

# **webmap - Automatische Generierung von Websites zur Interaktiven Datenveranschaulichung**

## **DIPLOMARBEIT**

verfasst im Rahmen der

**Reife- und Diplomprüfung**

an der

**Höhere Lehranstalt für Informationstechnologie,  
Ausbildungsschwerpunkt Medientechnik**

Eingereicht von:

Jonas Dorfinger  
Sebastian scholl

Betreuer:

Dietmar Steiner

Projektpartner:

Christopher Stelzmüller, triply GmbH

Leonding, April 2022

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, April 2022

J. Dorfinger & S. Scholl

# Abstract

Triply GmbH develops high-quality software that helps understanding existing mobility situations and providing safe and sustainable mobility solutions. One core function is the visualisation of the results of their analysis. It is important to demonstrate those to clients already during the sales process. A lot of complex data is produced during those audits. Webmap should generate an appealing and interactive website on the basis of that data. This web-based tool should be provided to triply employees.



# Zusammenfassung

Triply GmbH entwickelt hochwertige Softwarelösungen, die dabei helfen, bestehende Mobilitäts-situationen zu verstehen und sinnvolle, sichere und nachhaltige Mobilität bereitzustellen. Eine der Kernfunktionen liegt darin Ergebnisse der Analysen einfach und verständlich darzustel- len. Bereits in den Verkaufsprozessen ist es wichtig, den Kundinnen und Kunden dies klarzumachen und aufzuzeigen. Bei diesen Analysen entstehen riesige und komplexe Datensätze. Mithilfe von webmap sollen aus diesen Daten automatisch ansprechende und interaktive Websites generiert werden. Das Tool soll webbasiert für die Mitarbeiterinnen und Mitarbeiter von triply zur Verfügung gestellt werden.



# **1 Danksagungen**

Unser Dank gilt Herrn Prof. Dietmar Steiner für die Betreuung während der gesamten Arbeit. Besonders bedanken möchten wir uns weiters bei der Firma triply GmbH, allen voran bei Christopher Stelzmüller und Sebastian Tanzer, für die Möglichkeit, die Diplomarbeit in Zusammenarbeit mit ihrem Start-up zu schreiben.

Vielen Dank auch an alle Kolleginnen und Kollegen bei triply, insbesondere an Amine Ahnine, für die Unterstützung bei der Implementierung.

Außerdem möchten wir uns bei allen bedanken, die sich für das Korrekturlesen dieser Arbeit Zeit genommen haben.

# Inhaltsverzeichnis

|   |            |
|---|------------|
| <b>1 Danksagungen</b>                             | <b>III</b> |
| <b>2 Einleitung</b>                               | <b>1</b>   |
| 2.1 Kurzbeschreibung . . . . .                    | 1          |
| 2.2 Aufgabenstellung . . . . .                    | 1          |
| 2.3 Zielsetzung . . . . .                         | 1          |
| 2.4 Geplantes Ergebnis . . . . .                  | 1          |
| <b>3 Technologien</b>                             | <b>2</b>   |
| 3.1 Versionierung . . . . .                       | 2          |
| 3.2 Projektkoordination . . . . .                 | 5          |
| 3.3 Frontend Framework/Library . . . . .          | 7          |
| 3.4 Map Frameworks . . . . .                      | 19         |
| 3.5 Static Site Generators . . . . .              | 24         |
| 3.6 Deployment Pipeline . . . . .                 | 25         |
| 3.7 Formate für die Konfigurationsdatei . . . . . | 29         |
| 3.8 Backend . . . . .                             | 33         |
| 3.9 nginx . . . . .                               | 34         |
| 3.10 Containerization . . . . .                   | 35         |
| <b>4 Umsetzung</b>                                | <b>40</b>  |
| 4.1 webmap Workflow . . . . .                     | 40         |
| 4.2 Frontend Implementierung . . . . .            | 41         |
| 4.3 Generator Implementierung . . . . .           | 50         |
| 4.4 Angular Projekt . . . . .                     | 51         |
| 4.5 Backend Implementierung . . . . .             | 54         |
| 4.6 Deployment Pipeline . . . . .                 | 57         |

|  |             |
|--|-------------|
| <b>5 Evaluation des Projektverlaufs</b>            | <b>60</b>   |
| 5.1 Meilensteine . . . . .                         | 60          |
| 5.2 Kommunikation . . . . .                        | 60          |
| 5.3 Probleme während der Implementierung . . . . . | 60          |
| <b>Abbildungsverzeichnis</b>                       | <b>VII</b>  |
| <b>Tabellenverzeichnis</b>                         | <b>VIII</b> |
| <b>Quellcodeverzeichnis</b>                        | <b>IX</b>   |
| <b>Anhang</b>                                      | <b>X</b>    |

## **2 Einleitung**

### **2.1 Kurzbeschreibung**

Triply entwickelt hochwertige Softwarelösungen, die dabei helfen, bestehende Mobilitäts-situationen (Verkehrsanalysen, Besucherströme) zu verstehen. Eine der Kernfunktionen liegt darin, Ergebnisse der Analysen einfach und verständlich darzustellen. Dafür soll eine Software geschaffen werden, die interaktive Kunden-Demos einfach und schnell erzeugen kann.

### **2.2 Aufgabenstellung**

Bei dem Diplomarbeitsprojekt webmap handelt es sich um die Implementierung eines Generators, welcher interaktive Webseiten automatisch erstellt. Für die erwartete Benutzung ist es erforderlich, dass die ganze Software vollständig im Webbrower und somit plattformunabhängig funktionieren wird.

### **2.3 Zielsetzung**

Mitarbeiterinnen und Mitarbeitern von triply soll es mit der Datenvisualisierungs-Pipeline möglich sein, schnell und einfach interaktive Websites auf der Basis von komplexen Datensätzen zu erstellen, um diese potentiellen Kundinnen und Kunden anschaulich zu präsentieren.

### **2.4 Geplantes Ergebnis**

Entwicklung eines effizienten Generators für interaktive Datenveranschaulichung. Weiters ist eines der Ziele, die Benutzung der Software so leicht und intuitiv wie möglich zu gestalten. Zusätzlich dazu soll der gesamte Prozess vom Starten der Konfiguration bis zur laufenden Website möglichst wenig Zeit in Anspruch nehmen.

# **3 Technologien**

## **3.1 Versionierung**

Der Begriff Versionierung beschreibt im Allgemeinen einen Prozess, welcher zur Dokumentation von Änderungen an Dokumenten oder Dateien stattfindet. Nach jeder Änderung wird im System eine aktuelle Version abgespeichert und mit einer eindeutigen Versionsnummer versehen. Diese wird meist automatisch durch das Versionierungstool vergeben. Zusätzlich wird auch gespeichert, wer welche Einträge wann und wo gemacht hat, so werden Änderungen transparent mitprotokolliert. Eine der wichtigsten Vorteile einer Versionierung ist jedoch die Funktion, dass man alte Stände von Dateien wiederherstellen kann. Eine Versionsverwaltungssoftware kann auch die gleichzeitigen Zugriffe auf eine Datei von mehreren Personen koordinieren. Versionierung ist nicht nur Standard in der Softwareentwicklungs Branche, sondern auch in etlichen weiteren Bereichen stark verbreitet.

### **3.1.1 automatische vs. manuelle Versionierung**

Wenn während dem Schreiben einer Arbeit Sicherheitskopien angelegt und diese mit zum Beispiel "arbeit-neu", "arbeit-fertig" oder "arbeit-fertig2" benannt werden, dann ist das nichts anderes als eine manuelle Versionierung, weil auch hier eine eindeutige Versionsnummer für jede Datei vergeben wurde. Das birgt aber auch große Risiken, da eine manuelle Versionierung viel Zeit und Disziplin erfordert und immer die Gefahr besteht, dass man eine wichtige Version überschreibt. Die meisten Probleme werden von der automatischen Versionierung gelöst. Bei dieser Variante werden automatisch, wenn man einen Stand als "fertig" markiert, diese mit einer Versionsnummer versehen und im Archiv schreibgeschützt abgelegt. Dadurch hat man die Risiken der manuellen Versionierung auf ein Minimum reduziert [?].

### 3.1.2 git

git ist *das* Versionierungssystem in der Softwareentwicklung. Es wurde designt um kleine aber auch extrem große und komplexe Projekte effizient zu verwalten. Grundsätzlich als CLI (Command Line Interface) konzeptioniert, gibt es mittlerweile auch etliche Grafische Implementierungen, um git interaktiver und intuitiver zu gestalten. Laut einer Umfrage von Stackoverflow nutzen mehr als 87% der Entwickler weltweit git [?].

#### Der git Workflow

Der git Workflow beschreibt die effizienteste Arbeitsweise mit git. Dabei gibt es vier Stufen zwischen denen mit verschiedenen Konsolen Befehlen gewechselt werden kann.

**1. Remote Repository** Das Remote Repository befindet sich auf einem externen Server (zum Beispiel Github Server 3.1.4). Mit einem einfache *git clone* Befehl in der Konsole, kann man das Repository klonen und somit ein lokales Repository erstellen. Auf dieses Archiv können mehrere Personen zugreifen und so gemeinsam an einem Projekt arbeiten.

**2. Lokales Repository** Das Lokale Repository ist eine Kopie der Version im Remote Repository, mit dem Unterschied, dass die lokalen Änderungen in diesem Repository gesammelt werden und erst durch den Befehl *git push* wieder zurück ins Remote Repository gelangen.

**3. Staging Area** In der Staging Area sind alle lokal modifizierten Files, welche für den nächsten Commit auserwählt sind, gesammelt. Durch den Befehl *git commit -m <message>* gelangen ausgewählte Änderungen in das lokale Repository. Ein Commit repräsentiert dabei eine Version des Programmes, welche mit einer eindeutigen Nummer im System gespeichert wird.

**4. Working Directory** Im Working Directory wird tatsächlich programmiert, alle Änderungen der Files werden hier durchgeführt. Mit dem Befehl *git add <filename>* können einzelne Files gezielt in die Staging Area verschoben werden um diese dann anschließend ins lokale Repository zu committen.

[?]

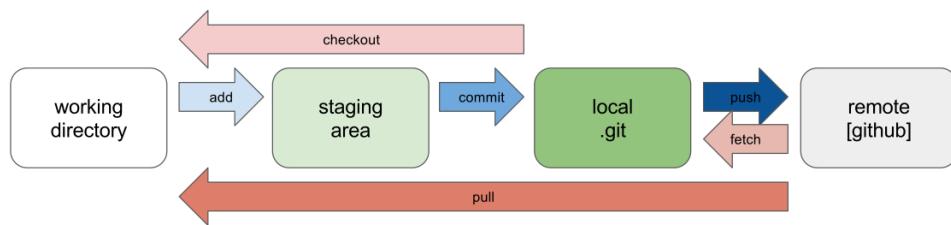


Abbildung 1: git workflow [?]

### 3.1.3 Semantische Versionierung

Eine semantische Versionsnummer besteht aus drei Teilen:

MAJOR.MINOR.PATCH

Jeder Teil soll nach Änderungen in der entsprechenden Kategorie in Einserschritten erhöht werden.

**MAJOR** Wenn API Änderungen durchgeführt werden, welche die API mit der Vorgängerversion inkompatibel machen

**MINOR** Wenn Funktionalität hinzugefügt wird, welche mit älteren Versionen noch kompatibel sind

**PATCH** Wenn kleine Bug fixes durchgeführt werden, welche auch mit älteren Versionen kompatibel sind

Zusätzlich können auch labels für *pre-release* oder *build metadata* angegeben werden:

MAJOR.MINOR.PATCH format

[?]

### 3.1.4 GitHub

GitHub Inc. ist ein amerikanisches gewinnorientiertes Softwareunternehmen, welches git Repositories in der Cloud anbietet, also das Remote Repository. Dadurch können Privatpersonen aber auch große Unternehmen und Konzerne ganz leicht git als Versionierungstool verwenden. Durch viele Features von GitHub wird das Arbeiten mit git so stark erleichtert, dass auch Anfänger bereits sehr professionell mit git arbeiten und

lernen können. Normalerweise verwendet man git mit der Kommandozeile des jeweiligen Betriebssystems, was hohes technisches Verständnis voraussetzt. Man kann einen kostenlosen Account erstellen und gratis Repositories anlegen und verwenden, deshalb ist Github in der Open-Source Community sehr stark verbreitet. Doch mittlerweile kann diese Software mehr als "nur" Projekte versionieren. Es gibt project-tracking-tools (GitHub Projects, Issues, Milestones, Labels, etc.), Pipelines für Continues Integration and Delivery, gratis Website Hosting und noch vieles mehr [?].

Zusätzlich gibt es aber auch andere Programme welche zu GitHub gehören. Zum einen ist es *Atom*, ein kostenloser und Open-Source Editor für Entwickler, weiters gibt es noch *Electron*, ebenfalls ein Open-Source JavaScript Framework, es wird verwendet um Web-Anwendungen zu Desktopanwendungen zu machen.

Unternehmen können kostenpflichtige Organisationen anlegen, um alle Repositories einer Firma gesammelt an einem Ort zu verwalten. Dazu gibt es wieder etliche Features, welche die Sicherheit, Rollen und vieles mehr betreffen.

Seit 2018 gehört GitHub zu Microsoft, welche für die Übernahme \$7,5 Milliarden US-Dollar bezahlt haben [?].

## 3.2 Projektkoordination

Triply verfügt über eine GitHub Organisation, daher ist es naheliegend, dass für webmap ein eigenes Repository angelegt wird.

Um Übersicht über den Projektverlauf zu behalten, ist die Entscheidung aufgrund von internen Vereinbarungen von triply sowie die Erfahrung des Teams auf GitHub Projects gefallen. Man kann ein Project Board ganz einfach über die GitHub Website erstellen und dabei auswählen, ob eine Vorlage ausgewählt werden soll.

Es gibt folgende Templates zur Auswahl: [?]

**None** Es wird ein völlig leeres Project Board erstellt, alle Spalten und Einstellungen müssen manuell gemacht werden.

**Basic Kanban** Ein Project Board mit den Spalten *To do*, *In progress* und *Done* wird erstellt.

**Automated Kanban** Es wird ein Basic Kanban Board erstellt, mit der zusätzlichen Funktion von eingebauten Triggern, welche automatisch Issues und Pull Requests zwischen den Spalten hin und her schieben.

**Automated Kanban with Reviews** Das Template *Automated Kanban with Reviews* erweitert das Automated Kanban Template um 2 weitere Spalten und einen Trigger , für die Pull Request Reviews.

**Bug triage** Dieses Template wird verwendet, um Bugs zu priorisieren und diese geordnet anzusehen, dabei gibt es folgende Spalten:

- To do
- High priority
- Low priority
- Closed

### 3.2.1 Issues

Issues haben die Möglichkeit, die anstehende Arbeit zu dokumentieren und planen.

[?]

### 3.2.2 Automatisierung

Durch die Verwendung spezieller Keywords in Commit messages oder in einem Kommentar bei einem Issue, kann der Zustand dieses Issues verändert werden. Dies geht mit der Automatisierung von Project Boards einher.

#### Keywords:

- close
- closes
- closed
- fix
- fixes

- fixed
- resolve
- resolves
- resolved

## Verwendung

<Keyword> <Issue Nummer> <Commit Message>

[?]

## 3.3 Frontend Framework/Library

Webmap ist als Webanwendung konzeptioniert, deshalb ist es für das Frontend wichtig, eine Technologie zu wählen, welche nicht nur langlebig ist, sondern auch eine gute Übersicht bietet. Das beinhaltet unter anderem, dass eine große Community hinter dem Framework oder der Library steht, aber auch die Anforderungen des Auftraggebers dürfen nicht unbeachtet bleiben. Anhand dieser definierten Kriterien wurden die in 2021 beliebtesten Frameworks beziehungsweise Libraries verglichen.

### 3.3.1 Framework vs. Library

Sowohl Frameworks als auch Libraries sind Codes, welche oft auftretende Probleme einfach lösen sollen. Um das konkret zu veranschaulichen, kann man dies mit einer Metapher besser beschreiben.

Eine Library ist wie der Besuch in einem Möbelhaus, man hat zwar grundsätzlich ein Haus aber keine Einrichtung, man braucht also eine leichte Lösung um das Haus zu möblieren. Einen Tisch selbst zu zimmern wäre zu aufwendig, deswegen wählt man einen Tisch aus der großen Auswahl in dem besuchten Geschäft.

Das Verwenden eines Frameworks ist wie der Bau eines Hauses, man hat ein paar Blaupausen und Vorgaben zum Design und zur Architektur des neuen Gebäudes. Es wird also ein Grundgerüst zur Verfügung gestellt, in welchem man den Regeln konform frei agieren kann.

Zusammengefasst sind Frameworks Gerüste, welche durch Entwickler projektspezifisch erweitert werden müssen und Libraries fertige Pakete, mit denen Programmierer Anwendungen um spezielle Funktionen erweitern.

### 3.3.2 Angular

Angular wurde 2016 von Google ins Leben gerufen, seitdem ist es ein beliebtes Framework für die Webentwicklung. Nur auf TypeScript basierend füllt es die Lücke zwischen der wachsenden Nachfrage von Technologien und traditionellen Ideen zur Performance Optimierung. Die Entwicklerplattform bietet etliche Services und Features an, dazu gehören nicht nur der Cross-Platform Support, sondern auch eine built-in Geschwindigkeits- und Performanceoptimierung. Auch für Unternehmen mit sehr hohen Nutzerzahlen ist Angular ein gern gewähltes Tool. Ein großer Vorteil von Angular ist zudem, dass es unter einer Open-Source-Lizenz veröffentlicht ist und somit nicht von dem Angular-Google Team, sondern auch von der Community betreut und erweitert wird. Angular hat mit dem Two-Way-Binding einen großen Vorteil gegenüber React 3.3.3. Dieses Feature stellt sicher, dass die View und das Model in Echtzeit miteinander synchronisiert werden. Das bedeutet, dass eine Änderung im Model direkt in der View angezeigt wird, aber auch eine Änderung in der View direkt ins Model übertragen wird. [?, ?]

#### Vorteile

- Features und Funktionalität wie das Two-Way-Binding sind direkt inkludiert
- Direkt eingebaute Features zum Updaten der View oder des Models
- Komfortable Wiederverwendung von Components durch Dependency Injection
- Große Community zum Lernen und zur Fehlersuche

#### Nachteile

- Schwieriger zu lernen, da Angular sehr umfangreich ist und es viele mögliche Lösungen gibt
- Durch die riesigen und komplexen Strukturen kann es sein, dass bei großen dynamischen Applikationen Performance Probleme auftreten
- Angular ist ein full-package Framework, weshalb es für kleine Applikationen ungeeignet ist

### 3.3.3 React

ReactJS ist die größte Konkurrenz von Angular, mit einem Vielfachen an Marktanteilen. Im Gegensatz zu Angular wird React von Facebook gewartet und betreut und verfügt über die Funktionalität des virtuellen Document Object Models (DOM). Das virtuelle DOM verbessert die Performance und verringert die Dauer von Render Prozessen, da nur die Inhalte verwendet werden, welche sich auch tatsächlich verändert haben und nicht die gesamte View neu gerendert wird. Durch dieses Feature wird eine Plattform geschaffen, welche sich sehr gut für Applikationen mit großem Traffic eignet. Ein Alleinstellungsmerkmal ist dazu auch noch, dass React Server-Side Rendering unterstützt. Für SPAs, also Single Page Applications ist es empfehlenswert sich für React zu entscheiden, durch das Wiederverwenden der Komponenten kann man in kurzer Zeit gute und interaktive User Interfaces entwickeln.

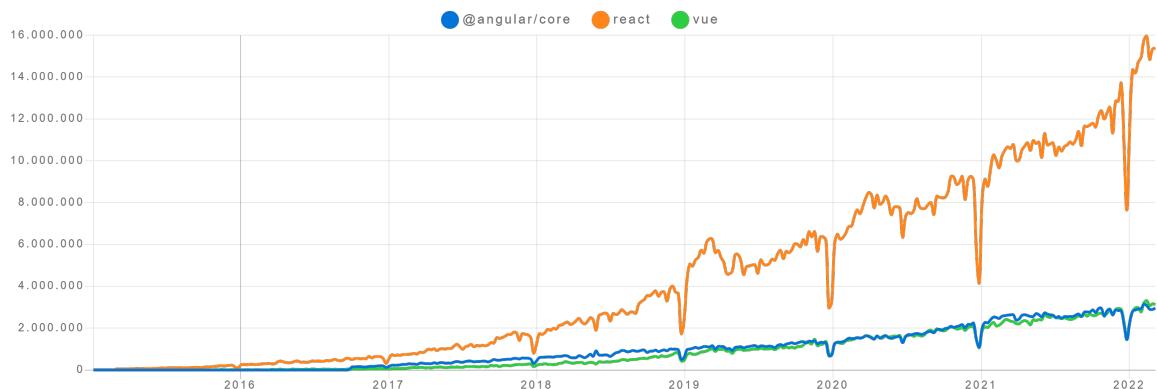


Abbildung 2: wöchentliche Downloads [?]

|   |               | Stars | Issues | Version | Updated | Created | Size        |              |                        |
|---|---------------|-------|--------|---------|---------|---------|-------------|--------------|------------------------|
| ■ | @angular/core | npm   | GitHub | -       | -       | 13.2.6  | 7 days ago  | 6 years ago  | minzipped size 73.3 KB |
| ■ | react         | npm   | GitHub | -       | -       | 17.0.2  | a year ago  | 10 years ago | minzipped size 2.8 KB  |
| ■ | vue           | npm   | GitHub | 28.174  | 564     | 3.2.31  | a month ago | 8 years ago  | minzipped size 33.7 KB |

Abbildung 3: Direkter Vergleich der Frameworks [?]

## Vorteile

- Durch das virtuelle DOM ist die Performance sehr optimiert und kann gut mit vielen gleichzeitigen Zugriffen zureckkommen
- Unabhängige Components sind leicht zu implementieren und wiederzuverwenden
- React Developer Tools sind mit einer großen Nutzbarkeit sehr ausgereift

## Nachteile

- Durch etliche Updates ist die Dokumentation lückenhaft
- Schnelle Änderungen in kurzer Zeit, neu gelerntes ist in naher Zukunft bereits wieder "alt"
- Inhalte wie JSX sind komplex und schwer zu lernen
- ReactJS setzt einiges an Erfahrung in JavaScript oder TypeScript voraus

### 3.3.4 Vue

Vue.js ist ein kleines, aber sehr einfach funktionierendes Frontend System und wird immer beliebter. Es ist gut sehr darin, Probleme mit denen Angular Developer umgehen müssen zu vermindern oder gar zu lösen. Vue.JS ist außerdem sehr leichtgewichtig und klein, hat aber trotzdem viele nützliche Features wie das Two-Way-Binding out-of-the-box dabei, auch die Verwendung als PWA (Progressive Web Apps) ist möglich. Der wohl größte Nachteil ist die niedrige Trucknumber [?, ?]. Bei Projekten beschreibt dies, wie viele Personen ausfallen können, damit das Projekt weiterlaufen kann. Je höher die Trucknumber, desto mehr Personen könnten ausfallen und das Projekt würde trotzdem weiterlaufen. Im Gegensatz dazu, je kleiner die Trucknumber, desto weniger Personen können ausfallen. Das geht so weit, dass das gesamte Projekt an einer einzigen Person hängt. Das letzte Beispiel ist fast bei Vue.js gegeben, in das Core Repository von Vue committed regelmäßig nur der Gründer von Vue, Evan You. VueJS kann in beiden, TypeScript und JavaScript geschrieben werden.

## Vorteile

- Sehr leicht zu lernen, vor allem mit basis JavaScript Wissen
- Flexibles App Layout
- Umfangreiche und ausführliche Dokumentation

## Nachteile

- Sehr kleine Community (Wartung und Support)
- Sehr geringe Truck Number
- Stabilitätsprobleme bei großen Projekten

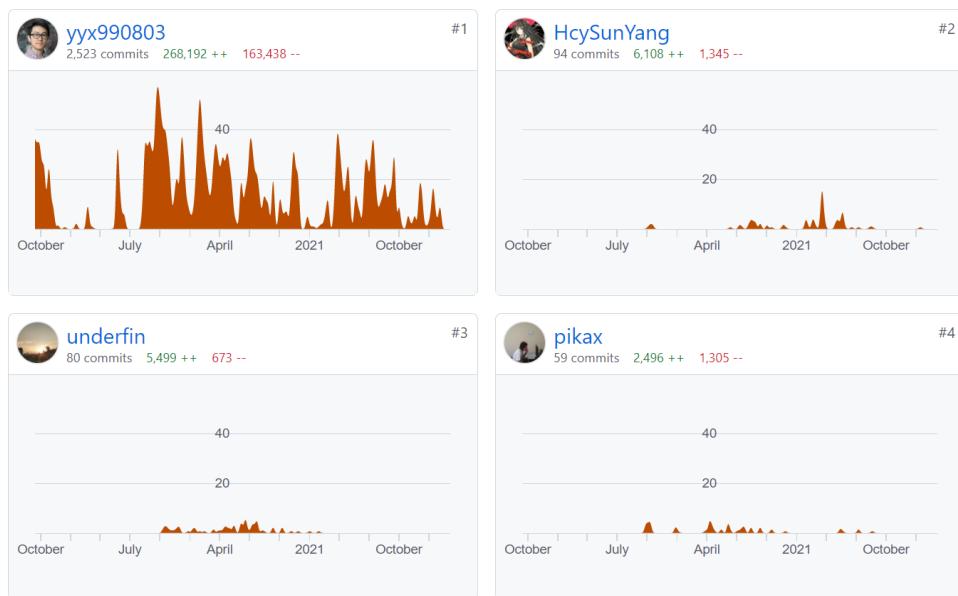


Abbildung 4: Vue.js Core Repository Commits im Zeitraum 16. September 2018 - 9. Februar 2022 [?]

[?, ?, ?]

### 3.3.5 Verwendung von Angular

#### Warum Angular?

Einer der Entwickler hat bereits mehr als zweieinhalb Jahre Erfahrung in der Entwicklung von UserInterfaces und Web-Oberflächen mit Angular. Zusätzlich dazu wird Angular in der HTL Leonding ab dem 4. Jahrgang in der Abteilung Medientechnik unterrichtet, wodurch die Wahl auf dieses Framework naheliegend war. Unabhängig von der Schule und den Entwicklern ist es außerdem eine Anforderung der Betreuerfirma triply Angular zu verwenden. Triply vertraute bisher bei allen Frontend Projekten auf Angular.

#### Angular installieren und verwenden

**Node.js** Node.js ist eine asynchrone open-source Event gesteuerte JavaScript Entwicklerumgebung. Node.js ist Plattformunabhängig, das heißt, dass in node geschriebene Anwendungen sowohl auf Windows oder Linux als auch auf macOS problemlos funktionieren. JavaScript wird grundsätzlich im Browser ausgeführt, durch node kann JavaScript jedoch auch serverseitig ausgeführt werden. Dabei erhält man durch die Node.js API umfangreiche Funktionen, um mit dem lokalen Betriebssystem zu interagieren. Node.js wird verwendet, um skalierbare Netzwerk Applikationen zu designen. Im folgenden

Beispiel wird gezeigt, wie man einen einfachen Node.js Webserver implementiert. Durch diese Implementierung können etliche Verbindungen gleichzeitig behandelt werden. Bei jeder Verbindung wird die Callback Funktion ausgeführt. Werden keine Verbindungen aufgebaut, so geht Node.js in den Wartezustand. Wenn man Node.js verwendet, muss man sich keine Sorgen über dead-locks machen, weil es durch die Asynchronität keine locks gibt. Node.js Funktionen führen kaum I/O Prozesse durch, dadurch kann ein Prozess nie blockieren außer es wird die I/O Funktion synchron über die Node.js Library aufgerufen. Weil keine Prozesse blockieren, ist es naheliegend skalierende Systeme in Node.js zu entwickeln [?].

```

1  const http = require('http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${
14   hostname}:${port}/`);
15 });

```

**npm** npm oder auch ausgeschrieben der Node-Package-Manager ist eine Software um Packages zu Node.js Applikationen hinzuzufügen. npm wird standardmäßig mit Node.js mitinstalliert, es handelt sich um eine CLI (Command-Line-Interface) welche auf die online registry zugreift. Die npm-registry ist eine Online-Datenbank welche public (open-source) und private (auch kostenpflichtige) Packages zur Verfügung stellt. Die Packages werden über die CLI installiert und können über die npm Website gesucht werden. npm ist ein Produkt von Github, Github wiederum gehört seit 2018 zu Microsoft [?].

**TypeScript** TypeScript ist eine typisierte Programmiersprache welche auf JavaScript aufbaut. TypeScript ist außerdem eine Übermenge von JavaScript. Das bedeutet, dass TypeScript alle Eigenschaften von JavaScript hat und zusätzlich noch eigene Funktionalität mitbringt. Folgende Features werden im Gegensatz zu JavaScript unterstützt:

- Typisierung von Variablen (auch während des Kompilierens)
- Type inference
- Type erasure
- Interfaces
- Enumerated types

- Generics
- Namespaces
- Tuples
- Async/await
- Classes
- Modules
- Verkürzte Syntax von anonymen arrow-functions
- Optionale Parameter and Defaultwerte für Parameter

Wenn man ein TypeScript Programm ausführt, dann wird der Quellcode von TypeScript Code in JavaScript Code transpiled. Dies wird erzielt, indem eine von mehreren möglichen Optionen ausgewählt wird. Die Auswahl beschränkt sich dennoch auf den TypeScript Checker oder Babel, beide Varianten sind stark verbreitet. TypeScript kann sowohl clientseitig, als auch serverseitig (zum Beispiel in Kombination mit Node.js) eingesetzt werden.

### **Beispiel: Funktionen mit und ohne Types**

JavaScript Funktion zum Addieren von 2 Zahlen (ohne types)

```

1  function add(number1, number2) {
2      return number1 + number2;
3  }
4
5
6  add(6, 9); // returns 15
7
8  add("Hallo ", "Welt"); // returns "Hallo Welt"

```

Bei JavaScript gibts es keine typisierten Funktionen, deshalb verhält sich die Funktion immer den Parametern entsprechend.

TypeScript Funktion zum Addieren von 2 Zahlen (mit types)

```

1  function add(number1: number, number2: number): number {
2      return number1 + number2;
3  }
4
5  add(6, 9); // returns 15
6
7  add("Hallo ", "Welt"); // Error: Argument of type 'string' is not assignable to
                           parameter of type 'number'.

```

Da TypeScript Typisierungen unterstützt, wird bereits zur Compilezeit sichergestellt, dass die richtigen Daten als Parameter übergeben werden. Deshalb tritt beim zweiten Aufruf ein Error auf, weil ein String nicht als Nummer verwendbar ist.

[?, ?]

### Wie funktioniert Angular?

*Ivy* ist die neue Compilation und Rendering Pipeline für Angular. Seit der Angular Version 9 wird standardmäßig die neue Pipeline verwendet und ersetzt damit die Alter Pipeline, welche unter dem Namen *View Engine* bekannt ist.

Da *Ivy* nicht nur eine Rendering Engine, sondern auch eine Compile Engine ist, öffnen sich ganz neue Türen was die Optimierung für und über das gesamte Framework angeht.

Anstatt Template Daten zu generieren, diese in einen Interpreter zu geben, der wiederum entscheidet, welche Operationen ausgeführt werden, wird direkt eine Sammlung an *Template Instructions* generiert. Die *Template Instructions* beinhalten die Logik, welche Components instanziert, DOM Nodes erstellt und Change Detection in Echtzeit ausführt.



Abbildung 5: Ivy Pipeline

**Incremental DOM** Jede Component wird in eine eigene Sammlung an Instructions compiled, diese erstellen DOM Trees und aktualisieren die Daten an Ort und Stelle, wenn diese sich ändern. Zum Erstellen des *Document Object Models* gibt es drei verschieden Funktionen:

- *elementStart* (opening tag)
- *text* (inner html)
- *elementEnd* (closing tag)

Beim Aktualisieren des DOMs wird nicht automatisch jedes Mal ein komplett neuer Tree erstellt, sondern es werden alle Nodes verglichen und nur jene verändert, welche auch tatsächlich verändert wurden. Das spart nicht nur Arbeitsspeicher, sondern auch Zeit.

Der Grund warum Incremental DOM verwendet sind die Ziele welche erreicht werden möchten. Der Fokus liegt auf einer besseren Performance von Web-Applikationen auf Mobilen Endgeräten (Handys, Tablets). Um dies zu erreichen, ist es erforderlich zwei Dinge zu optimieren, *bundle size* und *memory footprint*.

**Shakable Tree** Bei Verwendung der Incremental DOM Strategie wird nicht die Component interpretiert, sondern jede Component verweist auf verschiedene Instructions. Wenn kein Verweis auf eine Instruction zu finden ist, dann wird dieser Teil nicht verwendet und ist somit überflüssig. Durch die Vorteile von Incremental DOM ist dies bereits zur Compile Zeit bekannt, weshalb alle nicht benützten Instructions vom bundle entfernt, was wiederum die *bundle size* beträchtlich verringern kann.

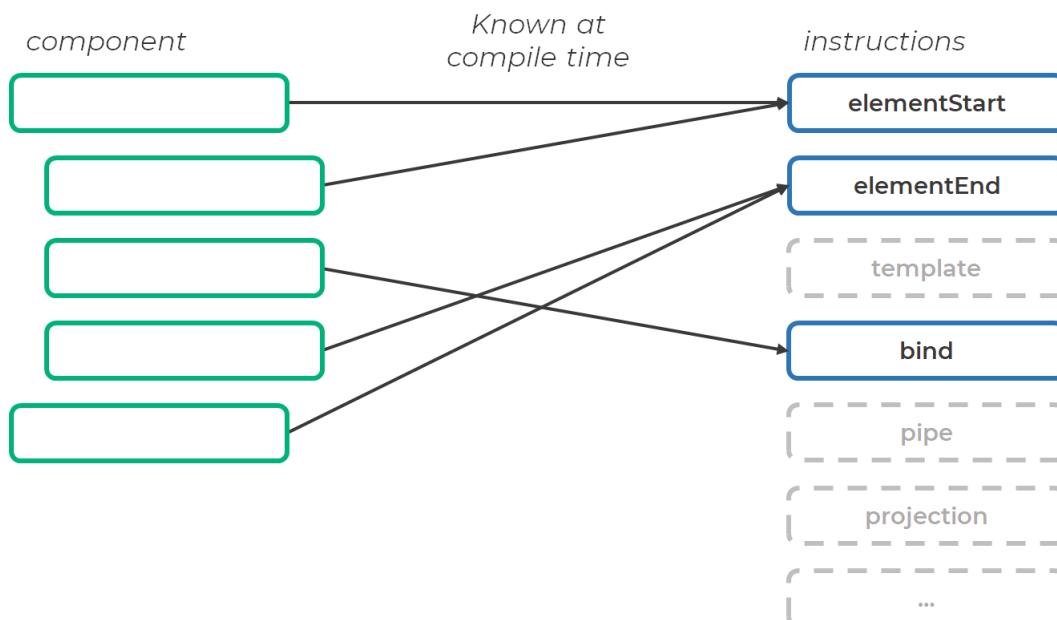


Abbildung 6: Skakable Tree

[?, ?]

## Angular Material

Angular Material ist eine Component-Library von Angular, diese beinhaltet stylistisch aufeinander abgestimmte Elemente. Diese Komponenten können ganz einfach importiert werden. Durch das Responsive Design der Inhalte, eignen sich diese sehr gut für die Umsetzung einer Web-Oberfläche, da nicht viel Zeit für das Design verloren geht, sondern der Fokus voll und ganz auf die Funktionalität gelegt werden kann.

Angular Material kann jederzeit über einen Konsolenbefehl zu einem Angular Projekt hinzugefügt werden:

```
ng add @angular/material
```

Bei der Installation kann man bereits vordefinierte Themes von Angular auswählen, zur Auswahl stehen insgesamt 4 verschiedene, zwei Dark- und zwei Lighttheme Varianten [?]:

- Deep Purple & Amber (Light)
- Indigo & Pink (Light)
- Pink & Blue-grey (Dark)
- Purple & Green (Dark)

Weiters hat man die Möglichkeit eigene Themes zu erstellen. Mithilfe von Custom Themes können die von Angular bereitgestellten Komponenten für jedes Projekt individuell eingefärbt werden. Um das zu erreichen, wählt man beim Installieren die fünfte Auswahlmöglichkeit *Custom* aus. Dies erstellt den dafür benötigten Code in dem *styles.scss* File. Man kann sich über diverse Internet Seiten Farbpaletten generieren lassen, diese werden benötigt, um ein eigenes Theme zu gestalten. Ein Angular Theme hat genau "drei" verschiedenen Farben, wobei es eher Farbpaletten sind.

**primary-palette** Die Primary Palette legt die Hauptfarbe fest, welche in fast allen Komponenten standardmäßig verwendet wird. Man kann allerdings auch bei den meisten Komponenten individuell als Parameter angeben, welche der drei Paletten verwendet werden soll.

**accent-palette** Die Accent Palette definiert eine Akzentfarbe, welche einen hohen Kontrast zur Hauptfarbe haben soll, um bestimmte Elemente besser hervorheben zu können.

**warn-palette** Die Warn Palette besteht meist aus einem Rot, welcher für Fehlermeldungen verwendet wird.

Eine Farbpalette besteht aus einer Basisfarbe und mehreren Abstufungen dieser Farbe, um auch hier unterschiedliche Töne für Hervorhebungen zu haben.

Ein Beispiel einer Farbpalette für eine primary-palette:

```

1  $primary-palette: (
2    50 : #e2f3f4,
3    100 : #b8e2e4,
4    200 : #88ced2,
5    300 : #58babf,
6    400 : #35acb2,
7    500 : #119da4,
8    600 : #0f959c,
9    700 : #0c8b92,
10   800 : #0a8189,
11   900 : #056f78,
12   A100 : #a7f7ff,
13   A200 : #74f3ff,
14   A400 : #41efff,
15   A700 : #27ecff,
16   contrast: (
17     50 : #000000,
18     100 : #000000,
19     200 : #000000,
20     300 : #000000,
21     400 : #000000,
22     500 : #ffffff,
23     600 : #ffffff,
24     700 : #ffffff,
25     800 : #ffffff,
26     900 : #ffffff,
27     A100 : #000000,
28     A200 : #000000,
29     A400 : #000000,
30     A700 : #000000,
31   )
32 );

```

Insgesamt hat man die Auswahl aus 36 verschiedenen Komponenten [?, ?, ?].

Die Komponenten werden ganz einfach über einen Custom HTML Tag eingebunden.

## triply Material Library

Triply hat in den letzten Monaten einige eigene Angular Komponenten erstellt und in einer internen Bibliothek gesammelt. Auf diese Inhalte wurde besonders zurückgegriffen, um zum Beispiel einen Slider oder einen true/false Switch zu verwenden. Der Unterschied zu Angular Material liegt darin, dass die Firmeninterne Library aus Files besteht, welche

in das Projekt kopiert werden, vereinfach gesagt, dass dabei Custom-Components verwendet werden.



Abbildung 7: Triply Slider und Triply Switch

### Model-View-Controller Design Pattern

Dieses Pattern wird oft für die Entwicklung von User Interfaces verwendet, dabei wird die Logik in drei verschiedenen Elementen (Files) aufgeteilt. Das hat zum Ziel, interne Abbildungen und Referenzen von Daten beziehungsweise Informationen in einer Art und Weise aufzuteilen. Das Model ist von der restlichen Anwendung komplett unabhängig. Im Gegensatz dazu ist die View stark abhängig von der zu programmierenden Anwendung. Die Aufgabe der View beschränkt sich rein auf die Visualisierung der durch den Controller bereitgestellten Daten. Der Controller hat eine Vermittlerrolle zwischen Model und View. In Angular ergeben alle drei Files eine gemeinsame Component.

**Model** Das Model File ist das Herzstück der Component, es beinhaltet alle Daten Strukturen und den Aufbau, dabei werden Daten, Logik und Regeln der Component festgelegt und verwaltet. Bei Angular sind das entweder Interfaces oder Klassen ohne Logik. [?]

**View** Die View beinhaltet alle Details zur Darstellung der Daten und Informationen. Es kann mehrere Views für die gleiche Information geben, zum Beispiel ein Balkendiagramm für den Manager oder eine Tabelle für die Buchhaltung.

**Controller** Der Controller ist das Gehirn einer Component, er ist das Verbindungsstück zwischen View und Model. Dabei werden die Daten mittels Getter-Funktionen aus dem Model geladen und so verändert beziehungsweise manipuliert, dass die View diese richtig anzeigen kann. Wenn dadurch Änderungen der Daten gemacht werden, speichert der Controller diese Änderung mittels Setter-Funktionen im Model.

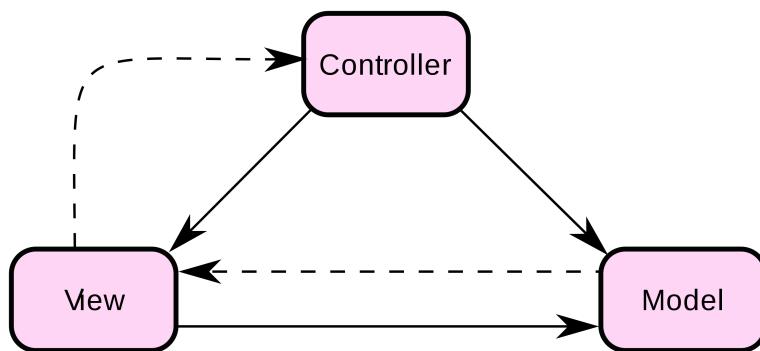


Abbildung 8: Model-View-Controller-Pattern

## 3.4 Map Frameworks

Geo-Daten sind sehr komplexe Daten, welche nur sehr schwer und mit viel Erfahrung ohne Visualisierung interpretiert werden können. Um solche Daten visualisieren zu können werden Map Frameworks benötigt. Diese können auf digitalen Karten die Koordinaten grafisch darstellen, jedoch sind auch interaktive Darstellungen möglich. Drei der bekanntesten Frameworks sind:

- Mapbox
- Leaflet
- Google Maps

### 3.4.1 Mapbox

Mapbox ist ein amerikanisches Unternehmen, welches sich auf Custom-Maps spezialisiert und die Nische stark vergrößert hat. Dabei geht es um die Darstellung von Geo-Daten auf digitalen Kartensystemen und deren Individualisierung. Weiters ist Mapbox Contributor zum OpenStreetMap (OSM) Projekt, das ist eine Geo-Datenbank mit Kartendaten und Informationen für den gesamten Globus. Große Konzerne wie Amazon, Facebook oder Snapchat setzen auf OpenStreetMap, um ihre eigenen Angebote und Services den Kundinnen und Kunden anzubieten. Unternehmen wie Tesla, mit einer eigenen Navigationssoftware, verwenden die OSM-Datenbank ebenfalls für ihre eigenen Zwecke [?, ?].

#### Vorteile

- OpenSource (bis Version 2.0.0) [?]

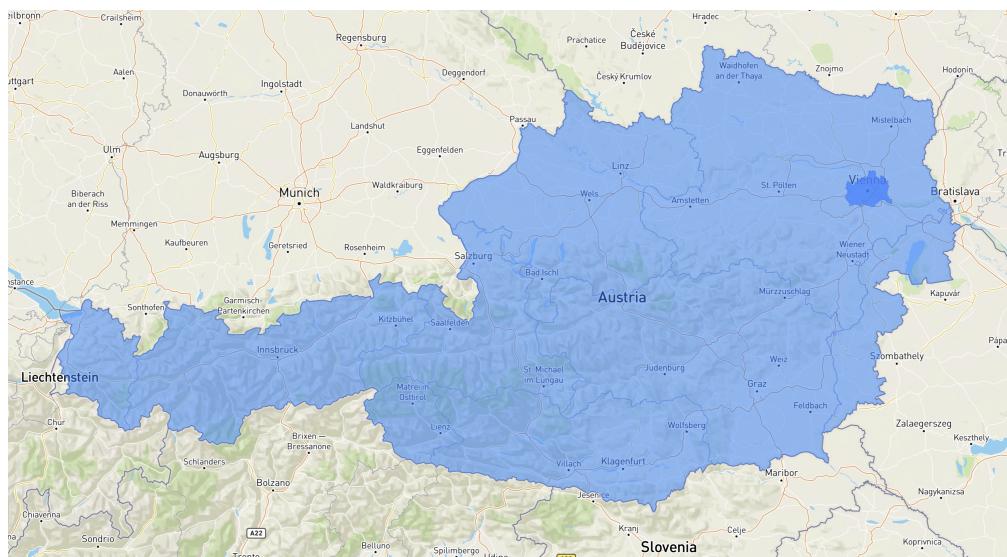


Abbildung 9: Österreich mit Bundesländern in Mapbox

- Gutes Handling von großen Datensätzen
- Individualisierbar
- Gute Unterstützung von komplexen Strukturen

### Nachteile

- Ausbaufähige Dokumentation
- Kein ausgereifter Angular Wrapper

### 3.4.2 Leaflet

Leaflet ist das laut eigenen Angaben führende Open-Source JavaScript Library, wenn mobile und interaktive Karten gefordert sind. Durch die sehr gute API Dokumentation können ganz leicht Geo-Daten auf Karten dargestellt und interaktive Elemente eingebaut werden. Der Fokus liegt dabei auf den Grundfunktionalitäten, das beinhaltet auch eine Performance Optimierung für fast alle Geräte und Plattformen. Dank einer leichten Anbindungsmöglichkeit für Plugins, gibt es etliche davon im Internet, die man einfach verwenden kann, um die Funktionalitäten zu erweitern. Leaflet wird auch von den ganz großen Anbietern verwendet: GitHub, Pinterest, Facebook, European Commission, The Washington Post und noch viele mehr. [?]

### Vorteile

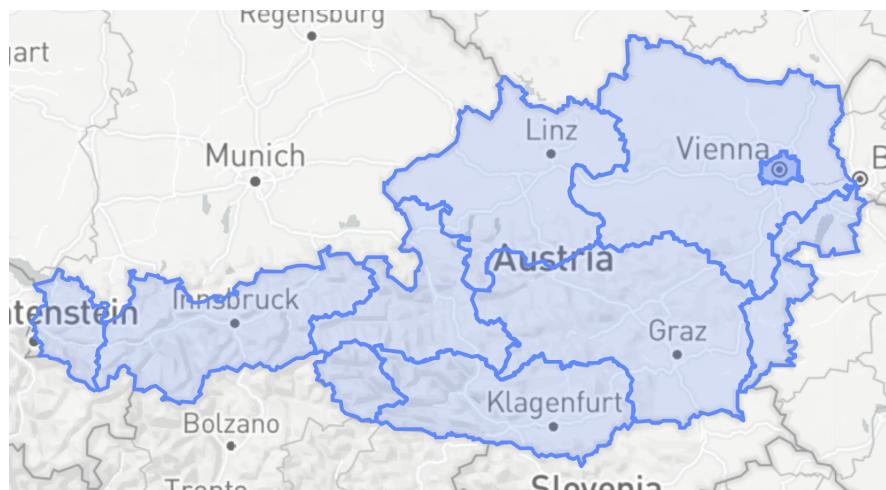


Abbildung 10: Österreich mit Bundesländern in Leaflet

- Open-Source (keine Restriktion oder Terms of Service)
- Hervorragende Dokumentation
- Leichtgewichtig (39KB)
- Plugin Support
- Angular Wrapper
- Bereits von triply in Verwendung

### Nachteile

- Offizielle Dokumentation beinhaltet nur grundlegende Beispiele
- Möglicherweise ist das Verwenden von GIS Programmen (zum Beispiel QGIS) notwendig, um Informationen aufzubereiten

### 3.4.3 Google Maps

Google Maps ist der Kartendienst von Google und der Kartenprovider mit den meisten Daten, welche zur Verfügung stehen. OpenStreetMap ist vergleichsweise auf dem Globus nicht überall verfügbar, jeder kann aber an der Vergrößerung der Daten oder Informationen mitwirken. Google Maps hingegen hat Daten bis in jede noch so kleine Straße, wodurch eine sehr gute Informationslage ermöglicht wird. In Kombination mit Places API und der Maps images API kann niemand mit Google mithalten. Das liegt auch daran, dass Google mehr als Maps betreut und deshalb viel mehr Möglichkeiten als zum Beispiel OpenStreetMap hat. Der wohl größte Nachteil liegt an der kostenpflichtigen

Nutzung ab einer gewissen Nutzerzahl. Ab 1000 Anfragen pro Monat ist Google Maps nicht mehr gratis und daher frei zu nutzen [?].

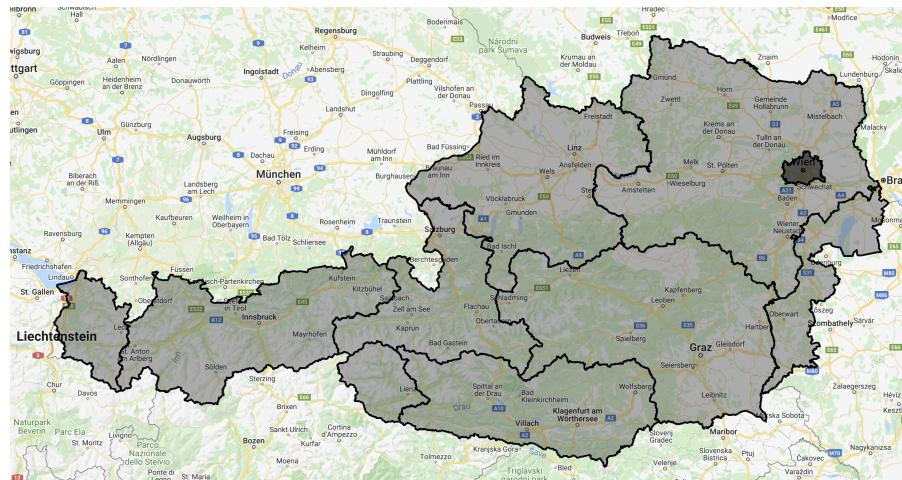


Abbildung 11: Österreich mit Bundesländern in Google Maps

## Vorteile

- Google Ökosystem
- hohe Abdeckung des Globus
- Angular Wrapper

## Nachteile

- Kann kostenpflichtig werden
- Terms of Use und Restriktionen
- Google Logo immer sichtbar
- Farbschema kann kaum verändert werden.

### 3.4.4 Performance im Vergleich

Um eine Wahl auf ein bestimmtes Framework treffen zu können, wurden mithilfe der Chrome-Developer-Tools die Ladezeiten unter bestimmten Voraussetzungen der einzelnen Frameworks getestet. In diesen Tools gibt es die Möglichkeit zwischen einem schnellem (fast), einem mittel-schnellem (mid-tier) und einem langsamen (low-end) Gerät zu wählen. Dadurch änderten sich die Ladezeiten teils sehr drastisch. Folgende Ergebnisse wurden bei dem Versuch erhalten:

|                        | Leaflet | Mapbox | Google Maps |
|------------------------|---------|--------|-------------|
| <b>low-end device</b>  | 132s    | 138s   | 192s        |
| <b>mid-tier device</b> | 39,5s   | 41,69s | 56,12s      |
| <b>fast device</b>     | 1,2s    | 3,9s   | 3,23s       |

Aufgrund der oben genannten Vor- und Nachteilen, sowie persönlichen Erfahrungen der Entwickler und nach Absprache mit der Betreuerfirma ist die Wahl schlussendlich auf Leaflet in Kombination mit einer Mapbox Tile-layer gefallen. Tile-layers bestimmen die Darstellung der Karte an sich und werden von externen Servern direkt für den aktuell zu sehenden Bereich angefordert.

## 3.5 Static Site Generators

Um die Website zu generieren wurden verschiedene Static Site Generators analysiert. Dabei wurde darauf geachtet, ob diese sich eignen, aufgrund von Geodaten eine Seite zusammenzubauen.

### 3.5.1 Next.js

Next.js ist eines der beliebtesten Frameworks, um statische Websites zu generieren. Die Templates für diese Seiten werden jedoch in React geschrieben. Vorteile von Next.js sind die einfache Installation und die Flexibilität des Frameworks [?].

### 3.5.2 Jekyll

Jekyll ist ein Static Site Generator, der in Ruby geschrieben ist [?]. Im Gegensatz zu Next.js nutzt Jekyll kein JavaScript Framework, um die Templates zu erstellen, sondern setzt auf *Liquid* [?]. Es besteht zwar die Möglichkeit, es mit Angular zu verknüpfen. Da aber Liquid und Angular beide mit doppelten geschwungenen Klammern arbeiten, um auf Variablen zuzugreifen, muss dies in Angular geändert werden. Das ist grundsätzlich möglich, kann aber eine Sicherheitslücke darstellen und es wird davon abgeraten [?].

Weiters wird Jekyll hauptsächlich für Blogs und Seiten, die nur Text enthalten, verwendet. Dementsprechend wird wenig Funktionalität für Seiten mit anderem Inhalt bereitgestellt.

### 3.5.3 Scully

Scully ist ein Static Site Generator, der auf Angular aufbaut. Dabei liegt der Fokus jedoch auf serverseitigem Rendern und es werden keine weiteren Tools bereitgestellt, um auf den Aufbau der Seite Einfluss zu nehmen. Somit kann zwar die Zeit, die eine Seite braucht, um im Browser dargestellt zu werden, verringert werden, für das dynamische Generieren der Seite eignet sich Scully aber nicht.

Das serverseitige Rendern wird mithilfe eines sogenannten Headless Browsers bewerkstelligt. Dieser Browser hat die selben Funktionalitäten, wie ein normaler Browser, besitzt jedoch keine grafische Benutzeroberfläche und kann über eine API von einem anderen Programm gesteuert werden.

Diese Technologie wird oft verwendet, um Websites zu *scrapen*, also Websites automatisch aufzurufen und zu analysieren. Ein weiterer Einsatzbereich ist das automatisierte Testen von Webapplikationen. Scully benützt einen Headless Browser, um die Website einmal zu rendern und die entstehenden Dateien zu speichern.

Weiters ist die Community von Scully um einiges kleiner, als die von anderen Static Site Generators. Bei Problemen und Fragen gibt es daher weniger Ressourcen, auf die zugegriffen werden kann.

## 3.6 Deployment Pipeline

Nach dem Download des generierten Projekts sollte die Möglichkeit bestehen, kleine Änderungen daran vorzunehmen und es dann schnell und einfach auf einen Server zu deployen. Der Server ist dabei entweder eine Linux Virtual Machine oder eine *Firebase Hosting* Instanz.

Dazu gab es mehrere Ansätze: Ist das Projekt ein Git-Repository, kann eine Automatisierungssoftware, wie Jenkins oder GitHub Actions verwendet werden, die bei jedem *Push*-Event das Projekt buildet und auf den Server deployed. Die zweite Möglichkeit ist ein Node.js Skript, das im Projekt enthalten ist und von der Kommandozeile aus ausgeführt wird.

### 3.6.1 SSH

Secure Shell (SSH) ist ein Protokoll, das genutzt wird, um eine sichere Verbindung zwischen einem Client und einem Server aufzubauen. Authentifizierung, Befehle, Output und Dateiübertragung sind dabei verschlüsselt. Das Protokoll wird typischerweise benutzt, um Befehle auf einem anderen Computer auszuführen und Dateien sicher zu übertragen. Dabei eignet es sich gut, um solche Prozesse zu automatisieren und wird daher oft in Continuous Integration und Continuous Deployment Pipelines verwendet.

Das Protokoll arbeitet mit dem Client-Server-Modell. Das heißt, dass die Verbindung von einem SSH Client zu einem SSH Server aufgebaut wird. Mit einem Public Key wird die Identität des Servers sichergestellt. Nach dem Verbindungsauftbau werden die übertragenen Daten symmetrisch verschlüsselt und mithilfe von Hashing-Algorithmen wird die Integrität dieser Daten sichergestellt [?].

Dateiübertragung über SSH wird mithilfe des SSH File Transfer Protocols (SFTP) realisiert.

### SSH Keys

Benutzerinnen und Benutzer können zwischen verschiedenen Formen der Authentifizierung wählen. Die beliebtesten Methoden sind Passwort-basierte Authentifizierung und Public Key Authentifizierung. Letztere wird vor allem bei automatisierten Prozessen eingesetzt. Dabei haben die Benutzerin oder der Benutzer ein kryptografisches Schlüsselpaar. Dieses besteht aus Public Key und Private Key. Der Public Key wird am Server hinterlegt und jeder mit einer Kopie des dazu passenden Private Keys kann auf diesen Server zugreifen [?].

### 3.6.2 Firebase Hosting

Firebase ist ein Backend-as-a-Service (BaaS), das von Google angeboten wird. Es besteht aus verschiedenen Produkten, die Softwareentwicklerinnen und Softwareentwicklern das Entwickeln von Backends ermöglicht, ohne sich um Server kümmern zu müssen. Beispiele für Produkte sind Firebase Authentication, das eine einfache Authentifizierung von Usern ermöglicht, Realtime Database, eine NoSQL Datenbank oder Firebase Hosting, das das Hosten von Webanwendungen, statischen Websites oder Microservices übernimmt [?].

Firebase Hosting ist für eine Verwendung von bis zu 10 GB Speicherplatz gratis [?] und stellt mit der Firebase CLI einen einfachen und schnellen Weg zur Verfügung, die gewünschten Dateien hochzuladen. Es eignet sich somit für das Hosten der generierten Websites. Weitere Vorteile sind, dass für jedes Firebase Hosting Projekt eine eigene Domain zur Verfügung gestellt wird und dass die Website auf das globale Content Delivery Network (CDN) verteilt wird, um die Ladegeschwindigkeiten zu erhöhen [?].

### 3.6.3 Jenkins

Jenkins ist ein Open-Source Automatisierungsserver. Das Projekt wird in Java entwickelt und kann mithilfe von Plugins an spezifische Anforderungen angepasst werden [?].

Der Server wird vor allem zur Automatisierung von Aufgaben, wie das Builden, Testen und Deployen von Softwareprojekten genutzt. Jenkins muss selbst gehostet werden. Dazu wird ein offizielles Docker Image im Docker Hub bereitgestellt [?].

Die Konfiguration einer Pipeline wird in einem *Jenkinsfile* vorgenommen, das sich im Git-Repository des Projekts befindet. Darin werden mehrere sogenannte Stages definiert, die dann nacheinander ausgeführt werden. Jede Stage ist wiederum in mehrere Steps gegliedert, die verschiedene Aufgaben übernehmen und auch in der definierten Reihenfolge ausgeführt werden.

Eine Pipeline mit den Stages *Build*, *Test* und *Deploy* würde zum Beispiel folgendermaßen aussehen: [?]

```

1  pipeline {
2      agent any
3
4      stages {
5          stage('Build') {
6              steps {
7                  echo 'Building..'
8              }
9          }
10         stage('Test') {
11             steps {
12                 echo 'Testing..'
13             }
14         }
15         stage('Deploy') {
16             steps {
17                 echo 'Deploying....'
18             }
19         }
20     }
21 }
```

### 3.6.4 GitHub Actions

GitHub Actions ist eine Automatisierungssoftware, die von GitHub zur Verfügung gestellt wird. Im Gegensatz zu Jenkins werden die Server dabei von GitHub gehostet. Die Konfiguration von sogenannten Workflows wird in YAML-Dateien im *.github/workflows* Directory eines Repositories vorgenommen.

GitHub Actions sind tief in das GitHub Ökosystem integriert und bieten viele Möglichkeiten, auf verschiedene Teile davon zuzugreifen. Damit können Workflows erstellt werden, die nicht nur die Software builden, testen und deployen, sondern auch zum Beispiel die Labels von Issues verändern.

Ein Workflow enthält einen oder mehrere Jobs, die sequentiell oder parallel ausgeführt werden können. Jeder Job läuft auf einer eigenen Virtuellen Maschine, auch Runner genannt und ist in verschiedene Actions unterteilt, die wiederum mehrere Steps

enthalten. Dabei läuft auf den Runnern entweder Windows, Linux oder macOS als Betriebssystem.

Ein Workflow, der bei jedem Push oder Pull Request auf den *main* Branch die Steps *Build*, *Test* und *Deploy* ausführt, könnte beispielsweise folgendermaßen aussehen:

[?]

```

1 name: Build & Deploy
2
3 on:
4   push:
5     branches: [ main ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10  build-deploy:
11    runs-on: ubuntu-latest
12    steps:
13      - uses: actions/checkout@v2
14      - name: Build
15        run: echo Building
16      - name: Test
17        run: echo Testing
18      - name: Deploy
19        run: echo Deploy

```

## Secrets

Secrets sind verschlüsselte Umgebungsvariablen, die für eine GitHub Organisation oder ein Repository erstellt werden. Darin können sensible Informationen, wie z.B. Passwörter oder Private Keys gespeichert werden. Diese Secrets sind dann in GitHub Actions Workflows verfügbar. GitHub stellt sicher, dass Secrets verschlüsselt werden, bevor sie die GitHub Server erreichen und verschlüsselt bleiben, bis sie in einem Workflow verwendet werden [?].

| Repository secrets |                                       |                        |   |
|--------------------|---------------------------------------|------------------------|---|
|                    | FIREBASE_SERVICE_ACCOUNT_WEBMAP_F8AB0 | Updated on 14 Jul 2021 | <button>Update</button> <button>Remove</button> |
|                    | REMOTE_HOST                           | Updated on 7 Jul 2021  | <button>Update</button> <button>Remove</button> |
|                    | REMOTE_TARGET                         | Updated on 7 Jul 2021  | <button>Update</button> <button>Remove</button> |
|                    | REMOTE_USER                           | Updated on 7 Jul 2021  | <button>Update</button> <button>Remove</button> |
|                    | SSH_PRIVATE_KEY                       | Updated on 7 Jul 2021  | <button>Update</button> <button>Remove</button> |

Abbildung 12: Repository secrets

In folgendem Step eines Workflows werden diese Secrets dann referenziert:

```

1 - name: Deploy to Server
2   uses: easingthemes/ssh-deploy@v2
3   env:
4     SSH_PRIVATE_KEY: ${{ secrets.SSH_PRIVATE_KEY }}
5     SOURCE: static/
6     REMOTE_HOST: ${{ secrets.REMOTE_HOST }}

```

```

7      REMOTE_USER: ${ secrets.REMOTE_USER }
8      TARGET: ${ secrets.REMOTE_TARGET }

```

### 3.6.5 Node.js Skripts

Node.js 3.3.5 Skripts haben den Vorteil gegenüber dem Ansatz mit einer Automatisierungssoftware, dass dafür kein GitHub Repository erstellt werden muss und nur die Kommandozeile benötigt wird. Da Node.js plattformunabhängig ist, können diese in Skripts sowohl auf Windows als auch auf Linux und macOS ausgeführt werden.

## 3.7 Formate für die Konfigurationsdatei

### 3.7.1 JSON

JavaScript Object Notation (JSON) ist ein Format zum Datenaustausch. Es wurde entworfen, um für Menschen einfach zu lesen und zu schreiben zu sein. Außerdem ist es auch für Maschinen einfach zu parsen oder zu generieren. Das Format ist unabhängig von Programmiersprachen.

JSON baut auf zwei Strukturen auf:

- Object: Eine Sammlung an Key/Value Paaren. In den meisten Programmiersprachen wird das als Object, Record, Struct oder Dictionary umgesetzt.
- Array: Eine geordnete Sammlung an Werten. Diese Struktur wird als Array, Vector, List oder Sequence umgesetzt.

Ein Object ist ein ungeordnetes Set an Key/Value Paaren. Es beginnt mit { und endet mit }. Die darin enthaltenen Werte haben das Format *key: value* und werden mit Beistrichen voneinander getrennt. Der Key ist immer ein String und muss deshalb in Anführungszeichen stehen.

Ein Array ist eine geordnete Sammlung an Werten. Es startet mit [ und wird mit ] beendet. Die darin enthaltenen Werte werden wiederrum mit Beistrichen getrennt.

Werte können ein String sein, der in Anführungszeichen steht, eine Zahl, *true*, *false*, *null* oder wieder ein Object oder Array [?].

Ein valides JSON-Object könnte also folgendermaßen aussehen:

```

1  {
2      "string": "Hello World",
3      "number": 69,

```

```

4   "boolean": false,
5   "null": null,
6   "array": [
7     419,
8     420,
9     "421",
10    null
11  ]
12 }

```

## JSON-Schema

Ein JSON Schema wird dazu verwendet, JSON Strukturen zu beschreiben und zu validieren. Damit kann beispielsweise eine Dokumentation für eine Schnittstelle, die über JSON kommuniziert, definiert werden. Diese Dokumentation ist dabei sowohl für Menschen, als auch für Maschinen einfach lesbar. Ein weiterer Einsatzbereich von JSON-Schemata ist das Validieren von Daten bei automatisierten Tests [?].

Ein JSON-Schema ist selbst wiederrum nur ein JSON-Object. Dieses besitzt meistens einen *title* und eine *description*, die das definierte JSON beschreiben, aber keine Auswirkung auf die Struktur haben. Das *type* Keyword definiert den Typ einer Property. Dieser kann einer der folgenden Strings sein:

- "null"
- "boolean"
- "object"
- "array"
- "number"
- "string"

Ist der Typ "object", muss dieses Object weiter beschrieben werden. Dazu wird das *properties* Keyword verwendet. Der Wert davon ist ein Object mit den Properties des zu beschreibenden Object als Keys. Die Werte dieser Keys sind wiederum Objects, die die jeweilige Property beschreiben. Dazu wird wieder das *type* Keyword verwendet. Mit dem *description* Keyword kann man auch dieser Property wieder eine Beschreibung geben.

Eine Property vom Typ "array" beschreibt mit dem *items* Keyword den Typ der Werte, die das Array enthält.

Für die meisten Typen gibt es noch zusätzliche Keywords, mit denen die Werte noch genauer beschrieben werden können. Dazu zählen beispielsweise *required* für Werte

vom Typ "object", das ein Array der Namen der erforderlichen Properties enthält oder *maximum* und *minimum* für Werte vom Typ "number" bzw. "integer".

Ein JSON-Schema für das JSON-Object aus dem vorherigen Kapitel 3.7.1 könnte beispielsweise folgendermaßen aussehen:

```

1      {
2          "title": "Example JSON Object",
3          "description": "A JSON Object to demonstrate JSON Syntax",
4          "type": "object",
5          "properties": {
6              "string": {
7                  "description": "Some string",
8                  "type": "string"
9              },
10             "number": {
11                 "description": "Some number",
12                 "type": "integer",
13                 "maximum": 100
14             },
15             "boolean": {
16                 "description": "Some boolean",
17                 "type": "boolean"
18             },
19             "null": {
20                 "description": "Null",
21                 "type": "null"
22             },
23             "array": {
24                 "description": "Some array",
25                 "type": "array",
26                 "items": {
27                     "oneOf": [
28                         {
29                             "type": "number"
30                         },
31                         {
32                             "type": "string"
33                         },
34                         {
35                             "type": "null"
36                         }
37                     ]
38                 }
39             },
40             "required": [ "string", "number", "array" ]
41         }
42     }
```

### 3.7.2 XML

Auch die Extensible Markup Language (XML) ist ein Format zum Datenaustausch. Wie auch JSON, ist XML einfach von Menschen sowie von Maschinen lesbar.

XML ist auf sogenannten Tags aufgebaut. Ein Tag beginnt mit < und wird mit > beendet. Zwischen diesen Zeichen steht der Name der Property. Bei Tags wird zwischen Start Tags und End Tags unterschieden. End Tags zeichnen sich dadurch aus, dass sie ein / Zeichen vor dem Namen enthalten. Ein XML Element beginnt mit einem Start Tag und wird mit einem äquivalenten End Tag beendet. Diese Elemente können einen Wert oder eines oder mehrere Unterelemente enthalten [?].

Die meisten XML Dokumente enthalten außerdem eine sogenannte XML Declaration. Darin sind Metadaten des Dokuments enthalten. Meistens sieht diese XML Declaration folgendermaßen aus:

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

Hier wird die Version des verwendeten XML Standards und die Codierung des Dokuments festgelegt.

Ein zu dem JSON-Objekt aus dem vorherigen Kapitel äquivalentes XML-Element könnte wie folgt aussehen:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <example>
3   <string>Hello World</string>
4   <number>69</number>
5   <boolean>false</boolean>
6   <null>null</null>
7   <array>
8     <element>419</element>
9     <element>420</element>
10    <element>421</element>
11    <element>null</element>
12  </array>
13 </example>
```

Auch wenn dies hier aufgrund der Namen der Elemente angenommen werden könnte, wird im Gegensatz zu JSON bei XML nicht zwischen verschiedenen Datentypen unterschieden.

## XML Schema

Mit einem XML Schema kann man die Struktur eines XML Dokuments beschreiben und validieren. Dies kann beispielsweise dazu verwendet werden, Schnittstellen zu dokumentieren. XML Schema hat den Vorteil gegenüber anderen Sprachen, mit denen ein XML Dokument beschrieben werden kann, dass es selbst in XML geschrieben ist. Es muss also keine weitere Sprache gelernt werden [?].

Ein XML Schema wird in einem `<xs:schema>` Element beschrieben. Darin befinden sich `<xs:element>` Elemente, die in Simple Types und Complex Types eingeteilt werden. Complex Types enthalten weitere Elemente, Simple Types nicht.

Auch wenn in der XML Syntax nicht zwischen Datentypen unterschieden wird, kann in einem XML Schema der Datentyp eines Elements angegeben werden.

Obiges XML Dokument könnte beispielsweise mit folgendem XML Schema beschrieben werden:

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
2   <xs:element name="example">
3     <xs:complexType>
4       <xs:sequence>
```

```

5      <xs:element type="xs:string" name="string"/>
6      <xs:element type="xs:integer" name="number"/>
7      <xs:element type="xs:boolean" name="boolean"/>
8      <xs:element type="xs:string" name="null"/>
9      <xs:element name="array">
10         <xs:complexType>
11             <xs:sequence>
12                 <xs:element type="xs:string" name="element" maxOccurs="unbounded"/>
13             </xs:sequence>
14         </xs:complexType>
15     </xs:element>
16     </xs:sequence>
17   </xs:complexType>
18 </xs:element>
19 </xs:schema>

```

## Valid und Well-Formed

XML Dokumente können den Status Valid und Well-Formed haben.

Well-Formed bedeutet, dass das Dokument der XML Syntax entspricht [?]. Dazu zählen folgende Kriterien:

- Das Dokument beginnt mit einer XML Declaration
- Start Tags haben passende End Tags
- Es gibt ein eindeutiges Root Element

Weiters kann ein Dokument Well-Formed sein. Das bedeutet, dass es außerdem der in einem XML Schema definierten Struktur entspricht [?].

## 3.8 Backend

Das Backend von webmap erfüllt zwei Aufgaben:

- Das Hosten des Generators
- Eine REST-API für das Generieren eines Angular-Projekts aufgrund der im Generator erstellten Konfiguration

Für das Hosting wurde nginx verwendet und die REST-API wurde in TypeScript implementiert.

### 3.8.1 TypeScript

Um webmap leicht warten zu können, wurde auch das Backend in TypeScript 3.3.5 geschrieben. Damit wird nur eine Programmiersprache für das gesamte Projekt verwendet. Ausgeführt wird das Backend in der Laufzeitumgebung Node.js. Diese wurde

entwickelt, um skalierbare Netzwerkanwendungen zu erstellen und eignet sich daher für die REST-API [?].

### **express.js**

Die API wurde mit dem Node-Package *express.js* entwickelt. Express ist ein schnelles und minimalistisches Web Framework. Die Kernfunktion liegt darin, schnelle und robuste APIs zu entwickeln [?].

### **node-tar**

Zum Komprimieren des generierten Projekts wird das Node-Package *node-tar* verwendet. Das Package ahmt dabei den UNIX-Befehl *tar* nach [?].

Ein *tar*-File oder *tarball* ist ein Archiv von Filesystem-Einträgen wie Dateien, Directories oder Symbolic Links. Diese können dann noch komprimiert werden, um ein *.tar.gz*-File zu erhalten. Tar wird hauptsächlich zur Distribution oder für Backups von mehreren Dateien verwendet [?].

### **ESLint**

ESLint ist ein Open-Source Linting Tool für JavaScript und TypeScript. Es wird verwendet, um geschriebenen Code zu analysieren, um Muster zu finden, von denen man weiß, dass sie problematisch und anfällig für Bugs sind. Weiters kann damit ein bestimmter Code Style definiert werden, um den Code konsistent und lesbar zu machen. Alle diese Aspekte werden als sogenannte Rules definiert. Die Library stellt eine Menge an vorgefertigten Rules zur Verfügung, es können aber auch sehr einfach eigene Rules erstellt werden [?].

## **3.9 nginx**

nginx ist ein Open-Source HTTP Server, Reverse Proxy Server, Mail Proxy Server und generischer TCP/UDP Proxy Server. Konfigurationen werden im */etc/nginx/nginx.conf* File durchgeführt. Standardmäßig stellt der Webserver alle Dateien im */usr/share/nginx/html* Directory Clients zur Verfügung [?]. Auch für nginx steht ein offizielles Docker Image am Docker Hub zur Verfügung [?].

## Reverse Proxy

Ein Reverse Proxy Server wie nginx befindet sich typischerweise hinter einer Firewall in einem privaten Netzwerk und verteilt Requests von Clients auf die entsprechenden Backend Server. Damit erhält man eine weitere Abstraktionsschicht, um den Verkehr zwischen Clients und Servern zu verwalten und zu überwachen [?].

nginx kann mit folgendem *nginx.conf* File als Proxy Server konfiguriert werden, der Requests zu */api* an einen anderen Server weiterleitet. Bei allen anderen Requests wird eine Datei aus dem */data* Directory als Response gesendet.

```

1 events {};
2
3 http {
4     include mime.types;
5
6     server {
7         location /api {
8             proxy_pass http://localhost:4000/;
9         }
10
11        location / {
12            root /data;
13        }
14    }
15 }
```

# 3.10 Containerization

## 3.10.1 Docker

Docker ist eine Softwareplattform, die Benutzerinnen und Benutzern erlaubt, Applikationen schnell zu Builden, Testen und Deployen. Dabei wird die Software in standardisierte Pakete, die Container genannt werden, verpackt [?].

### Container

Container sind abgeschlossene Systeme, die Software sowie die gesamte Infrastruktur, die diese Software benötigt, um zu laufen, enthält. Dazu gehören Libraries, sowie eine Laufzeitumgebung. Die Idee von Containern erinnert an Virtuelle Maschinen. Bei Containern wird jedoch das Betriebssystem virtualisiert, im Gegensatz zu Virtuellen Maschinen, bei denen die Hardware virtualisiert wird. Deshalb benötigt jede VM eine eigene Kopie eines Betriebssystems, was mehrere GB an Speicherplatz beansprucht. Container haben hingegen typischerweise eine Größe von weniger als 100 MB [?].

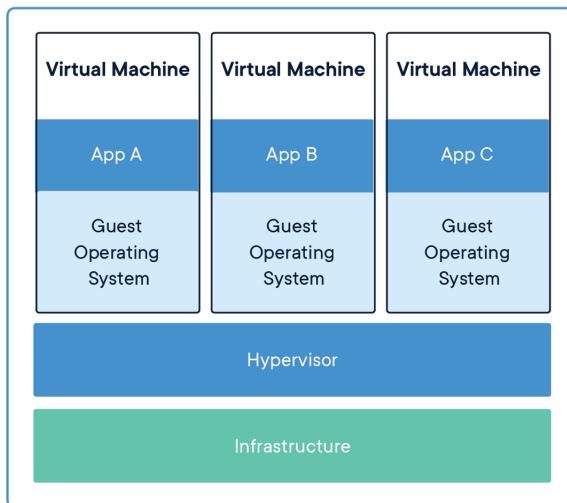


Abbildung 13: Virtualisierung mit Virtuellen Maschinen [?]

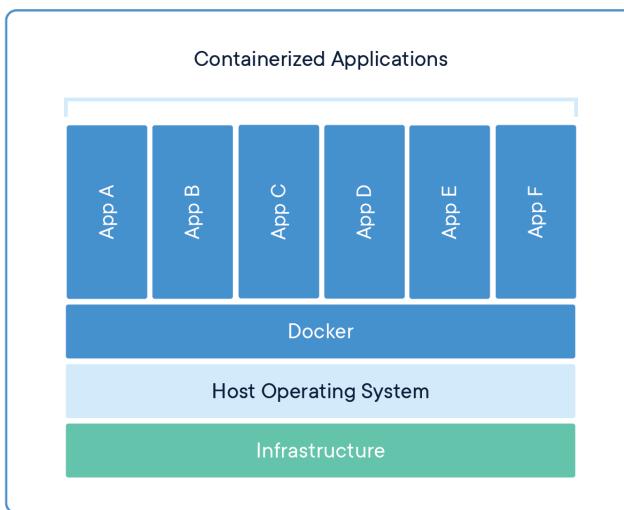


Abbildung 14: Virtualisierung mit Containern [?]

## Image

Container werden aus sogenannten Images erstellt. Diese Images erhalten alles, was ein Container benötigt, um zu laufen. Dazu gehören das Filesystem mit allen Dependencies und Konfigurationen, Umgebungsvariablen und ein Befehl, um die Software im Container zu starten. Beim Start des Containers wird der Inhalt des Images kopiert und darin der Befehl ausgeführt, der die Software startet. Dafür wird der Docker CLI Befehl `docker run [image name]` verwendet [?].

Images können entweder von einem Registry heruntergeladen oder selbst in einem sogenannten *Dockerfile* definiert werden. Jedes *Dockerfile* startet mit einem Base Image, auf dem aufgebaut wird. Danach können zum Beispiel Files in das Image kopiert werden, aus denen dann eine Applikation gebaut wird. Zum Schluss benötigt ein *Dockerfile* noch einen Befehl, der beim Start eines Containers ausgeführt wird.

Folgendes Dockerfile verwendet das Base Image *ubuntu*. Das gesamte Directory, in dem sich das *Dockerfile* befindet, wird daraufhin in das */app* Directory des Images kopiert. Darin wird der *make* Befehl ausgeführt, um die Applikation zu bauen. Als Befehl, der beim Start ausgeführt wird, wird *python /app/app.py* definiert [?].

```
1  FROM ubuntu:18.04
2  COPY . /app
3  RUN make /app
4  CMD python /app/app.py
```

## Persistierung

Docker bietet zwei Wege an, um Daten, die in Containern generiert werden, zu persistieren. Bind Mounts synchronisieren ein Directory im lokalen Filesystem mit einem Directory im Filesystem des Containers. Diese Bind Mounts werden jedoch öfter dazu verwendet, um Daten von außerhalb in den Container zu kopieren.

Um Daten, die im Container generiert werden, zupersistieren werden meistens Volumes verwendet. Volumes enthalten ein Directory des Containers und werden vollständig von Docker verwaltet. Wird der Container gestoppt oder gelöscht, bleiben die Daten im Volume erhalten.

Volumes werden beim Erstellen des Containers konfiguriert. Die Docker CLI stellt dafür die *-v* bzw. *--volume* Flag zur Verfügung [?].

## Microservices

Microservices beschreibt einen Ansatz in der Softwareentwicklung, bei dem die Software in kleine, unabhängige Services aufgeteilt ist, die über definierte APIs miteinander kommunizieren. Diese Architektur machen Applikationen leichter skalierbar. Diese Services werden oft als Docker Container realisiert.

Das Gegenteil der Microservice Architektur ist ein Monolith. Dabei ist die gesamte Applikation ein einzelner Service. Dieser enthält alle Komponenten, die eng miteinander verbunden sind. Bei erhöhter Auslastung muss dann die gesamte Applikation skaliert werden. Jede zusätzliche Komponente erhöht außerdem die Komplexität des gesamten Services. Das macht es schwieriger, mit neuen Technologien zu experimentieren und neue Ideen einzubauen. Durch die Abhängigkeit der Komponenten voneinander kann ein einzelner Fehler in einem Teil des Services eine große Auswirkung auf die gesamte Applikation haben.

Microservices sollen diese Probleme lösen. Die Komponenten funktionieren unabhängig von einander und so muss bei der erhöhten Auslastung eines Services nur dieser skaliert werden. Da die Services über eine definierte API miteinander kommunizieren kann die Technologie in den Services einfach ausgetauscht werden. Weiters beschränken sich die Ausmaße von Fehlern nur auf den Service, in dem sie auftreten.

Mit der steigenden Verbreitung von Cloud Computing werden Microservices immer beliebter, da die Skalierung von Services dabei automatisiert werden kann [?].

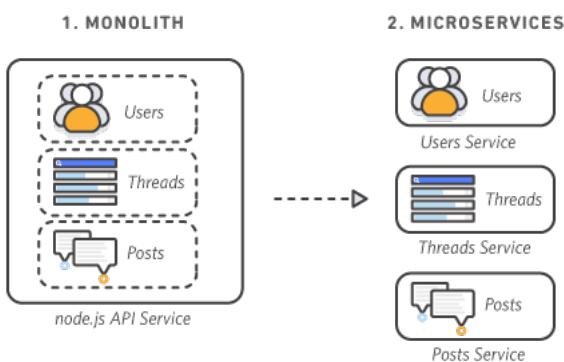


Abbildung 15: Monolith vs. Microservices [?]

## Docker Hub

Docker Images können in sogenannte Registries hochgeladen und so verteilt werden [?]. Das größte Registry ist der Docker Hub [?]. Darin werden auch offizielle Images für Softwarepakete, wie Ubuntu, nginx oder Node.js zur Verfügung gestellt. Diese Images werden immer auf dem neusten Stand gehalten, um die Sicherheit zu gewährleisten [?].

### 3.10.2 Docker Compose

Docker Compose ist ein Tool zur Orchestrierung von Docker Containern. Es wird verwendet, um Applikationen mit mehreren Containern zu verwalten. In einem *YAML* File werden die Services der Applikation definiert, die dann alle mit einem einzelnen Befehl gestartet werden können [?].

## Networking

Alle Services, die im gleichen File definiert werden, befinden sich automatisch im selben Docker Netzwerk. Das bedeutet, dass sie miteinander kommunizieren können, aber nicht von außen auf sie zugegriffen werden kann. Jeder Service kann jedoch einzelne Ports

definieren, die auch von außerhalb erreichbar sein sollen. Diese Abstraktionsschicht erhöht die Sicherheit, da zum Beispiel nur Services innerhalb eines Docker Netzwerks auf eine Datenbank zugreifen können, diese aber von Prozessen außerhalb nicht erreichbar ist. Docker Compose erleichtert das Routing innerhalb der Docker Netzwerke außerdem, indem der Hostname eines Services gleich seinem Namen ist [?].

# 4 Umsetzung

## 4.1 webmap Workflow

Um den Geschäftsprozess besser visualisieren zu können, wurde zu Beginn der webmap Workflow definiert.

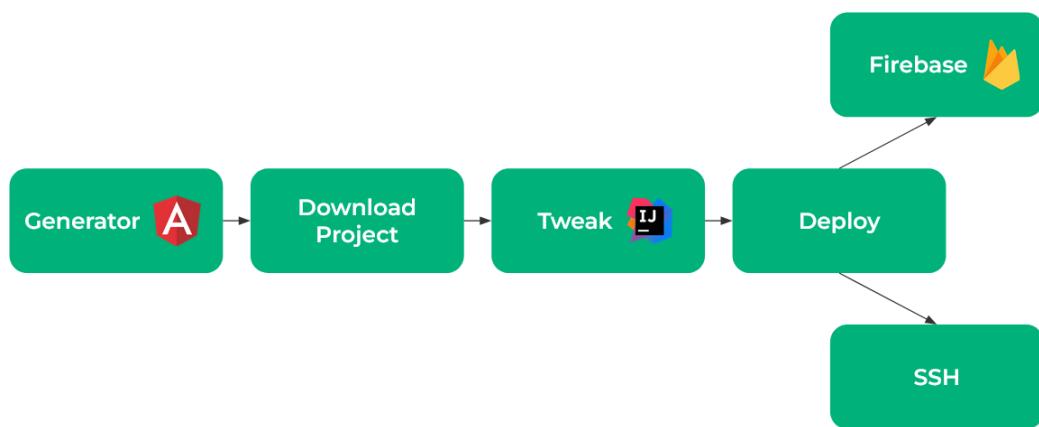


Abbildung 16: webmap Workflow

In diesem wurden die einzelnen Schritte festgelegt, die eine Benutzerin oder ein Benutzer von webmap von der Eingabe der Daten bis zu einer fertigen Website durchläuft.

### 4.1.1 Generator

Der Generator ist der erste Schritt des webmap Workflows. Darin werden die Daten eingegeben, die später mithilfe der generierten Website visualisiert werden. Realisiert wurde dies mit einer Webanwendung.

### 4.1.2 Download Project

Mit dem Klick auf einen Button im Generator wird ein Angular-Projekt, das die zuvor eingegebenen Daten enthält, heruntergeladen.

### 4.1.3 Tweak

Danach kann dieses Projekt in einem Editor oder einer IDE geöffnet werden. Darin können noch Änderungen vorgenommen werden, um die Website perfekt auf die Kundin oder den Kunden anzupassen.

### 4.1.4 Deploy

Zum Schluss besteht noch die Möglichkeit, die Website auf einen Server zu deployen. Dabei kann zwischen einer Firebase Hosting Instanz oder einer Linux Virtual Machine entschieden werden.

## 4.2 Frontend Implementierung

Die Anforderungen an das Frontend sind sehr vielseitig, einerseits soll es wie eine Präsentation mit mehreren Slides aufgebaut, andererseits soll das gesamte Projekt leicht wartbar und individualisierbar bleiben. Die Daten werden dynamisch aus einem Konfigurationsfile ausgelesen, welches durch einen eigenen Generator generiert werden kann.

Um die UserExperience zu verbessern, wird auch auf wichtige Dinge wie Angular Routing oder angemessenes styling Wert gelegt. Nicht zu vernachlässigen ist dabei der Fakt, dass webmap interaktiv sein soll. Das bedeutet, dass der Userin oder dem User die Möglichkeit geboten werden soll, durch Slider oder true/false Switches die Darstellung der aktuellen Daten auf der Page zu manipulieren.

### 4.2.1 Konfigurationsfile

Das Konfigurationsfile ist ein JSON File, welches festlegt welche Slides (Pages) existieren, aber auch die zu darstellenden Daten definiert. Hinweis: Grafische Darstellungen der Konfiguration sind weiter unten, bei anderen Kapiteln angeführt.

```

1   {
2     "name": "webmap showcase",
3     "description": "this config is made to show how webmap configs look like",
4     "pages": [
5       {
6         "title": "Bersbuch",
7         "subtitle": "Anreise Dauer zum Bahnhof Bersbuch.",
8         "description": [
9           "Bersbuch ist ein Dorf in der Gemeinde Andelsbuch im Bregenzerwald."
10        ],
11        "mapOptions": {
12          "zoom": 13,
13          "center": [
14            47.40198749647868,
15            9.861946105957031
16          ]
17        },
18        "datasources": [
19          {
20            "options": {},
21            "id": "bersbuch-walk"
22          }
23        ],
24        "controls": {
25          "filters": [
26            {
27              "name": "Anreise Dauer Rad",
28              "type": "slider",
29              "dataSource": "bersbuch-bike",
30              "unit": "Sekunden",
31              "filterValue": "time"
32            }
33          ]
34        }
35      }
36    ],
37    "datasources": [
38      {
39        "options": {},
40        "data": "bersbuch_walk.json",
41        "id": "bersbuch_walk",
42        "colorScheme": "my-color-scheme",
43        "colorSchemeOrderValue": "time",
44        "tooltip": [
45          {
46            "type": "value",
47            "name": "time"
48          },
49          {
50            "type": "text",
51            "content": " Sekunden"
52          }
53        ]
54      }
55    ],
56    "customColorSchemes": [
57      {
58        "id": "my-color-scheme",
59        "colors": [
60          "#f6d2a9",
61          "#f5b78e",
62          "#f19c7c",
63          "#ea8171",
64          "#dd686c",
65          "#ca5268",
66          "#b13f64"
67        ]
68      }
69    ]
70  }

```

## Allgemeine Properties

**Name** Verleiht der ganzen Seite einen Namen. Wird aktuell nicht angezeigt, kann aber durch Individualisierungen verwendet werden.

**Description** Beschreibt die erzeugte Website etwas genauer. Wird aktuell nicht angezeigt, kann aber durch Individualisierungen verwendet werden.

## Pages

*Pages* ist ein Array, welches die Ansichten definiert. Jedes Element im Array, hat eine eigene Map Ansicht, sowie eigene Daten, die in der Sidebar angezeigt werden.

**Title** Der Title ist der Name der Seite und wird groß hervorgehoben angezeigt.

**Subtitle** Der Subtitle beschreibt in wenigen Wörtern und etwas genauer worum es bei der ausgewählten Karte geht.

**Description** Description ist in diesem Fall ein Array von Strings, das bedeutet, es können mehrere Texteinträge angeführt werden, was wiederum einen designtechnischen Grund hat. Für jeden Eintrag wird ein eigener Absatz in der Sidebar generiert.

**Map Options** Hier ist es möglich in einem freien JSON Feld Leaflet-Map-Options übergeben werden, um die Darstellung der Map gezielt zu verändern. Ein häufiger Anwendungsfall dafür ist es, den Ausrichtungspunkt zu ändern, um automatisch den richtigen Kartenausschnitt präsentiert zu bekommen.

**Data Sources** Bei dem Array *Data Sources* können Geo-Daten eingebunden werden, weiter unten werden diese global definiert und hier können diese referenziert werden. Zusätzlich können wieder *options*, in Form von einem JSON Objekt, übergeben werden.

**Controls** Hier werden interaktive Elemente ausgewählt werden. Es gibt vordefinierte Controls, aus denen man wählen kann. Nach Absprache mit triply wurde entschieden, dass für diese Version ein Filter ausreichend ist. Dieser Filter ist ein Slider, welcher auf eine bestimmte Property gebindet werden kann (*filterValue*). Mit dem Attribut *dataSource* wird die zutreffende Data Source ausgewählt. Zusätzlich kann noch eine Einheit (*unit*) und auch ein Name für den Filter angegeben werden.

## Data Sources

Ist ein Array welches mehrere Data Sources beinhalten kann. Jedes JSON-Objekt hat mehrere Attribute die eine Data Source beschreiben. Diese werden global definiert und können in jeder Page auch mehrmals referenziert werden.

**ID** Die ID wird benötigt, um eine Data Source referenzieren zu können, zum Beispiel bei der Erstellung von Pages.

**Data** Data ist ein File-upload Feld, bei dem das Geo-JSON File hochgeladen werden kann, in welchem die Daten gespeichert sind.

**Options** Hier können Leaflet-Map-Options in einem freien JSON-Feld übergeben werden, um die Art der Darstellung oder das Verhalten der Map zu verändern.

**Color Scheme** Dieses Feld referenziert auf die ID eines Color Schemes, entweder gibt man den Namen eines Carto-Color-Schemes an oder man erstellt ein eigenes Custom-Color-Scheme.

**Color Scheme Order Value** Um das Color-Scheme richtig anzuwenden, braucht man einen Value, um eine angemessene Skalierung zu berechnen, dieser Attributname kann hier angegeben werden.

**Tooltip** Der Tooltip ist ein Feature, welcher es ermöglicht, Information beim hovern einer Data Source auf der Map anzuzeigen. Es ist deshalb ein Array, damit man unbegrenzt viele Informationen angeben kann. Jedes JSON-Objekt in diesem Array besitzt genau 2 Attribute, *type* und *name* oder *content*. Der *type* gibt an, ob es sich um einen Value oder einen Text handelt, bei einem Value muss der Attributname des Values bei dem anderen Attribute (*Name*) angegeben werden. Wird jedoch der Typ *Text* ausgewählt, so muss lediglich ein String angegeben werden, welcher angezeigt wird (zum Beispiel eine Einheit für den Value zuvor).

## Custom Color Schemes

Hierbei handelt es sich um ein Array, in dem individuelle Color Schemes gespeichert werden können. Wichtig, dass auch hier jedes JSON-Objekt eine ID erhält, um das Scheme referenzieren zu können. Weiters befindet sich ein Array *colors* in diesem Objekt, welches die HEX-Codes der Farben beinhaltet.

### 4.2.2 Frontend Architektur und Aufbau

Auf der linken Seite gibt es eine Sidebar in der alle Informationen angezeigt werden, welche für die aktuell ausgewählte Seite relevant sind. Auf der rechten Seite, wird die Map dargestellt, auf dieser werden die Data Sources angezeigt, sowie die Map-Options angewandt. Die Services verarbeiten und verteilen Daten zwischen den Components.

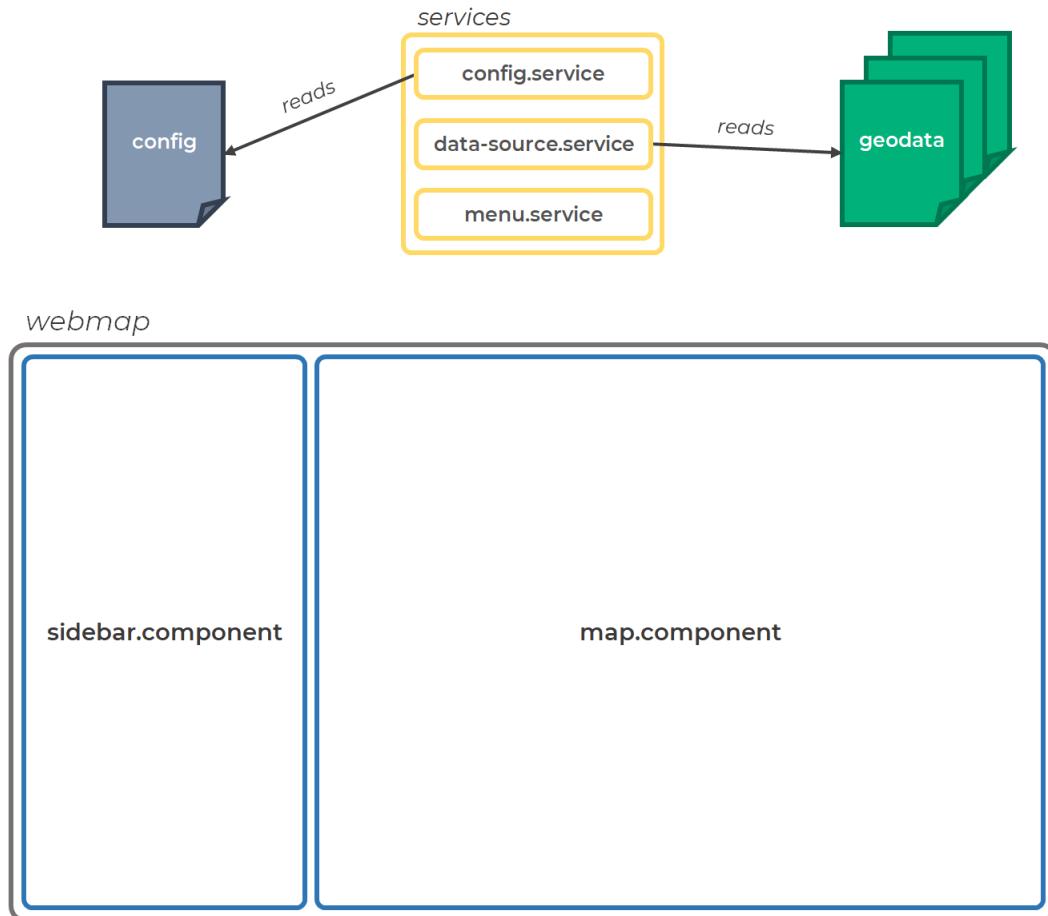


Abbildung 17: Frontend Architektur

### 4.2.3 Services

Services sind bei Angular ein spezielles Konstrukt zur Datenverarbeitung, der Hauptanwendungsfall ist der Austausch von Daten zwischen mehreren Components. Der große Vorteil von Services ist, dass es dabei um Singletons handelt. Das bedeutet, es gibt genau eine Instanz für alle Komponenten, somit ist es möglich, dass alle auf die gleichen Daten zugreifen.

#### Config Service

Der Config Service liest das Konfigurationsfile aus und bietet diesen Inhalt an andere Components an. Über die App Component wird die Konfiguration angefordert und weiterverarbeitet, so werden die Data Source Konfigurationsdaten und die Custom Color Schemes herausgefiltert und an den 4.2.3 weitergegeben.

#### Data Source Service

Das Ziel mit diesem Service ist es, einen globalen Pool an Data Sources zu erstellen, aus der jede Component jederzeit die benötigten Data Sources anfordern und referenzieren kann. Somit muss jede Data Source nur einmal erstellt und kann endlose Male weiterverwendet werden.

Jede Data Source besteht aus mehreren Layern, dadurch ist eine Data Source eine Leaflet *LayerGroup*. Der Service erhält zu Beginn die Informationen aus dem Konfigurationsfile, zuerst werden die Geo-Daten ausgelesen und entsprechend verarbeitet. Es werden die verschiedenen Layer entsprechend der Daten erstellt und eingefärbt, wenn angegeben, wird hier auch der Tooltip zum Layer hinzugefügt.

Es werden alle *Geometry Types* laut der *GeoJSON Specification (RFC 7946)* [?] unterstützt:

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon

- *GeometryCollection* ist nicht offiziell im Standard inkludiert, wird aber auch unterstützt. Diese Art von *Geometry Types* kann mehrere andere *Types* beinhalten.

## Menu Service

Um die Navigation zu erleichtern, kann man mit den Pfeiltasten zwischen den Pages wechseln, aber auch ein globales Menü steht zur Verfügung, um direkt zur gewünschten Page zu gelangen. Dabei hat man eine kurze Übersicht über alle Pages und mit einem Klick auf die entsprechende Seite gelangt man auch dorthin.



Abbildung 18: Menü zur Page Auswahl

### 4.2.4 Custom Color Schemes

Color Schemes ermöglichen es, Bereiche auf Karten visuell besser darzustellen. Es werden standardmäßig die Schemes der CartoColor Library verwendet, welche exzellente Schemes für Karten bereitstellt. Ein Color Scheme besteht aus mehreren Farben, welche durch Abstufungen angeordnet sind (meistens von dunkel bis hell). Auch bei den Individuellen Schemes ist dies nicht anders, es wird ein eindeutiger Name vergeben und HEX-Codes zugeordnet.

Der 4.2.3 stellt sicher, dass alle Custom Color Schemes importiert und verwendet werden, wenn sie so konfiguriert sind.

### 4.2.5 Sidebar

Die Sidebar auf der linken Seite bietet Platz, um Informationen anzuzeigen. Ganz oben ist der *title* zu sehen, darunter der *subtitle*, der zu sehende Textblock, ist die *description*,

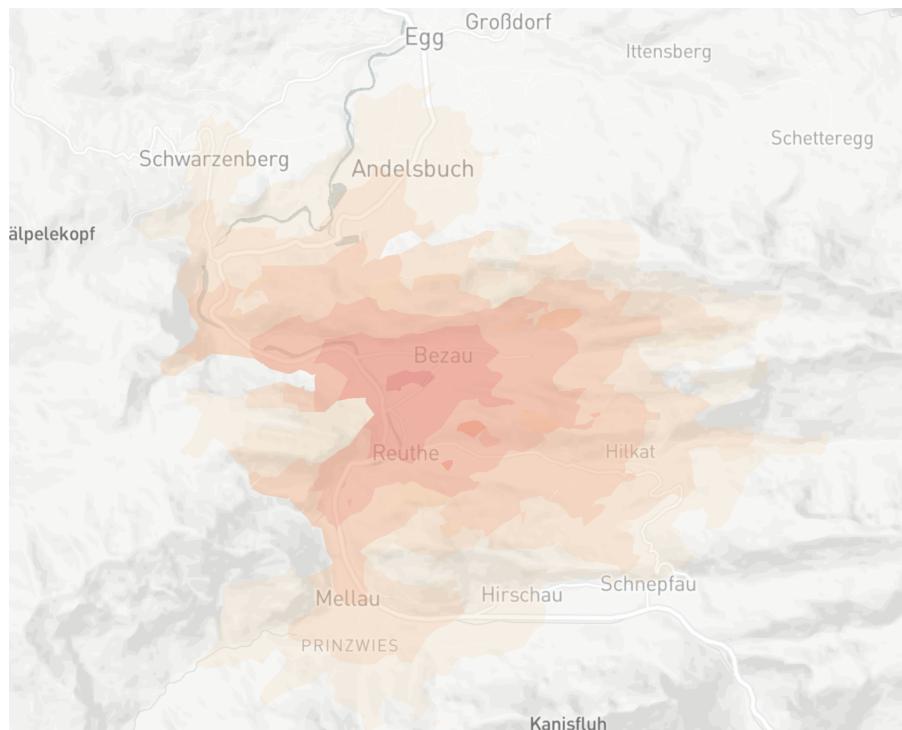


Abbildung 19: Color Scheme Beispiel: Je heller die Farbe, desto länger dauert die Anreise mit dem Fahrrad in die Mitte

für welche für jeden Array Eintrag ein eigener Absatz erstellt wird. Am unterende Ende der Sidebar sind die Filter angeordnet, für jede Data Source wird standardmäßig ein Sichtbarkeits-Filter angelegt, mit dem sich die Data Source ein- oder ausblenden lässt. Der zweite unterstütze Filter ist der *Slider*. Mit diesem kann man den Wertebereich der angezeigten Daten verändern, zum Beispiel, wenn man nur die Daten zwischen einer Anreisedauer von 800 Sekunden bis 1000 Sekunden angezeigt haben möchte, kann das hier gemacht werden. Gelöst wird das ganze mit einem Value-Binding auf das Attribut, welches in der Konfiguration angegeben wird, auch die Skalierung des Slider wird automatisch berechnet.

### 4.2.6 Map

In der Map werden alle Daten auf der Karte visualisiert. In den hochgeladenen Geo-JSON Files sind unzählige Koordinaten enthalten, welche punkteweise auf der Karte dargestellt werden. Je nach Art der Darstellung werden anschließend Linien zwischen Punkten berechnet, welche auch Formen ergeben können. Die Arten dieser Visualisierungen sind im RFC 7946 [?] Standard genau beschrieben, diese werden ebenso bei webmap angewandt.

### ☰ Bezau

#### Anreise Daten zum Bahnhof Bezau.

Bezau ist eine Marktgemeinde im österreichischen Bundesland Vorarlberg im Bezirk Bregenz mit 2031 Einwohnern (Stand 1. Jänner 2021).

Die Gemeinde ist Hauptort des Bregenzerwalds und Sitz des Bezirksgerichts.

Der Name „Bezau“ (1248: Baezenouwe, später Baetenow, Beznau, dann Bezau) ist zweigeteilt. Der erste Namensteil „Bez“ bzw. „Baez“ soll sich, wie in „Bersbuch“, von Bären ableiten. Der zweite Namensteil „au“ hat im Auwald seine Wurzeln.

Bezau wäre somit als die Bärenau zu verstehen.



Abbildung 20: Sidebar mit Beispiel Daten

Durch den Data Source Service gibt es bereits eine Sammlung an LayerGroups, welche nur noch angezeigt werden müssen. Das erhöht nicht nur die Performance, wenn die Page gewechselt wird, sondern macht die gesamte Applikation gut strukturiert.

Wenn alle Components eingebunden sind, kann das Frontend so ausschauen:

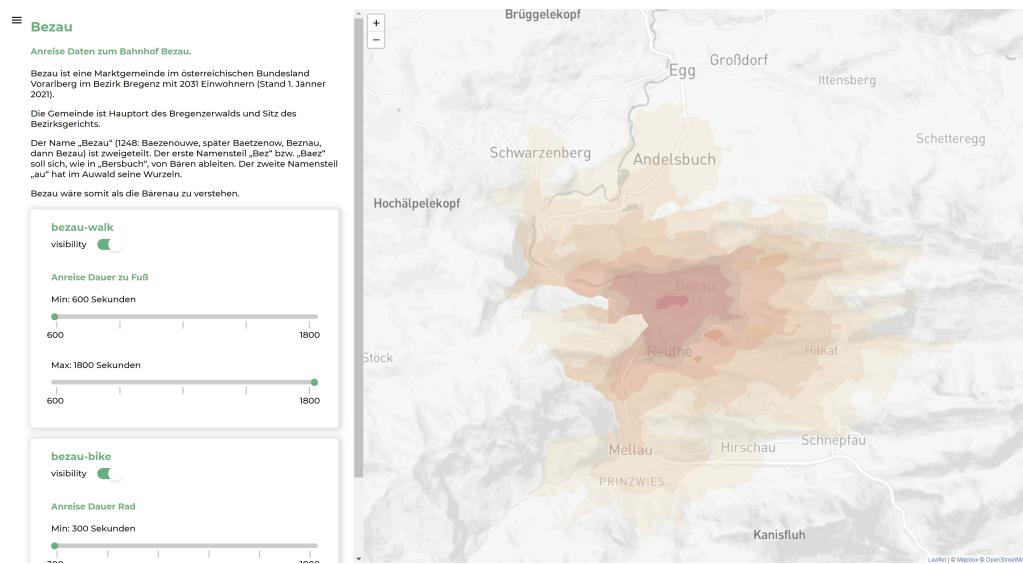


Abbildung 21: Webmap Frontend

## 4.3 Generator Implementierung

Der Generator soll dazu dienen, ein Konfigurationsfile bzw. ein Angular Projekt basierend auf diesem Konfigurationsfile schnell und einfach in einer Benutzeroberfläche zu erstellen.

### 4.3.1 JSON Schema

Da der Generator dynamisch aufgebaut sein sollte und die Möglichkeit bestehen sollte, das Format der generierten Daten zu verändern, fiel die Entscheidung darauf, den Generator aufgrund eines JSON Schemas, das die Daten beschreibt, aufzubauen.

Das Format des Schemas wurde daher so angepasst, dass es die benötigten Daten möglichst genau beschreiben kann.

Jede Property hat ein *type* und *description* Feld. *type* ist erforderlich, während *description* optional ist. Der Wert von *type* kann einer der folgenden Strings sein:

- "object"
- "array"
- "string"
- "number"
- "boolean"
- "file"

Ist die Property ein Object, hat es die Felder *properties* und *required*, die beide optional sind. *properties* ist ein Object mit den Namen der Properties des zu beschreibenden Objects als Keys. Der Wert ist dabei wiederum eine Property. Werden keine Properties definiert, kann im Generator an dieser Stelle ein beliebiges JSON-Object eingegeben werden. *required* ist ein Array mit den Namen der Properties, die erforderlich sind.

Ist die Property ein Array, hat es zusätzlich das erforderliche *items* Feld. Der Wert beschreibt die Properties, die sich in diesem Array befinden. Im Gegensatz zur offiziellen Definition des JSON-Schemas kann hier nur *eine* Property definiert werden.

Properties vom Typ "number" können optional die Felder *minimum* und *maximum* enthalten.

Properties, die den Typ "file" haben, können im optionalen Feld *fileExtensions* die erlaubten Dateiendungen als Array definieren. Im Generator wird an dieser Stelle ein File-Upload angezeigt.

Properties vom Typ "string" und "boolean" haben keine zusätzlichen Felder.

## 4.4 Angular Projekt

Da sich reaktive Formulare einfach in Angular umsetzen lassen, fiel die Entscheidung auf dieses Framework. Ein weiterer Vorteil ist, dass damit der Generator und das Frontend beide in Angular umgesetzt sind. Das erleichtert die Wartung des gesamten Projekts.

Das Kernstück des Generators ist das *src/assets/schema.json* File, in dem sich das Schema für das zu generierende Konfigurationsfile befindet. Aufgrund dessen werden Input-Felder generiert, in welchen Benutzerinnen und Benutzer die entsprechende Konfiguration eingeben können.

Für jeden Typ, der eine Property haben kann wurde eine Angular-Komponente definiert. Weiters wurde mit sogenannten *FormControl* Objekten der Inhalt der Input-Felder auf die im Schema definierten Kriterien überprüft. So kann man zu jeder Zeit einfach überprüfen, ob alle eingegebenen Werte valide sind.

Beim Laden der Seite wird also für jede Property, die im Schema definiert ist, eine entsprechende Komponente generiert. Beim Klick auf einen Button werden dann alle Werte, die in die Komponenten eingegeben wurden, in ein entsprechendes JSON-Objekt gespeichert, welches dann als Konfigurationsdatei an das Backend gesendet wird.

Um den Fokus auf die Funktionalität legen zu können, wurde für das Design *Angular Material* verwendet.

### 4.4.1 Object Component

Ein Object Component dient lediglich zur Visualisierung, welche Properties zu welchem Objekt gehören. Die Funktionalität befindet sich hauptsächlich in den anderen Komponenten.

Eine Ausnahme bildet ein Objekt, bei dem im Schema keine *properties* definiert sind. In diesem Fall wird ein Textfeld angezeigt, in welchem ein beliebiges JSON-Objekt ein-

gegeben werden kann. Mittels *FormControl* wird laufend überprüft, ob der eingegebene Wert ein valides JSON-Objekt ist.

Dazu wird die in JavaScript standardmäßig vorhandene Funktion *JSON.parse* verwendet. Diese Methode erhält einen beliebigen JSON-String als Parameter und gibt den Inhalt als JavaScript-Objekt zurück. Ist dieser String kein valider JSON-String, wird ein Error geworfen.

Bei jeder Änderung im Textfeld wird also *JSON.parse* mit dem Inhalt des Textfelds als Parameter aufgerufen. Wird dabei ein Error geworfen, ist der Inhalt kein valides JSON-Objekt.

#### 4.4.2 Array Component

Ein Array Component listet alle Werte, die sich in diesem Array befinden, auf. Mit einem Button kann ein neuer Wert hinzugefügt werden.

#### 4.4.3 File Component

Bei einem File Component wird ein Button zum Uploaden von Files angezeigt. Ein *FormControl* Objekt überprüft bei einem Upload, ob die Dateiendung mit den im Schema definierten Endungen übereinstimmt. In den Namen eines hochgeladenen Files wird der momentane UNIX-Timestamp eingefügt und es wird zwischengespeichert. Das hat den Vorteil, dass mehrere Files mit dem gleichen Namen hochgeladen werden können, ohne sich gegenseitig zu überschreiben. Um das Konfigurationsfile übersichtlich zu halten, werden die hochgeladenen Files darin nur mit ihrem Namen referenziert.

#### 4.4.4 Number Component

Ein Number Component besteht aus einem einfachen Input-Feld, in das nur Zahlen eingegeben werden. Dabei wird auch das im Schema definierte *minimum* und *maximum* überprüft.

#### 4.4.5 String Component

Der String Component wird durch ein Input-Feld dargestellt, in das beliebige Zeichen eingegeben werden können.

## 4.4.6 Boolean Component

Der Boolean Component wurde mit einem Schieberegler realisiert, der zwei Zustände annehmen kann: *true* und *false*.

## 4.4.7 Generieren des Konfigurationsfiles

Mit einem Klick auf den Button *Download config.json* kann die Konfigurationsdatei heruntergeladen werden. Wird auf *Create project* geklickt, wird diese Konfigurationsdatei, sowie alle weiteren hochgeladenen Dateien, an das Backend gesendet und das daraus generierte Projekt wird zurückgegeben.

```

1   {
2     "title": "Example JSON Object",
3     "type": "object",
4     "properties": {
5       "string": {
6         "type": "string"
7     },
8       "number": {
9         "type": "number"
10    },
11      "boolean": {
12        "type": "boolean"
13    },
14      "array": {
15        "type": "array",
16        "items": {
17          "type": "string"
18        }
19    },
20      "object": {
21        "type": "object",
22        "properties": {
23          "another string": {
24            "type": "string"
25          },
26          "another number": {
27            "type": "number"
28          }
29        }
30      }
31    }
32  }

```

Folgende Seite wird aufgrund obigem JSON-Schema angezeigt:

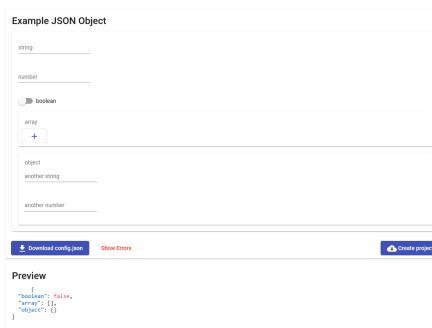


Abbildung 22: Generator

## 4.5 Backend Implementierung

### 4.5.1 Static Site Generators

Da JSON als Format für die Daten und Angular als Framework für die Visualisierung ausgewählt wurde, wurden keine Static Site Generators zum Generieren der Websites verwendet. In Angular können JSON Files zur Laufzeit eingelesen werden, aus denen dann mithilfe von Angular Komponenten diese Daten visualisiert werden können.

### 4.5.2 Aufbau

Das Backend besteht aus zwei Teilen, die verschiedene Aufgaben übernehmen:

- Eine REST-API für das Generieren eines Angular-Projekts aufgrund der im Generator erstellten Konfiguration
- Das Hosten des Generators

#### REST-API

Die API wurde in TypeScript geschrieben und läuft in der Laufzeitumgebung Node.js. Für die Implementierung der Endpoints wurde express.js verwendet.

Das Template-Angular-Projekt, aus dem später mithilfe einer Konfigurationsdatei das Projekt generiert wird, befindet sich im *template* Directory. Aus diesem wird beim Start mithilfe des Node-Packages *node-tar* das Archiv *template.tar* erzeugt.

Die API enthält drei Endpoints:

- GET /api/status
- GET /api/configuration/:filename
- POST /api/configuration

**GET /api/status** Der Endpoint dient zur Überprüfung, ob die API erreichbar ist. Er sendet in diesem Fall *Server is running* als Response.

**GET /api/configuration/:filename** Dieser Endpoint dient zum Download der generierten Projekte. Der Parameter *filename* ist dabei der Name einer *.tar.gz*-Datei, die sich im *public* Directory befindet.

**POST /api/configuration** Ein Request zum diesem Endpoint enthält einen Request-Body des Typs *multipart/form-data*. Darin sind die Konfigurationsdatei für das zu generierende Projekt, sowie alle weiteren benötigten Files enthalten.

Zuerst wird ein temporäres Directory im Pfad für temporäre Dateien des jeweiligen Betriebssystems erstellt. Darin werden alle gesendeten Files gespeichert. Da das Directory mit der Methode *mkdtemp* des Node-Packages *fs* erstellt wird, erhält es einen zufälligen und eindeutigen Namen. Dadurch können mehrere Requests zur gleichen Zeit abgewickelt werden. Danach wird eine Kopie des *template.tar* Archives in diesem Directory erstellt. Zu diesem werden die gesendeten Files hinzugefügt und es wird anschließend mit *gzip* komprimiert. Das entstandene File wird dann in das *public* Directory kopiert und das temporäre Directory wird wieder gelöscht.

Waren alle diese Schritte erfolgreich, wird ein Response mit dem HTTP Status 201 (Created) gesendet. Der Response-Header *Location* enthält dabei die URL zum Download des generierten Projekts. Nach 10 Minuten läuft dieser Downloadlink ab und das generierte Projekt wird gelöscht.

## Webserver

Als Webserver und Reverse-Proxy wird nginx verwendet. Der Webserver stellt alle Files des Generators, die sich im */usr/share/nginx/html/generator* Directory befinden, zur Verfügung, während der Reverse-Proxy alle Requests an die Route */api* zur REST-API weiterleitet. Dieser läuft am Port 4000.

Die Konfigurationsdatei dafür sieht folgendermaßen aus:

```

1   events {}
2
3   http {
4     include mime.types;
5
6     server {
7       location /api {
8         proxy_pass http://server:4000/api/;
9       }
10
11     location / {
12       root /usr/share/nginx/html/generator;
13     }
14   }
15 }
```

### 4.5.3 Containerization

Da beide Teile des Backends verschiedene Aufgaben übernehmen lag es nahe, dieses in der Microservice Architektur umzusetzen. Daher laufen die beiden Teile in unterschiedlichen Docker Containern und werden mithilfe von Docker Compose orchestriert.

Die Images dafür werden in *Dockerfiles* definiert.

Das *docker-compose.yml* File sieht folgendermaßen aus:

```

1  version: '3.8'
2
3  services:
4    nginx:
5      build: .
6      ports:
7        - 80:80
8      restart: always
9      volumes:
10        - ./nginx:/etc/nginx:ro
11
12  server:
13    build: server
14    restart: always
15    environment:
16      NODE_PORT: 4000
17      NODE_ENV: production
18      TEMPLATE_DIR: template
19      PUBLIC_DIR: public
20    volumes:
21      - ./server:/usr/src/app
22      - /usr/src/app/node_modules
23      - /usr/src/app/dist

```

Das gesamte Backend kann somit mit dem Befehl *docker-compose up -d* gestartet werden.

### Webserver Image

Das Image für den Webserver und Reverse Proxy macht sich sogenannte Multi-Stage-Builds zunutze. Dabei werden mehrere Base Images angegeben. Das kann beispielweise genutzt werden, um die Applikation in einem Base Image zu bilden und in einem anderen zu starten.

Im Fall des Webservers wird zuerst das *node* Image verwendet, um den Generator zu komplizieren. Die entstehenden Files werden dann in das *nginx* Image kopiert, wo sie vom Webserver bereitgestellt werden.

Das *Dockerfile* dazu sieht folgendermaßen aus:

```

1  FROM node:16-alpine AS builder
2  WORKDIR /usr/src/app
3  COPY client/package*.json ..
4  RUN npm install
5  COPY client/ ../
6  RUN npm run lint \
7    && npm run build
8
9  FROM nginx:1.21.1-alpine
10 COPY --from=builder /usr/src/app/dist /usr/share/nginx/html

```

## REST-API Image

Im Dockerfile der REST-API werden folgende Schritte ausgeführt:

- Das Base Image *node* wird verwendet
- */usr/src/app* wird als das Directory festgelegt, in dem alle folgenden Befehle ausgeführt werden
- *package.json* und *package-lock.json* werden in das Image kopiert
- Dependencies werden mit *npm install* installiert
- Alle weiteren Files aus dem Directory, in dem sich das *Dockerfile* befindet, werden in das Image kopiert
- Das Projekt wird gelintet und gebaut
- *npm start* wird als Befehl definiert, der beim Start eines Containers ausgeführt wird

Das entsprechende *Dockerfile* sieht wie folgt aus:

```
1 FROM node:16-alpine
2
3 WORKDIR /usr/src/app
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 RUN npm run lint \
12     && npm run build
13
14 CMD [ "npm", "start" ]
```

## 4.6 Deployment Pipeline

Mit der Deployment Pipeline sollen Benutzerinnen und Benutzer die Möglichkeit haben, das generierte Projekt schnell und einfach zu deployen. Das Ziel des Deployments ist dabei entweder eine Linux Virtual Machine oder eine Firebase Hosting Instanz.

### 4.6.1 Jenkins

Um das Projekt mit Jenkins deployen zu können, wurden zwei Files erstellt: *Jenkinsfile.ssh* und *Jenkinsfile.firebaseio*. Je nach Ziel des Deployments konnte eines der beiden beim Erstellen der Pipeline im Web-Interface von Jenkins angegeben werden.

Um die Dateien zur Virtuellen Maschine zu senden, wurde das Jenkins-Plugin *Publish Over SSH* verwendet. Für das Deployen zu *Firebase Hosting* stand kein Plugin zur Verfügung. Stattdessen wurde auf die *Firebase CLI* zurückgegriffen.

Die SSH-Konfiguration der Virtuellen Maschine musste dabei nur einmal in Jenkins vorgenommen werden und konnte dann in jeder Pipeline verwendet werden. Auch für Firebase musste nur einmalig ein Token generiert werden, mit dem sich dann in jeder Pipeline authentifiziert werden konnte.

Ein großer Nachteil von Jenkins war jedoch, dass für jedes Projekt zuerst ein GitHub Repository und eine neue Pipeline im Web-Interface von Jenkins erstellt werden musste, bevor es automatisch deployed werden konnte.

## 4.6.2 GitHub Actions

Für den Ansatz mit GitHub Actions wurden die beiden Files *deploy.yml* und *firebase.yml* im *.github/workflows* Directory des Projekts erstellt.

Sowohl für das Deployen auf eine Virtual Machine, als auch zu *Firebase Hosting* standen Plugins zur Verfügung. Das machte die Konfiguration einfacher als bei Jenkins. Hostname, Username, Target Direcory und SSH Private Key der Virtual Machine und Informationen zur Firebase Instanz wurden als Secrets gespeichert und so in den Workflows verwendet.

Wie auch bei einer Jenkins Pipeline musste für jedes Projekt jedoch auch beim Ansatz mit GitHub Actions ein GitHub Repository erstellt werden.

## 4.6.3 Node.js Skripts

Da für ein Deployment mithilfe von Skripts kein GitHub Repository für jedes Projekt erstellt werden muss, fiel die Entscheidung auf diesen Ansatz. Um das gesamte Projekt leichter wartbar zu machen, wurden auch die Deployment Skripts in JavaScript geschrieben. Diese werden über den Package Manager *npm* mit *npm run deploy:ssh* bzw. *npm run deploy:firebase* aufgerufen. Diese Befehle werden im *package.json* File definiert:

```
1  {
2    "name": "webmap",
3    "version": "0.0.0",
4    "scripts": {
5      ...
6      "deploy:firebase": "node deploy/firebase.js",
7      "deploy:ssh": "node deploy/ssh.js"
8    },
9  }
```

```
9   ...
10 }
```

## Firebase Skript

Das Firebase Skript führt folgende Schritte aus:

- Installieren der Dependencies
- Linten des Projekts
- Builden des Projekts
- Deployen mithilfe der Firebase CLI

Um die jeweiligen Konsolenbefehle in Node.js aufzurufen wurde die Methode *spawn* des Node-Packages *child\_process* verwendet. Dieses wird standardmäßig mit Node.js mitgeliefert und benötigt daher keine zusätzliche Installation.

Bevor zu Firebase deployed werden kann, muss die Benutzerin oder der Benutzer die Project ID des jeweiligen Firebase Projekts eingeben. Dieses wird im *.firebaserc* File gespeichert, das dann von der Firebase CLI ausgelesen wird.

Die Firebase CLI besteht aus einer ausführbaren Datei, die sich im *node\_modules/.bin* Directory befindet. Der Name dieser Datei unterscheidet sich auf Windows jedoch von dem auf anderen Betriebssystemen. Daher musste beim Aufrufen der CLI überprüft werden, auf welchem Betriebssystem das Skript ausgeführt wird. Diese Information stellt Node.js in der Variable *process.platform* bereit.

## SSH Skript

Das SSH Skript unterscheidet sich vom Firebase Skript nur im letzten Schritt. In diesem wird das Node-Package *node-ssh* verwendet, um das gebaute Projekt zu deployen.

Auch hier werden wieder Informationen, die für das Deployment benötigt werden, von Benutzerin oder Benutzer über die Kommandozeile eingelesen. Diese werden im *ssh-config.json* File gespeichert.

# **5 Evaluation des Projektverlaufs**

## **5.1 Meilensteine**

Zu Beginn wurden fünf Meilensteine definiert:

- 12.05.2021 Anforderungen sind definiert
- 14.07.2021 Syntax von Konfigurationsdateien ist definiert
- 14.07.2021 Recherche zu Visualisierungs-Frameworks ist abgeschlossen
- 09.08.2021 Beta-Version fertiggestellt
- 28.03.2022 Projektdokumentation fertiggestellt und übergeben

Es war uns möglich, alle Meilensteine fristgerecht abzuschließen.

## **5.2 Kommunikation**

Um einen reibungslosen Ablauf, welcher für alle beteiligten Personen zufriedenstellend ist zu gewährleisten, wurde wöchentlich ein informelles Meeting mit Christopher Stelzmüller abgehalten. Darin wurden Entscheidungen sowie Pläne für die weitere Vorangehensweise besprochen und aufeinander abgestimmt. Zusätzlich dazu wurde der Betreuungslehrer jeden Freitag über den aktuellen Fortschritt schriftlich informiert.

## **5.3 Probleme während der Implementierung**

Dank der guten Betreuung durch triply und der bereits großen Erfahrung in den ausgewählten Technologien seitens der Entwickler sind während der Implementierung keine Probleme aufgetreten.

# Abbildungsverzeichnis

|    |  |    |
|----|--|----|
| 1  | git workflow [?]   | 4  |
| 2  | wöchentliche Downloads [?]   | 9  |
| 3  | Direkter Vergleich der Frameworks [?]  | 9  |
| 4  | Vue.js Core Repository Commits im Zeitraum 16. September 2018 - 9. Februar 2022 [?]                      | 11 |
| 5  | Ivy Pipeline   | 14 |
| 6  | Skakable Tree  | 15 |
| 7  | Triply Slider und Triply Switch  | 18 |
| 8  | Model-View-Controller-Pattern  | 19 |
| 9  | Österreich mit Bundesländern in Mapbox   | 20 |
| 10 | Österreich mit Bundesländern in Leaflet  | 21 |
| 11 | Österreich mit Bundesländern in Google Maps  | 22 |
| 12 | Repository secrets   | 28 |
| 13 | Virtualisierung mit Virtuellen Maschinen [?]   | 36 |
| 14 | Virtualisierung mit Containern [?]   | 36 |
| 15 | Monolith vs. Microservices [?]   | 38 |
| 16 | webmap Workflow  | 40 |
| 17 | Frontend Architektur   | 45 |
| 18 | Menü zur Page Auswahl  | 47 |
| 19 | Color Scheme Beispiel: Je heller die Farbe, desto länger dauert die Anreise mit dem Fahrrad in die Mitte | 48 |
| 20 | Sidebar mit Beispiel Daten   | 49 |
| 21 | Webmap Frontend  | 49 |
| 22 | Generator  | 53 |

# **Tabellenverzeichnis**

# Quellcodeverzeichnis

|                               |    |
|-------------------------------|----|
| hello-world.js . . . . .      | 12 |
| app.js . . . . .              | 13 |
| app.ts . . . . .              | 13 |
| style.scss . . . . .          | 17 |
| Jenkinsfile . . . . .         | 27 |
| deploy.yml . . . . .          | 28 |
| secrets.yml . . . . .         | 28 |
| example.json . . . . .        | 29 |
| example-schema.json . . . . . | 31 |
| example.xml . . . . .         | 32 |
| schema.xml . . . . .          | 32 |
| example.conf . . . . .        | 35 |
| Dockerfile . . . . .          | 37 |
| config.json . . . . .         | 42 |
| schema.json . . . . .         | 53 |
| nginx.conf . . . . .          | 55 |
| docker-compose.yml . . . . .  | 56 |
| Dockerfile.nginx . . . . .    | 56 |
| Dockerfile.server . . . . .   | 57 |
| package.json . . . . .        | 58 |

# **Anhang**