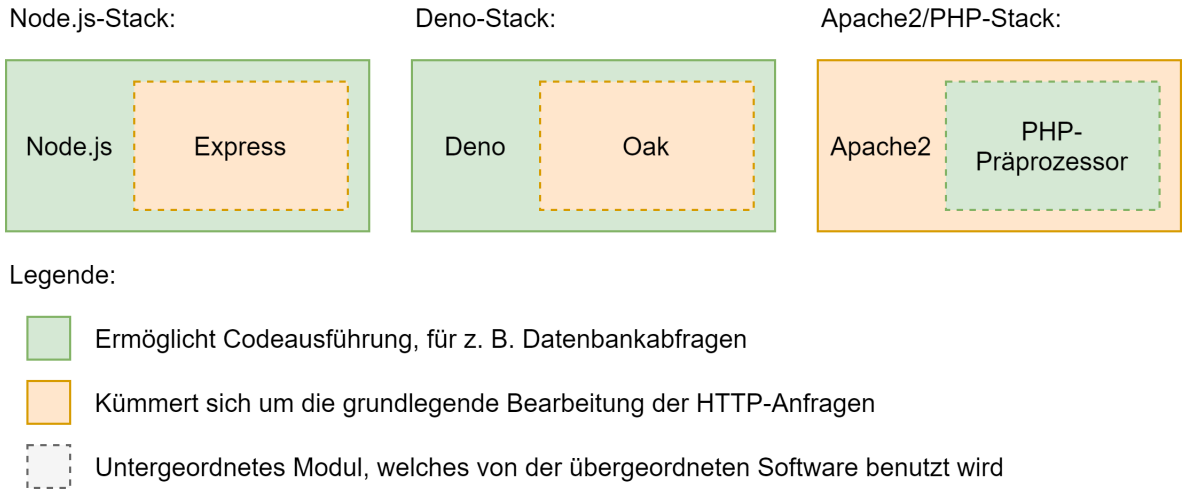
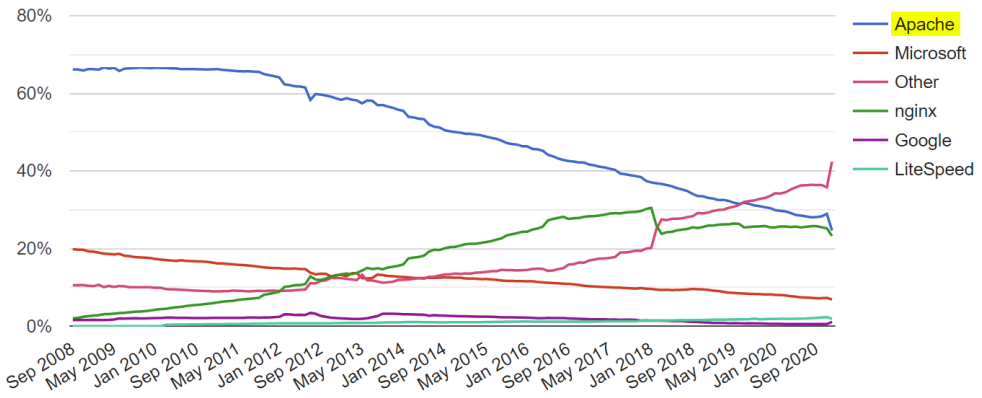


Abbildung V: *Web server developers: Market share of the top million busiest sites* [net20]



Developer	November 2020	Percent	December 2020	Percent	Change
Apache	289,229	28.92%	246,298	24.63%	-4.29
nginx	251,967	25.20%	232,075	23.21%	-1.99
Microsoft	72,656	7.27%	68,507	6.85%	-0.41
LiteSpeed	23,217	2.32%	18,612	1.86%	-0.46

Abbildung VI: Projekt File-Manager (PHP-Stack mit JavaScript): Vor und nach dem Absenden des Formulars, mit aktiviertem JavaScript-Blocker

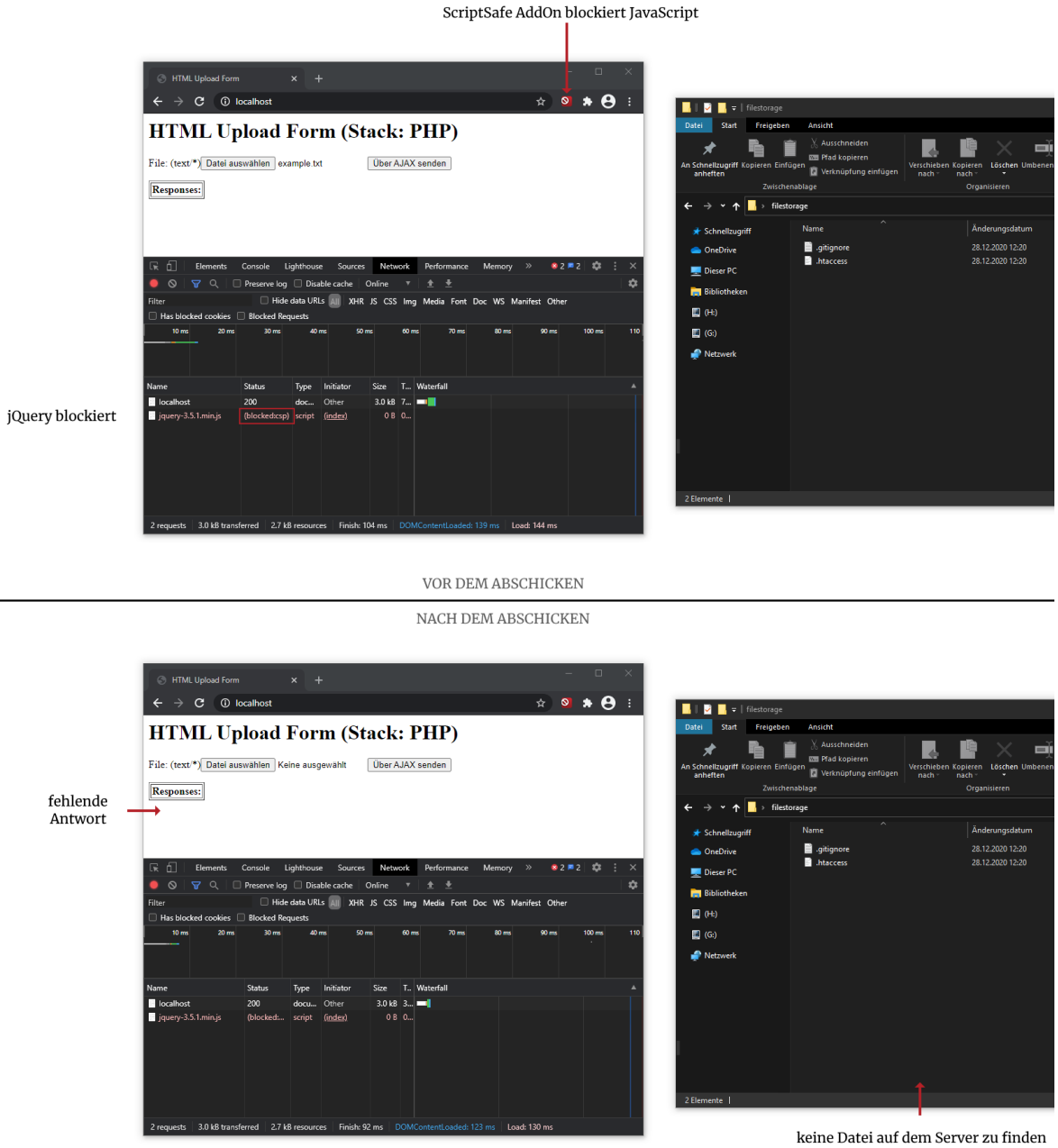


Abbildung VII: Projekt File-Manager (PHP-Stack ohne JavaScript): Vor und nach dem Absenden des Formulars, mit aktiviertem JavaScript-Blocker

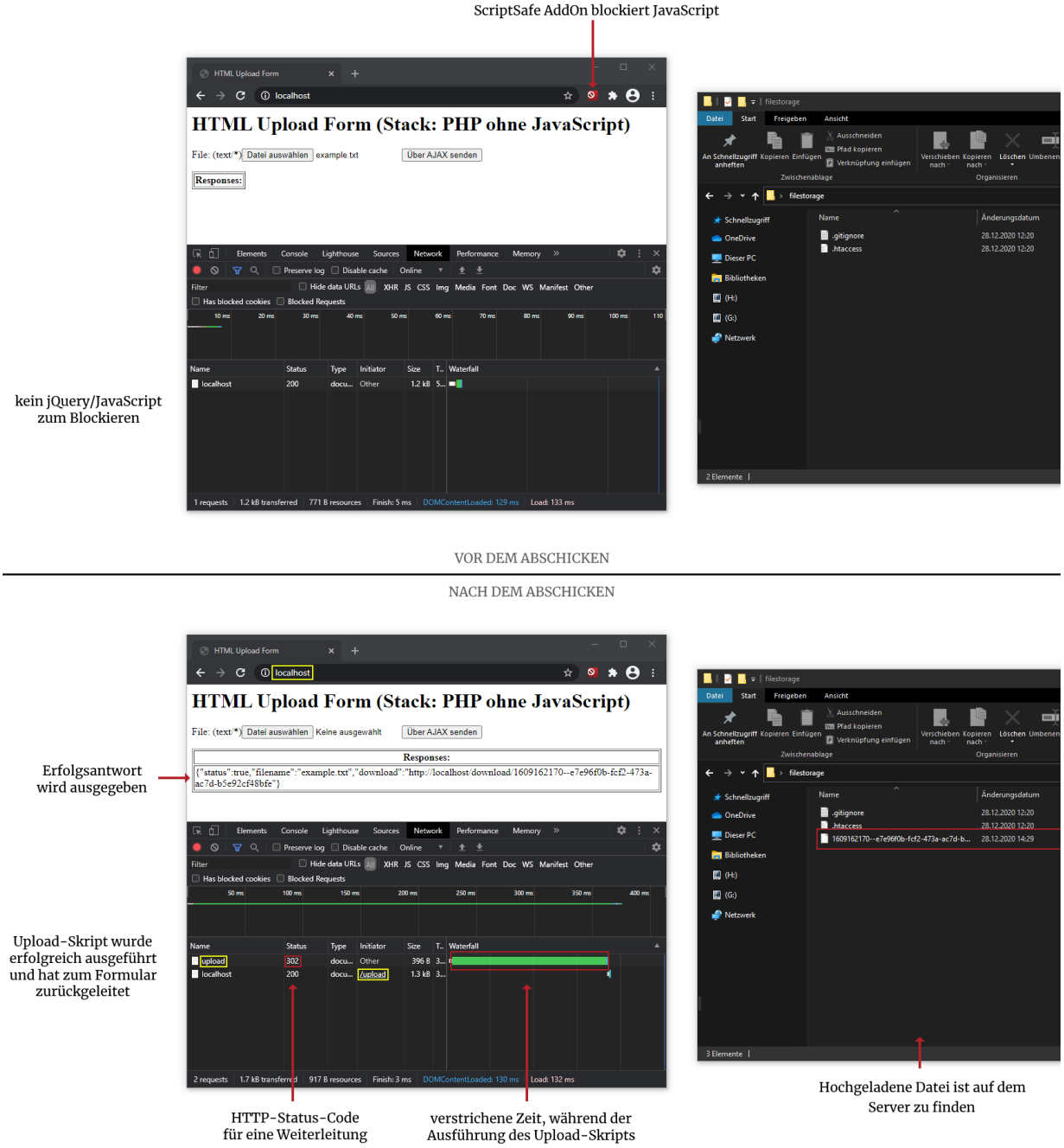


Abbildung VIII: Auflistung aller Abhängigkeiten von dem *Express*-Package

```
`-- express@4.17.1
  +-- accepts@1.3.7
  | +-- mime-types@2.1.27
  | | `-- mime-db@1.44.0
  | `-- negotiator@0.6.2
  +-- array-flatten@1.1.1
  +-- body-parser@1.19.0
  | +-- bytes@3.1.0
  | +-- content-type@1.0.4 deduped
  | +-- debug@2.6.9 deduped
  | +-- depd@1.1.2 deduped
  | +-- http-errors@1.7.2
  | | +-- depd@1.1.2 deduped
  | | +-- inherits@2.0.3
  | | +-- setprototypeof@1.1.1 deduped
  | | +-- statuses@1.5.0 deduped
  | | `-- toidentifier@1.0.0
  | +-- iconv-lite@0.4.24
  | | `-- safer-buffer@2.1.2
  | +-- on-finished@2.3.0 deduped
  | +-- qs@6.7.0 deduped
  | +-- raw-body@2.4.0
  | | +-- bytes@3.1.0 deduped
  | | +-- http-errors@1.7.2 deduped
  | | +-- iconv-lite@0.4.24 deduped
  | | `-- unpipe@1.0.0 deduped
  | `-- type-is@1.6.18 deduped
  +-- content-disposition@0.5.3
  | `-- safe-buffer@5.1.2 deduped
  +-- content-type@1.0.4
  +-- cookie@0.4.0
  +-- cookie-signature@1.0.6
  +-- debug@2.6.9
  | `-- ms@2.0.0
  +-- depd@1.1.2
  +-- encodeurl@1.0.2
  +-- escape-html@1.0.3
  +-- etag@1.8.1
  +-- finalhandler@1.1.2
  | +-- debug@2.6.9 deduped
  | +-- encodeurl@1.0.2 deduped
  | +-- escape-html@1.0.3 deduped
  | +-- on-finished@2.3.0 deduped
  | +-- parseurl@1.3.3 deduped
  | +-- statuses@1.5.0 deduped
  | `-- unpipe@1.0.0
  +-- fresh@0.5.2
  +-- merge-descriptors@1.0.1
  |
  |
  +-- methods@1.1.2
  +-- on-finished@2.3.0
  | `-- ee-first@1.1.1
  +-- parseurl@1.3.3
  +-- path-to-regexp@0.1.7
  +-- proxy-addr@2.0.6
  | +-- forwarded@0.1.2
  | `-- ipaddr.js@1.9.1
  +-- qs@6.7.0
  +-- range-parser@1.2.1
  +-- safe-buffer@5.1.2
  +-- send@0.17.1
  | +-- debug@2.6.9 deduped
  | +-- depd@1.1.2 deduped
  | +-- destroy@1.0.4
  | +-- encodeurl@1.0.2 deduped
  | +-- escape-html@1.0.3 deduped
  | +-- etag@1.8.1 deduped
  | +-- fresh@0.5.2 deduped
  | +-- http-errors@1.7.2 deduped
  | +-- mime@1.6.0
  | +-- ms@2.1.1
  | +-- on-finished@2.3.0 deduped
  | +-- range-parser@1.2.1 deduped
  | `-- statuses@1.5.0 deduped
  +-- serve-static@1.14.1
  | +-- encodeurl@1.0.2 deduped
  | +-- escape-html@1.0.3 deduped
  | +-- parseurl@1.3.3 deduped
  | `-- send@0.17.1 deduped
  +-- setprototypeof@1.1.1
  +-- statuses@1.5.0
  +-- type-is@1.6.18
  | +-- media-typer@0.3.0
  | `-- mime-types@2.1.27 deduped
  +-- utils-merge@1.0.1
  `-- vary@1.1.2
```

Abbildung IX: Projekt File-Manager (Deno-Stack): TypeScript Datentyp Error

Literatur

[App20] Apple, *Apple stellt den M1 vor*, 10. Nov. 2020, URL: https://www.apple.com/de/newsroom/2020/11/apple-unleashes-m1/ (letzter Zugriff am 30. 12. 2020 um 14:17).

[Arc13] Jake Archibald, *JavaScript Promises: An introduction*, 16. Dez. 2013, URL: https://web.dev/promises/ (letzter Zugriff am 21. 11. 2020 um 17:34).

[aro] arocom., *WAS IST NODE.JS?*, URL: https://www.arocom.de/fachbegriffe/webentwicklung/nodejs (letzter Zugriff am 06. 09. 2020 um 15:20).

[Aud18] Michael Auderer, *Basic Express Server in Node.js*, 4. Mai 2018, URL: https://www.digitalocean.com/community/tutorials/nodejs-express-basics (letzter Zugriff am 22. 09. 2020 um 13:12).

[Aur16] Matt Aurand, *Apache Modules Explained*, 18. Apr. 2016, URL: https://www.liquidweb.com/kb/apache-modules-explained/#:~:text=Themodulesallowforapache,removingonesthatarenot (letzter Zugriff am 22. 11. 2020 um 12:31).

[Ban20] Swapnil Banga, *PHP vs Node.js: Differences you need to Know*, 14. Aug. 2020, URL: https://hackr.io/blog/php-vs-node-js (letzter Zugriff am 06. 09. 2020 um 14:35).

[Bor19] Myles Borins, *Modern Asynchronous JavaScript with Async and Await*, 11. Feb. 2019, URL: https://nodejs.dev/learn/modern-asynchronous-javascript-with-async-and-await#why-were-asyncawait-introduced (letzter Zugriff am 21. 11. 2020 um 17:38).

[Cas20] Marcos Casagrande, *Answer to: Where can I see deno downloaded packages?*, 14. Mai 2020, URL: https://stackoverflow.com/a/61799552 (letzter Zugriff am 22. 11. 2020 um 12:21).

[Cho20] Mayank Choubey, *Deno vs Node.js: Performance comparison*, 8. Juni 2020, URL: https://choubey.medium.com/performance-comparison-deno-vs-node-js-part-1-hello-world-3f3b26dd98b9 (letzter Zugriff am 20. 02. 2021 um 19:15).

[chr20a] chrome web store, *NoScript*, 24. Dez. 2020, URL: https://chrome.google.com/webstore/detail/noscript/doojmbjmlfjjnbmnoijecmcbfeoakpjm/ (letzter Zugriff am 24. 12. 2020 um 12:41).

[chr20b] chrome web store, *ScriptBlock*, 24. Dez. 2020, URL: https://chrome.google.com/webstore/detail/scriptblock/hcdjknjpbnhdoabbngpmfekaecnpajba (letzter Zugriff am 24. 12. 2020 um 12:43).

[chr20c] chrome web store, *ScriptSafe*, 24. Dez. 2020, URL: https://chrome.google.com/webstore/detail/scriptsafe/oiigbmnaadbkfbmpbfijlflahbdbdgdf (letzter Zugriff am 24. 12. 2020 um 12:42).

[Cie19] Maciej Cieślar, *What's Deno, and how is it different from Node.js?*, 9. Juli 2019, URL: https://blog.logrocket.com/what-is-deno/ (letzter Zugriff am 23. 12. 2020 um 18:01).

[Dah20a] Ryan Dahl, *Deno 1.0*, 13. Mai 2020, URL: https://deno.land/posts/v1 (letzter Zugriff am 30. 12. 2020 um 13:56).

[Dah20b] Ryan Dahl, *Deno Standard Modules*, 14. Juli 2020, URL: https://deno.land/std (letzter Zugriff am 22. 11. 2020 um 12:26).

[Dat21] Datenschutz.org, *Session Cookie: Was hat es damit auf sich?*, 11. Jan. 2021, URL: https://www.datenschutz.org/session-cookie/ (letzter Zugriff am 02. 02. 2021 um 16:22).

[DeM20] Jack DeMeyers, *Add support for 32-bit ARM processors*, 20. Dez. 2020, URL: https://github.com/denoland/deno/issues/2295 (letzter Zugriff am 30. 12. 2020 um 14:00).

[emi20] emilioSp, *NodeJS vs Apache performance battle*, 26. März 2020, URL: https://dev.to/emiliosp/nodejs-vs-apache-performance-battle-for-the-conquest-of-my-5c4n (letzter Zugriff am 20. 02. 2021 um 19:15).

[Expa] Expressjs.com, *Express: Installation*, URL: https://expressjs.com/de/starter/installing.html (letzter Zugriff am 22. 11. 2020 um 12:20).

[Expb] Expressjs.com, *Statische Dateien in Express bereitstellen*, URL: https://expressjs.com/de/starter/static-files.html (letzter Zugriff am 22. 09. 2020 um 13:35).

[Fet11] Ian Fette, *The WebSocket Protocol*, 1. Dez. 2011, URL: https://tools.ietf.org/html/rfc6455 (letzter Zugriff am 13. 02. 2021 um 16:46).

[Gho20] Ghost, *NodeJS on Apple Silicon M1*, 20. Nov. 2020, URL: https://blockforums.org/topic/453-nodejs-on-apple-silicon-m1 (letzter Zugriff am 30. 12. 2020 um 14:40).

[Gir20] Richard Girges, *express-fileupload readme.md*, 6. Aug. 2020, URL: https://github.com/richardgirges/express-fileupload#using-usetempfile-options (letzter Zugriff am 16. 01. 2021 um 16:02).

[Goo20] Google Search Central, 31. Dez. 2020, URL: https://developers.google.com/search/docs/advanced/guidelines/url-structure (letzter Zugriff am 04. 01. 2021 um 12:41).

[IBM19] IBM Cloud Education, *MEAN Stack*, 9. Mai 2019, URL: https://www.ibm.com/cloud/learn/mean-stack-explained (letzter Zugriff am 11. 10. 2020 um 20:22).

[Iwa20] Bartek Iwańczuk, *Experimental support for Mac Arm64*, 8. Dez. 2020, URL: https://deno.land/posts/v1.6#experimental-support-for-mac-arm64 (letzter Zugriff am 30. 12. 2020 um 14:18).

[jan15] jankal, *Answer to: how are concurrent requests handled in PHP (using - threads, thread pool or child processes)*, 28. Nov. 2015, URL: https://stackoverflow.com/a/33971356 (letzter Zugriff am 21. 11. 2020 um 17:50).

[Jua15] Juan Manuel Osorio Gerdts, *DIE WAHRHEIT ÜBER SKALIERUNG UND LASTVERTEILUNG MIT NODE.JS*, 29. Mai 2015, URL: https://ebusiness2020.wordpress.com/2015/05/29/die-wahrheit-uber-skalierung-und-lastverteilung-mit-node-js/ (letzter Zugriff am 20. 02. 2021 um 19:12).

[Kan20a] Ilya Kantor, *Async/await*, 31. Okt. 2020, URL: https://javascript.info/async-await (letzter Zugriff am 21. 11. 2020 um 17:33).

[Kan20b] Ilya Kantor, *Error handling with promises*, 9. Juli 2020, URL: https://javascript.info/promise-error-handling (letzter Zugriff am 31. 12. 2020 um 15:28).

[lvo20] lvoogdt, *Building the MAMP stack (php, apache & mysql) on Apple Silicon ARM (native)*, 7. Dez. 2020, URL: https://gist.github.com/lvoogdt/538d93ae35297b35e8b56db95b49469c (letzter Zugriff am 30. 12. 2020 um 14:34).

[MDN20] MDN, *JavaScript modules*, 14. Apr. 2020, URL: https://developer.mozilla.org/de/docs/Web/JavaScript/Guide/Modules (letzter Zugriff am 22. 11. 2020 um 12:13).

[Mos11] Amr Mostafa, *Answer to: What is thread safe or non-thread safe in PHP?*, 12. Mai 2011, URL: https://stackoverflow.com/a/5978844 (letzter Zugriff am 21. 11. 2020 um 17:53).

[Mur14] Murali, *Answer to: What is a directive in Apache configuration?*, 26. Apr. 2014, URL: https://stackoverflow.com/a/23314359 (letzter Zugriff am 04. 01. 2021 um 16:11).

[net20] netcraft, *December 2020 Web Server Survey*, 22. Dez. 2020, URL: https://news.netcraft.com/archives/2020/12/22/december-2020-web-server-survey.html (letzter Zugriff am 25. 12. 2020 um 12:12).

[NPM] NPM Docs, *About semantic versioning*, URL: https://docs.npmjs.com/about-semantic-versioning (letzter Zugriff am 22. 12. 2020 um 12:13).

[Nym07] Robert Nyman, *The Importance Of A Semantic URL*, 16. März 2007, URL: https://robertnyman.com/2007/03/16/the-importance-of-a-semantic-url/ (letzter Zugriff am 04. 01. 2021 um 12:01).

[Ofo18] Victor Ofoegbu, *The only NodeJs introduction you'll ever need.* 23. Jan. 2018, URL: https://codeburst.io/the-only-nodejs-introduction-youll-ever-need-d969a47ef219 (letzter Zugriff am 22. 12. 2020 um 11:37).

[Rai19] Shane Rainville, *Installing PHP Module for Apache on Ubuntu*, 17. Dez. 2019, URL: https://www.serverlab.ca/tutorials/linux/web-servers-linux/installing-php-for-apache-on-ubuntu/ (letzter Zugriff am 22. 11. 2020 um 12:41).

[Rea] ReactPHP, *ReactPHP*, URL: https://reactphp.org (letzter Zugriff am 13. 02. 2021 um 16:41).

[rip] riptutorial.com, *What is Type Juggling?*, URL: https://riptutorial.com/php/example/9267/what-is-type-juggling- (letzter Zugriff am 01. 01. 2021 um 12:50).

[Rob14] Philip Roberts, *Was zum Henker ist überhaupt die Ereignisschleife? | Philip Roberts | JSConf EU*, 9. Okt. 2014, URL: https://www.youtube.com/watch?v=8aGhZQkoFbQ (letzter Zugriff am 06. 09. 2020 um 15:01).

[Roo15] Robert Roose, *LAMP vs MEAN, Deciding the right stack for your startup*, 30. Nov. 2015, URL: https://www.linkedin.com/pulse/lamp-vs-mean-deciding-right-stack-your-startup-robert-roose/ (letzter Zugriff am 11. 10. 2020 um 20:17).

[Roo20] Brent Roose, *Asynchronous and parallel PHP*, 12. Okt. 2020, URL: https://github.com/spatie/async (letzter Zugriff am 21. 11. 2020 um 17:46).

[Row14] Walker Rowe, *Nginx vs Apache*, 14. Mai 2014, URL: https://anturis.com/blog/nginx-vs-apache/ (letzter Zugriff am 21. 11. 2020 um 17:51).

[Sai20] Mohamed Said, *Asynchronous PHP*, 12. Feb. 2020, URL: https://divinglaravel.com/asynchronous-php (letzter Zugriff am 21. 11. 2020 um 17:43).

[Sch16] Isaac Schlueter, *kik, left-pad, and npm*, 23. März 2016, URL: https://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm (letzter Zugriff am 09. 10. 2020 um 19:35).

[The] The PHP Group, *PHP beziehen*, URL: https://www.php.net/manual/de/faq.obtaining.php (letzter Zugriff am 21. 11. 2020 um 17:54).

[Thea] The Apache Software Foundation, *Modul-Index*, URL: https://httpd.apache.org/docs/2.4/de/mod/ (letzter Zugriff am 22. 11. 2020 um 12:35).

[Theb] The Apache Software Foundation, *Reverse Proxy Guide*, URL: https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html (letzter Zugriff am 24. 01. 2021 um 12:35).

[TM19] Liran Tal und Simon Maple, *npm passes the 1 millionth package milestone! What can we learn?*, 4. Juni 2019, URL: https://snyk.io/blog/npm-passes-the-1-millionth-package-milestone-what-can-we-learn/ (letzter Zugriff am 12. 10. 2020 um 20:59).

[Tor] Christoph Tornau, *REST-SCHNITTSTELLEN*, URL: http://www.fit4php.net/funktionsbibliothek/xml-und-json/rest-schnittstellen/ (letzter Zugriff am 30. 10. 2020 um 17:01).

[Tor20] Ricardo Torrão, *DENO VS NODE - COMPARISON (2020)*, 8. Okt. 2020, URL: https://www.imaginarycloud.com/blog/deno-vs-node/ (letzter Zugriff am 21. 11. 2020 um 17:55).

[tuta] tutorialspoint, *ExpressJS - RESTFul APIs*, URL: https://www.tutorialspoint.com/expressjs/expressjs_restful_apis.htm#:~:text=AnAPIisalwaysneeded,REST(RepresentationalTransferState). (letzter Zugriff am 30. 10. 2020 um 17:05).

[tutb] tutorialspoint, *Node.js - Express Framework*, URL: https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm (letzter Zugriff am 06. 09. 2020 um 15:22).

[Upw15] Upwork, *What is a software stack?*, 27. Mai 2015, URL: https://www.upwork.com/resources/what-is-a-software-stack (letzter Zugriff am 06. 09. 2020 um 14:46).

[Wal16] Nathan Wall, *Answer to: async/await implicitly returns promise?*, 6. Feb. 2016, URL: https://stackoverflow.com/a/35302535 (letzter Zugriff am 31. 12. 2020 um 16:15).

[Wik20a] Wikipedia, *List of Apache modules*, 19. Sep. 2020, URL: https://en.wikipedia.org/wiki/List_of_Apache_modules (letzter Zugriff am 22. 11. 2020 um 12:38).

[Wik20b] Wikipedia, *MEAN (solution stack)*, 28. Juni 2020, URL: https://en.wikipedia.org/wiki/MEAN_(solution_stack) (letzter Zugriff am 03. 10. 2020 um 11:12).

[Wik20c] Wikipedia, *Softwarestack*, 14. Juni 2020, URL: https://de.wikipedia.org/wiki/Softwarestack (letzter Zugriff am 06. 09. 2020 um 14:37).

[Zap18] Zaphoid, *Answer to: What is deduped in npm packages list?*, 17. Okt. 2018, URL: https://stackoverflow.com/a/52860479 (letzter Zugriff am 09. 10. 2020 um 19:52).