

# Entwicklung von Web-Front-Ends mit Vue.js

Studienarbeit T3\_3101

des Studienganges Angewandte Informatik an der  
Dualen Hochschule Baden-Württemberg Mosbach



von

Lars Rickert

Matrikelnummer, Kurs 2858031, INF19B

Gutachter der Dualen Hochschule Philipp Abele

27. April 2022

### Eigenständigkeitserklärung

*Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema:  
„Entwicklung von Web-Front-Ends mit Vue.js“  
selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.  
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.*

---

Ort, Datum

Unterschrift

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>I</b>
<b>Abkürzungsverzeichnis</b>	<b>II</b>
<b>1. Einleitung</b>	<b>1</b>
<b>2. Technologien</b>	<b>2</b>
2.1. TypeScript . . . . .	2
2.2. Sass . . . . .	3
2.3. Bootstrap . . . . .	3
2.4. Vue . . . . .	4
2.4.1. Grundlagen von Komponenten . . . . .	5
2.4.2. Composition API . . . . .	7
2.4.3. Reaktivität . . . . .	7
2.5. Strapi CMS . . . . .	8
<b>3. Projektstruktur</b>	<b>9</b>
3.1. Frontend . . . . .	9
3.1.1. Startseite . . . . .	9
3.1.2. Landing page . . . . .	11
3.1.3. Blog . . . . .	13
3.1.4. Blogbeitrag . . . . .	14
3.1.5. Code-Struktur . . . . .	16
3.1.6. Lokale Entwicklung . . . . .	16
3.2. Backend . . . . .	17

## Inhaltsverzeichnis

<b>4. Deployment</b>	<b>19</b>
4.1. Bereitstellen des Linux Servers und der Domain . . . . .	19
4.1.1. Installation von Ubuntu . . . . .	19
4.1.2. Zugriff auf den VPS / Einrichtung eines SSH-Clients . . . . .	20
4.1.3. Konfiguration der Domain / DNS . . . . .	21
4.2. Einrichtung des nginx-proxy . . . . .	22
4.2.1. Installation von Docker und docker-compose . . . . .	22
4.2.2. Installation des nginx-proxy . . . . .	22
4.3. Starten des Front- und Backends . . . . .	23
4.4. Ersteinrichtung des Backends . . . . .	24
4.5. Datensicherung und Umzug auf einen anderen Server . . . . .	24
<b>5. Fazit</b>	<b>25</b>

## Literaturverzeichnis

### I. Datenträger CD

# Abbildungsverzeichnis

2.1. Beispiel für TypeScript-Code . . . . .	2
2.2. Erweiterung von JavaScript-Code um TypeScript Typ-Überprüfung . . . .	3
2.3. Vergleich von CSS und Sass . . . . .	4
2.4. Vue-Komponenten . . . . .	5
2.5. Beispiel einer Vue-Komponente . . . . .	6
2.6. Verwendung einer Vue-Komponente . . . . .	6
2.7. Beispiel einer Vue-Komponente mit Options API . . . . .	7
3.1. Startseite des Frontends Teil 1 . . . . .	10
3.2. Startseite des Frontends Teil 2 . . . . .	11
3.3. Landing page des Frontends Teil 1 . . . . .	12
3.4. Landing page des Frontends Teil 2 . . . . .	13
3.5. Blog des Frontends . . . . .	14
3.6. Blogbeitrag des Frontends . . . . .	15
3.7. Dashboard des Backends . . . . .	17
3.8. Inhaltstyp für die Blogbeiträge . . . . .	18
4.1. Installation von Ubuntu auf dem VPS . . . . .	20
4.2. Einrichtung von VS Code als SSH-Client . . . . .	20
4.3. Einrichtung der Domain . . . . .	21

# Abkürzungsverzeichnis

**SFC** Single-File Component

**CMS** Content Management System

**VPS** Virtual Private Server

**VS Code** Visual Studio Code

# 1. Einleitung

Die Anzahl der pflegebedürftigen Menschen in Deutschland ist von 1999 bis 2019 stetig gestiegen. Waren es 1999 noch ca. 2 Millionen Pflegebedürftige, so hat sich die Zahl der Pflegebedürftigen Ende 2019 auf ca. 4,13 Millionen mehr als verdoppelt [vgl. 1].

Für diese Menge an Pflegebedürftigen steht in Pflegeheimen im Jahre 2019 ein Personal von ca. 796.000 Pflegekräften zur Verfügung [vgl. 2]. Daraus ergibt sich eine Quote von ca. 5 Pflegebedürftigen pro Pflegekraft.

Ehrenamtliche Helfer:innen können demnach die Pflegeheime unterstützen und entlasten. Dabei werden teilweise sogenannte Aktivierungen durchgeführt. Diese sind ca. 45 bis 60 minütige Aktivitäten, die die ehrenamtlichen Helfer:innen oder die Pflegekräfte mit den Pflegebedürftigen unternehmen. Hierbei kann es sich z.B. um Spiele, Vorlesungen oder ähnliches handeln. Aktivierungen müssen häufig mehrere Stunden vorbereitet und geplant werden. Sollte eine Pflegekraft eine Aktivierungen vorbereiten, so geschieht dies in der Regel außerhalb der regulären Arbeitszeit.

Ziel dieser Studienarbeit ist es, eine Web-Anwendung bzw. Blog zu erstellen, auf der ehrenamtliche Helfer:innen und Pflegekräfte Wissen über Aktivierungen austauschen und bereits geplante Aktivierungen vorstellen können. So soll der Zeitaufwand für durchzuführende Aktivierungen verringert werden.

Um dieses Ziel zu erreichen werden im Laufe der Studienarbeit zwei Anwendungen erstellt. Die erste Anwendung ist die Web-Anwendung bzw. der Blog (Frontend), die die Informationen über die Aktivierungen (Blogbeiträge) anzeigt. Als zweites wird eine Anwendung zum Verwalten der Blogbeiträge (Backend) entwickelt, dass u.a. das Erstellen von neuen Beiträgen vereinfachen soll.

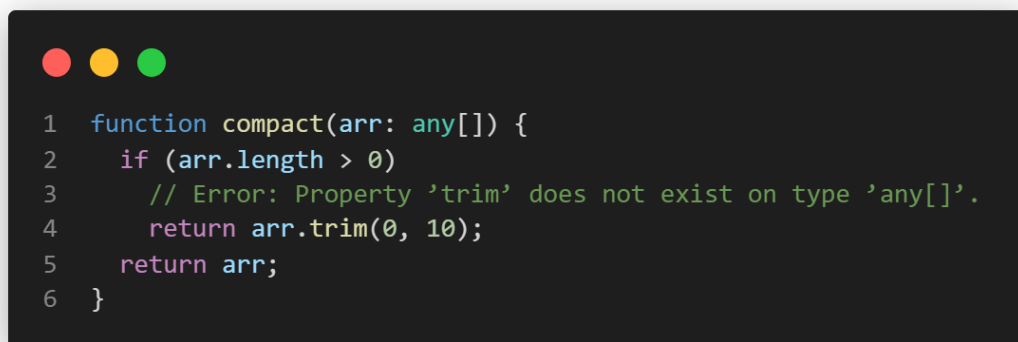
Es werden Grundkenntnisse von HTML, CSS, JavaScript, Node.js bzw. npm, Docker bzw. docker-compose und Bash vorausgesetzt.

## 2. Technologien

### 2.1. TypeScript

TypeScript ist eine Programmiersprache von Microsoft, die JavaScript um ein Typensystem erweitert. So können u.A. Fehler frühzeitig (während der Entwicklung) erkannt und behoben sowie Autovervollständigungen geboten werden. TypeScript-Code kann nicht direkt ausgeführt werden, sondern muss vorher zu JavaScript transpiliert werden. Jeder gültige JavaScript-Code ist dadurch auch gültiger TypeScript-Code [vgl. 3; vgl. 4].

Abbildung 2.1 zeigt einen TypeScript-Code, bei dem der Parameter der Funktion als Array typisiert wurde. Wird nun z.B. eine ungültige Methode, wie `trim` aufgerufen, gibt TypeScript eine Fehlermeldung aus, da Arrays diese Methode nicht besitzen.

A screenshot of a code editor with a dark background. At the top left, there are three colored circles: red, yellow, and green. The code is as follows:

```
1 function compact(arr: any[]) {  
2   if (arr.length > 0)  
3     // Error: Property 'trim' does not exist on type 'any[]'.  
4     return arr.trim(0, 10);  
5   return arr;  
6 }
```

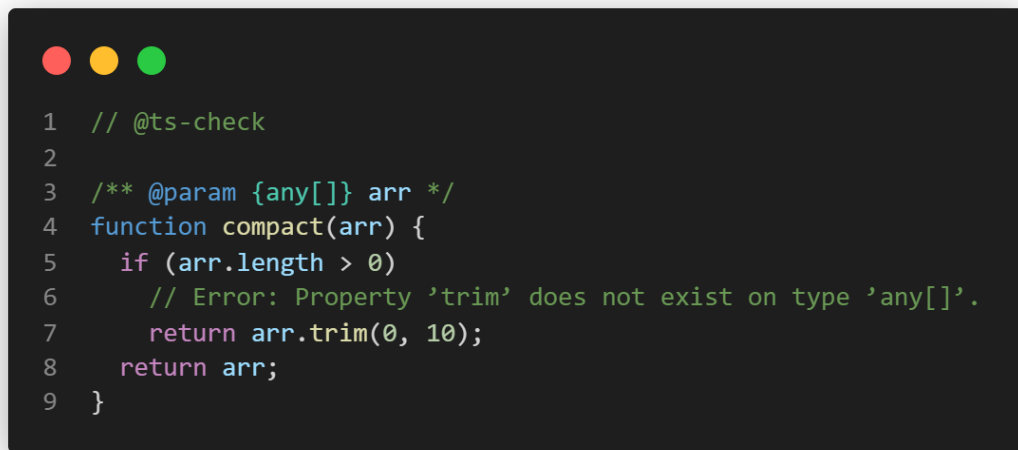
Abbildung 2.1.: Beispiel für TypeScript-Code

Bestehende JavaScript-Projekte können inkrementell auf TypeScript umgestellt werden, indem TypeScript vorerst nur als Typ-Überprüfung verwendet wird. Dabei können Kommentare in den vorhandenen JavaScript-Code eingefügt werden, wodurch TypeScript anschließend den Code überprüfen kann (siehe Abbildung 2.2) [vgl. 3]. Da hierbei anders als



## 2.2. SASS

in Abbildung 2.1 kein TypeScript-Code eingefügt wird, muss der Code nicht transpiliert werden, bietet jedoch auch nicht den vollen Funktionsumfang von TypeScript.

A screenshot of a code editor with a dark background and light-colored text. The code is written in TypeScript and includes a JSDoc-style comment and a function definition. A green error message is visible on line 6, indicating that the 'trim' property does not exist on the 'any' type. The code is as follows:

```
1 // @ts-check
2
3 /** @param {any[]} arr */
4 function compact(arr) {
5   if (arr.length > 0)
6     // Error: Property 'trim' does not exist on type 'any[]'.
7     return arr.trim(0, 10);
8   return arr;
9 }
```

Abbildung 2.2.: Erweiterung von JavaScript-Code um TypeScript Typ-Überprüfung

## 2.2. Sass

Sass ist eine Stylesheet-Sprache, die zu CSS kompiliert wird. Sie vereinfacht die Verwendung von CSS, indem weitere Funktionen, wie z.B. Verschachtlung, bereitgestellt werden. Dabei ist sie vollständig zu CSS kompatibel, d.h. jeder CSS-Code ist gültiger Sass-Code [vgl. 5]. Die erweiterten Funktionen helfen dabei robusten und wartbaren CSS-Code zu schreiben [vgl. 5].

Abbildung 2.3 zeigt einen Vergleich desselben Codes in CSS und Sass. Durch Sass kann hierbei redundanter Code verhindert werden. Für Interessierte sei für weitere Funktionen auf [6] verwiesen.

## 2.3. Bootstrap

Um die Gestaltung der Benutzeroberfläche (siehe Kapitel 3.1) zu vereinfachen, wird Bootstrap verwendet. Bootstrap ist Werkzeugsatz, der u.A. vorgefertigte Komponenten bzw. CSS sowie Icons bereitstellt [vgl. 7].

## 2.4. VUE



Abbildung 2.3.: Vergleich von CSS (links) und Sass (rechts) am Beispiel von Verschachtelung [6]

## 2.4. Vue

Für die Erstellung der Benutzeroberfläche (im Folgenden „Frontend“ genannt) wird Vue (Version 3.2) verwendet. Vue ist ein progressives JavaScript-Framework, das die Frontend-Entwicklung vereinfachen soll. Progressiv bedeutet hierbei, dass es für das gesamte Frontend oder auch nur für Teile desselben verwendet werden kann. Vue kann folglich mit anderen Technologien verwendet oder Schritt-für-Schritt (progressiv) in bereits vorhandene Projekte integriert werden [vgl. 8].

## 2.4. VUE

### 2.4.1. Grundlagen von Komponenten

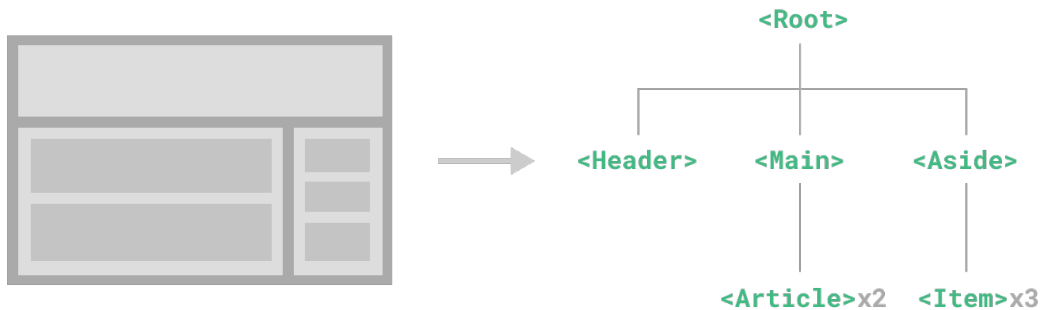


Abbildung 2.4.: Strukturierung des Frontends durch Vue-Komponenten [9]

Vue-Komponenten erlauben die Unterteilung des Frontends in unabhängige, isolierte und wiederverwendbare Teile. So wird das Frontend in kleinere Komponenten gegliedert (siehe Abbildung 2.4), die dann die benötigte Logik kapseln und unabhängig entwickelt werden können [vgl. 9].

Vue bietet mehrere Möglichkeiten, um eine Komponente zu definieren. Im Rahmen dieser Studienarbeit werden sogenannte Single-File Component (SFC) verwendet (siehe Abbildung 2.5). Jede Komponente wird hierbei in einer Datei mit der Dateierweiterung `.vue` definiert. Nachdem die Komponente in einer anderen Datei importiert wurde, kann sie wie ein herkömmliches HTML-Element verwendet werden (siehe Abbildung 2.6) [vgl. 9].

#### Sprach-Blöcke

In SFCs können standardmäßig die drei Sprach-Blöcke `<template>` (HTML), `<script>` (JavaScript) und `<style>` (CSS) verwendet werden (siehe Abbildung 2.5), die den entsprechenden Code der Komponente beinhalten. Mit dem `scoped`-Attribut des `<style>`-Blocks kann das CSS für die Komponente gekapselt werden, d.h. es wird nur innerhalb der Komponente angewendet [vgl. 10].

#### TypeScript und Sass

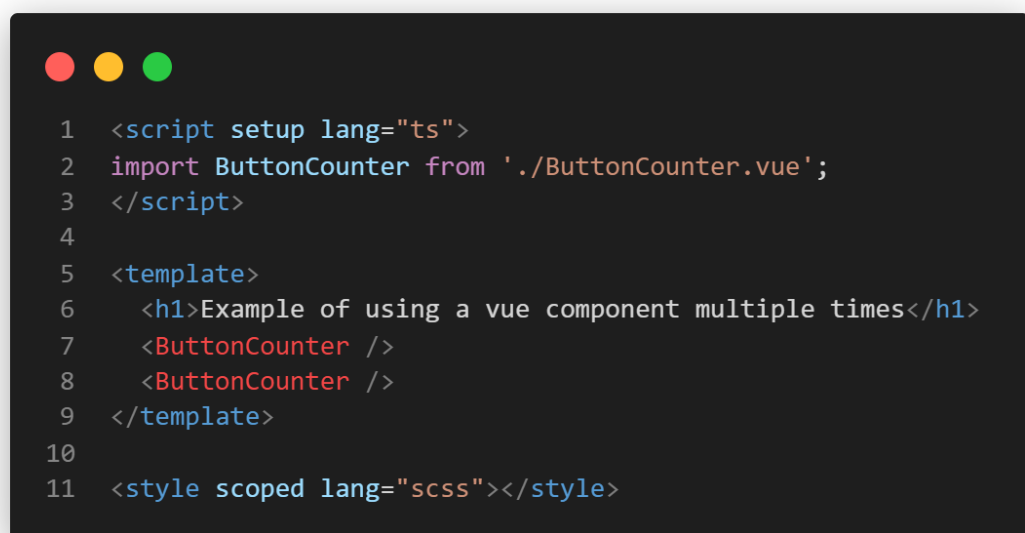
Für die Entwicklung des Frontends sollen TypeScript (Version 4.4) und Sass (Version 1.43) verwendet werden, die bereits in Kapitel 2.1 und 2.2 erläutert wurden. So soll sichergestellt werden, dass Fehler bereits während der Entwicklung behoben werden können und ein

## 2.4. VUE

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is a Vue Single File Component (SFC) in TypeScript. It consists of 11 lines: 1. `<script setup lang="ts">`, 2. `import { ref } from 'vue';`, 3. (empty line), 4. `const count = ref(0);`, 5. `</script>`, 6. (empty line), 7. `<template>`, 8. `<button @click="count++">You clicked me {{ count }} times.</button>`, 9. `</template>`, 10. (empty line), 11. `<style scoped lang="scss"></style>`.

```
1 <script setup lang="ts">
2 import { ref } from 'vue';
3
4 const count = ref(0);
5 </script>
6
7 <template>
8   <button @click="count++">You clicked me {{ count }} times.</button>
9 </template>
10
11 <style scoped lang="scss"></style>
```

Abbildung 2.5.: Beispiel einer Vue SFC [9]

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code shows how to use a custom Vue component named 'ButtonCounter'. It consists of 11 lines: 1. `<script setup lang="ts">`, 2. `import ButtonCounter from './ButtonCounter.vue';`, 3. `</script>`, 4. (empty line), 5. `<template>`, 6. `<h1>Example of using a vue component multiple times</h1>`, 7. `<ButtonCounter />`, 8. `<ButtonCounter />`, 9. `</template>`, 10. (empty line), 11. `<style scoped lang="scss"></style>`.

```
1 <script setup lang="ts">
2 import ButtonCounter from './ButtonCounter.vue';
3 </script>
4
5 <template>
6   <h1>Example of using a vue component multiple times</h1>
7   <ButtonCounter />
8   <ButtonCounter />
9 </template>
10
11 <style scoped lang="scss"></style>
```

Abbildung 2.6.: Verwendung einer Vue SFC [vgl. 9; vgl. 10]

robustes, wartbares Frontend entsteht.

Um TypeScript in einer SFC verwenden zu können, muss der `<script>`-Block (siehe Abbildung 2.5) um das Attribut `lang="ts"` erweitert werden. Sass kann nach der Erweiterung des `<style>`-Blocks um das Attribut `lang="scss"` verwendet werden [vgl. 10].

## 2.4. VUE

### 2.4.2. Composition API

Vue-Komponenten können (innerhalb des `<script>`-Blocks) entweder mit Vue's Options API oder mit der Composition API erstellt werden. Aufgrund der Empfehlung von Vue wird im Rahmen dieser Studienarbeit die Composition API in Verbindung mit SFCs verwendet. Des Weiteren wird das `setup`-Attribut des `<script>`-Blocks ergänzt (siehe Abbildung 2.5), wodurch der Umgang mit der Composition API erleichtert wird. Unter anderem werden alle Imports sowie Variablen für den `<template>`-Block bereitgestellt [vgl. 8].

In Abbildung 2.5 wird bereits die Composition API verwendet. Im Vergleich dazu zeigt Abbildung 2.7 dieselbe Komponente unter Verwendung der Options API.



```
1  <script lang="ts">
2  import { defineComponent, ref } from 'vue';
3
4  export default defineComponent({
5    data() {
6      return {
7        count: 0,
8      };
9    },
10  });
11 </script>
12
13 <template>
14   <button @click="count++">You clicked me {{ count }} times.</button>
15 </template>
16
17 <style scoped lang="scss"></style>
```

Abbildung 2.7.: Beispiel einer Vue-Komponente mit Options API [vgl. 8; vgl. 10]

### 2.4.3. Reaktivität

Vue stellt mehrere Funktionen zum Erstellen von reaktiven Variablen bereit. In Abbildung 2.5 wurde bereits die Funktion `ref` verwendet, die einen Wert reaktiv macht [vgl. 11]. Wird ein reaktiver Wert z.B. im `<template>`-Block der Komponente verwendet, aktualisiert Vue

## 2.5. STRAPI CMS

automatisch den HTML-Code, wenn sich der Wert ändert [vgl. 12]. Würde man also in Abbildung 2.5 „`const count = ref(0)`“ durch „`const count = 0`“ ersetzen, würde im HTML-Code selbst bei einer Änderungen des Wertes immer „`You clicked me 0 times.`“ angezeigt werden, da `count` dann nicht mehr reaktiv ist.

Für weitere Funktionen bezüglich der Reaktivität sei für Interessierte auf [11] verwiesen. Für detaillierte Informationen über Vue empfiehlt sich die offizielle Dokumentation, die unter [8] zu finden ist.

## 2.5. Strapi CMS

Strapi ist ein sogenanntes headless Content Management System (CMS), d.h. es ist eine Software zum Verwalten von Inhalten und Daten. Headless bedeutet hierbei, dass das CMS ausschließlich die Verwaltung (Erstellung, Speicherung etc.) der Inhalte übernimmt und sie nicht in z.B. einer Webanwendung oder App darstellt. Traditionelle CMS, wie z.B. WordPress, bündeln im Vergleich dazu die Verwaltung der Inhalte und die Darstellung in derselben Anwendung [vgl. 13].

Strapi ist in JavaScript geschrieben, Open Source und kann vollständig angepasst werden. Es soll die Entwicklung von APIs deutlich vereinfachen und kann selber gehostet werden. Dazu wird eine Benutzeroberfläche bereitgestellt, über die Inhaltstypen erstellt und verwaltet werden können. Anschließend sind diese durch REST oder GraphQL Endpunkte zugänglich [vgl. 14].

Im Rahmen dieser Studienarbeit wird Strapi (Version 4.x) als API (im Folgenden „Backend“ genannt) verwendet, um Blogbeiträge zu verwalten. Eine detaillierte Betrachtung des CMS erfolgt in Kapitel 3.2.

## 3. Projektstruktur

### 3.1. Frontend

Das Frontend besteht aus insgesamt vier Seiten, die im Folgenden näher betrachtet werden sollen. Dabei wird lediglich auf die Seiten als Ganzes und nicht auf jede individuelle Komponente eingegangen. Es sei angemerkt, dass im aktuellen Entwicklungsstand der Studienarbeit Platzhalter für Texte, Bilder etc. verwendet werden, da konkrete Inhalte noch nicht feststehen.

#### 3.1.1. Startseite

Abbildung 3.1 und 3.2 zeigen die Startseite des Frontends. Am Anfang der Seite befindet sich ein großflächiges Bild, das die Aufmerksamkeit des Nutzers anziehen soll. Im Folgenden werden dann die Angebote beschrieben, die die Pflegeeinrichtung anbietet sowie Zitate von Erfahrungsberichten dargestellt. Am Ende der Seite befinden sich Bilder und Daten zum Team und Partner der Pflegeeinrichtung, wodurch eine persönliche Bindung zum Besucher hergestellt werden soll.

### 3.1. FRONTEND

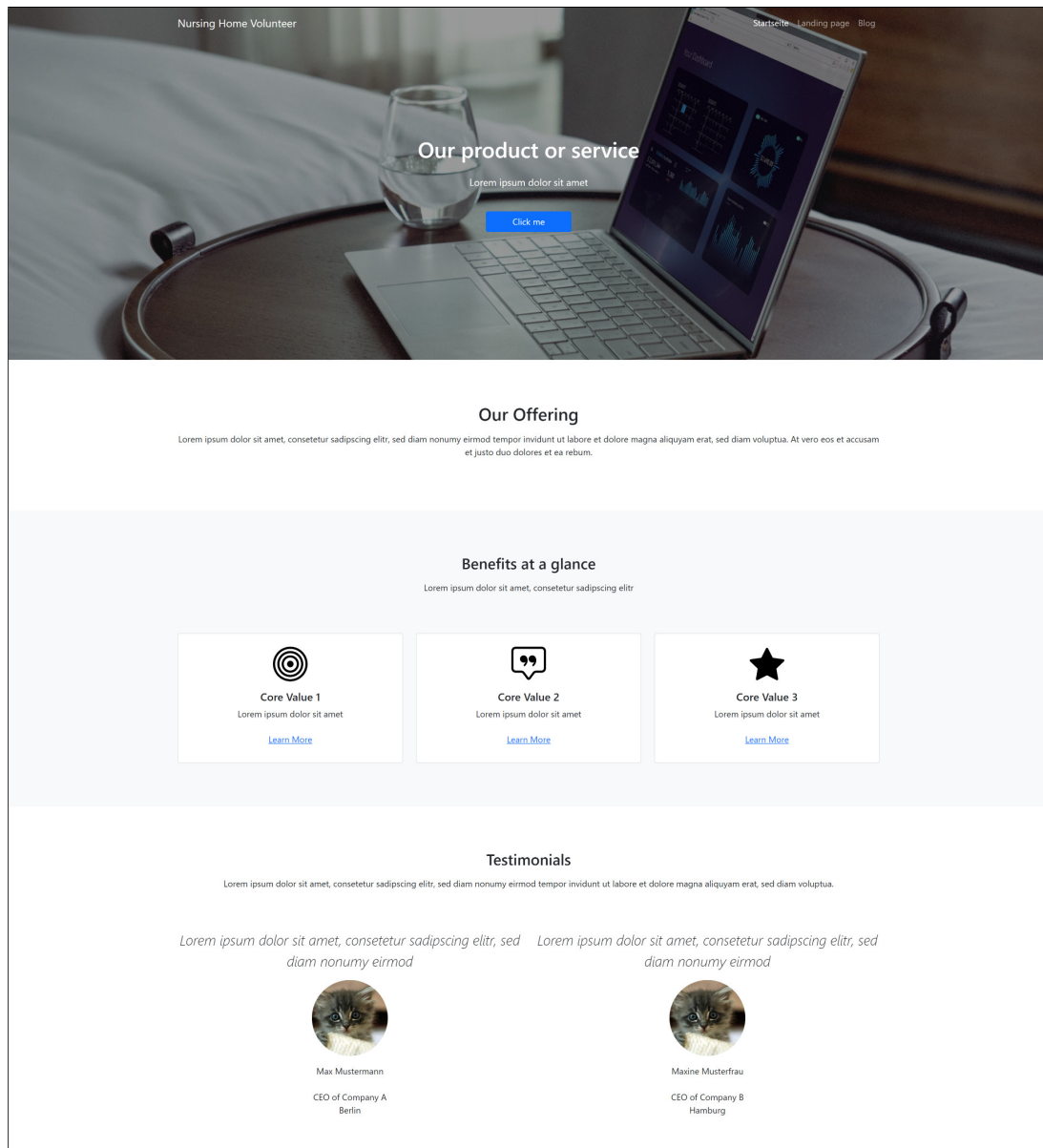


Abbildung 3.1.: Startseite des Frontends Teil 1



## 3.1. FRONTEND

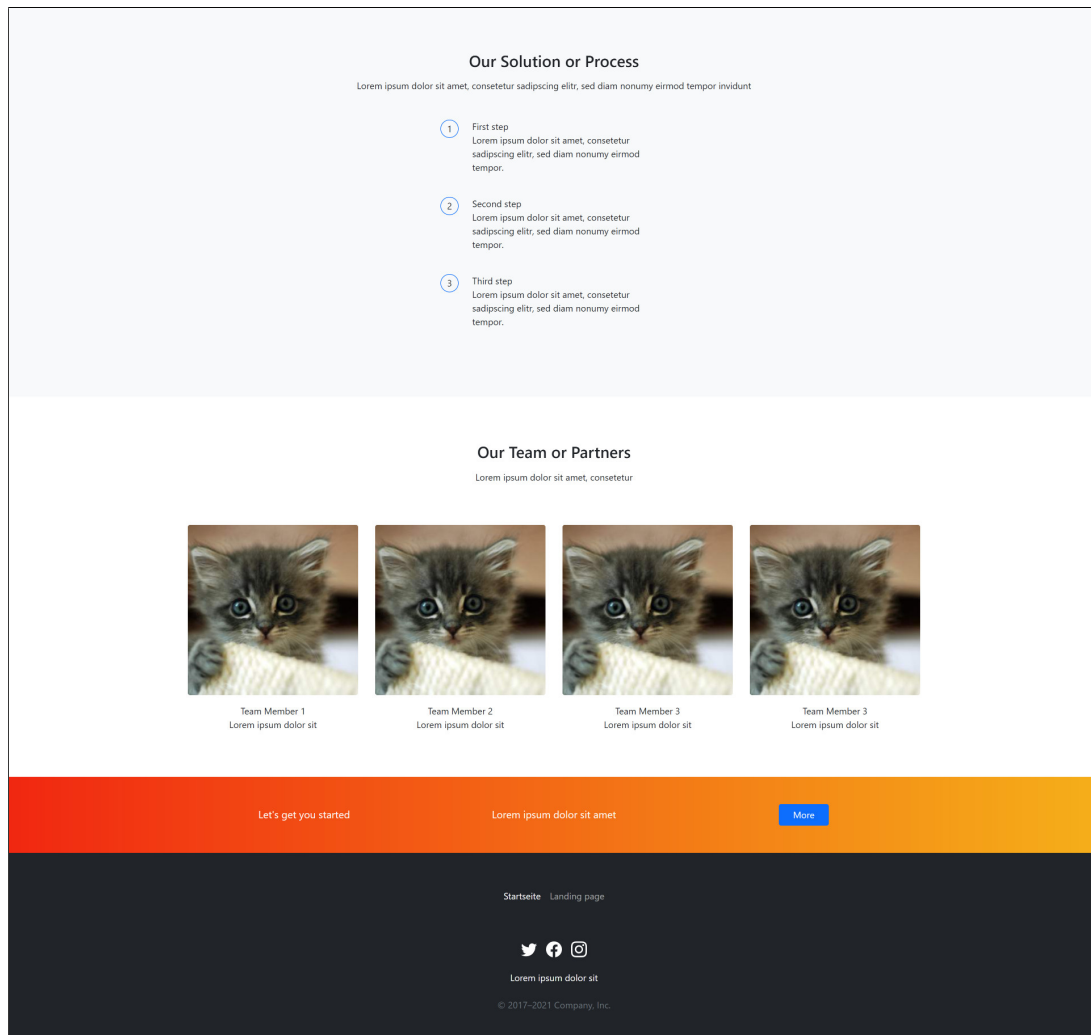


Abbildung 3.2.: Startseite des Frontends Teil 2

### 3.1.2. Landing page

Die zweite Seite des Frontends ist die sogenannte landing page (siehe Abbildung 3.3 und 3.4), die über die Route `/landing-page` erreichbar ist. Diese Seite soll kurz und knapp die wichtigsten Informationen über das Angebot der Pflegeeinrichtung darstellen. Die landing page kann z.B. an Interessierte weitergeleitet werden, um diesen einen vollständigen und kompakten Überblick über alle Angebote zu geben. Alternativ kann diese Seite als Austausch für die Startseite verwendet werden, sollte dieser Wunsch seitens der Pflegeeinrichtung bestehen.

### 3.1. FRONTEND

Hervorzuheben ist der interaktive Kopfbereich der Seite, indem der Besucher durch drei verschiedene Ansichten wechseln kann. In Abbildung 3.3 ist der Hintergrund zwar lediglich grau, jedoch kann für jede Ansicht ein individuelles Hintergrundbild eingefügt werden, um die Seite ansprechender zu gestalten.

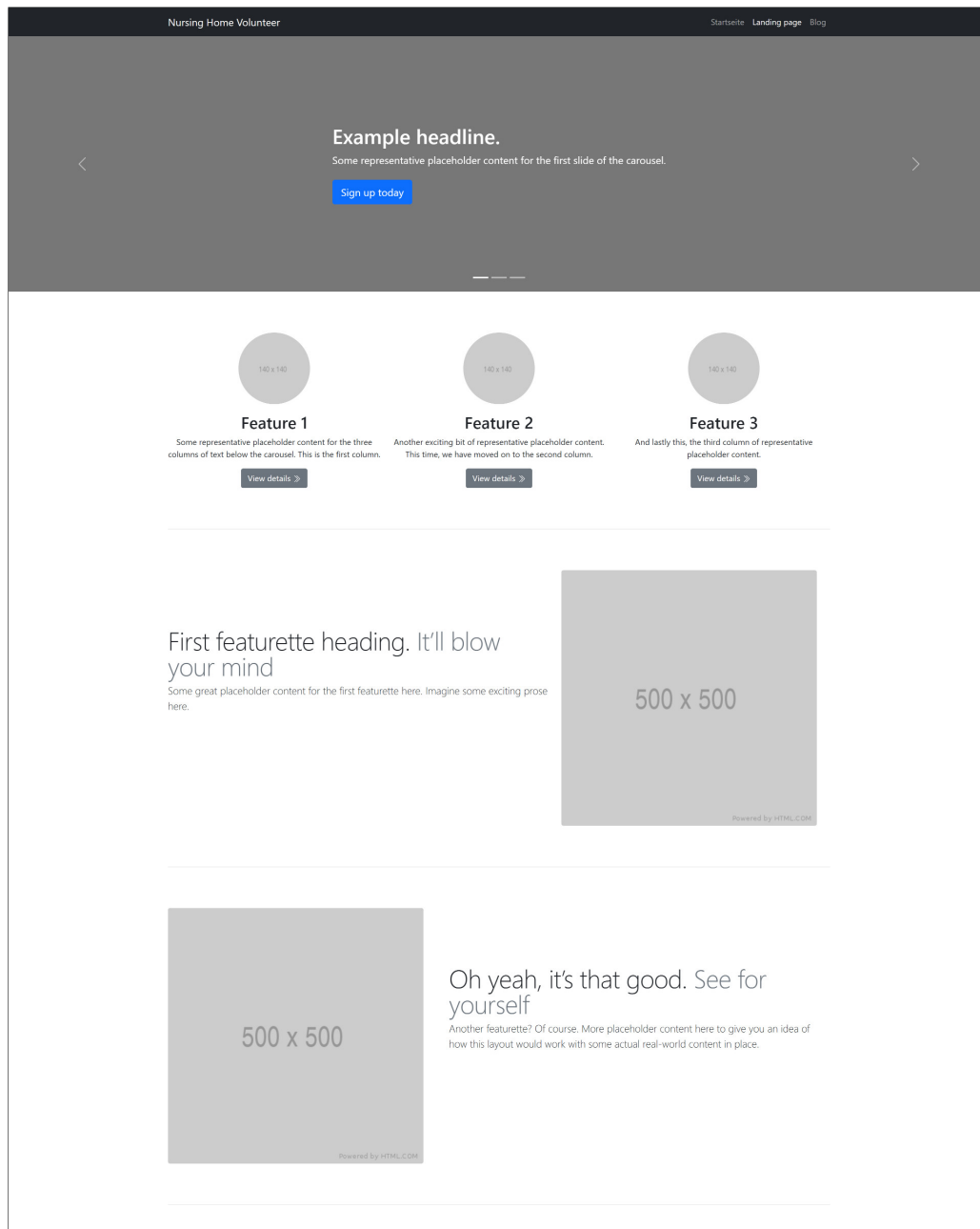


Abbildung 3.3.: Landing page des Frontends Teil 1

## 3.1. FRONTEND

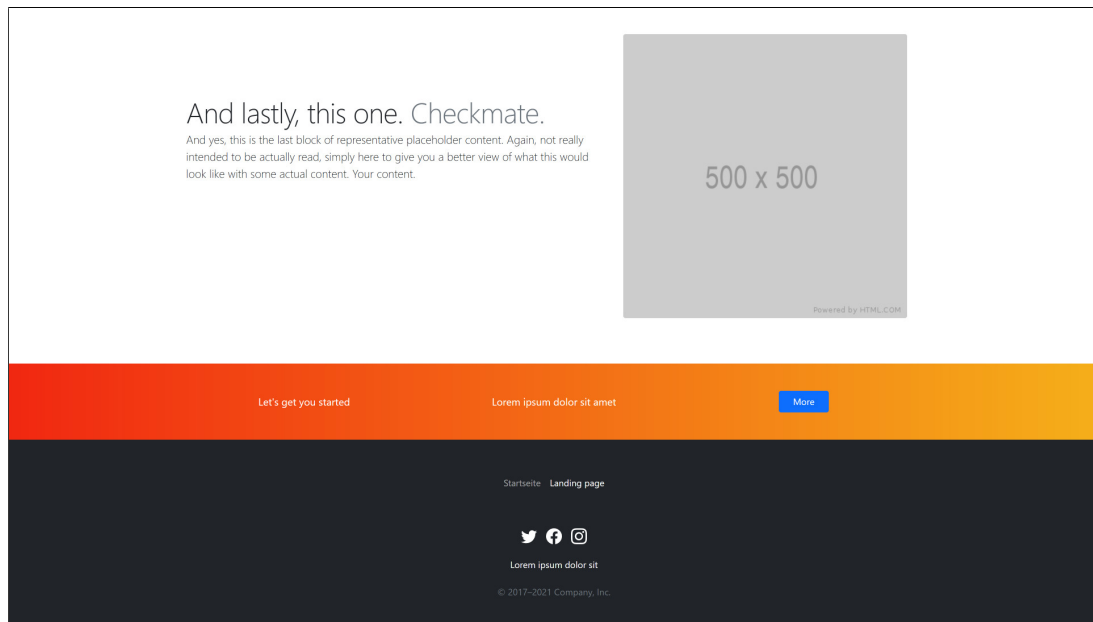


Abbildung 3.4.: Landing page des Frontends Teil 2

### 3.1.3. Blog

Der Blog ist die zentrale Stelle für die Ansicht aller Beiträge der Pflegeeinrichtung und ist über die Route `/blog` erreichbar. Dabei werden alle verfügbaren Blogbeiträge vom Backend abgerufen und eine Vorschau dieser angezeigt. Ein Klick auf das Bild oder auf die „Weiterlesen“-Schaltfläche führt auf die Detailseite des Blogbeitrags.

Links neben den Blogbeiträgen befindet sich ein Platzhalter für Filtermöglichkeiten. Zum aktuellen Entwicklungsstand der Studienarbeit sind die Anforderungen für die Filter seitens der Pflegeeinrichtung noch nicht bekannt. Daher sind diese nicht implementiert.

## 3.1. FRONTEND

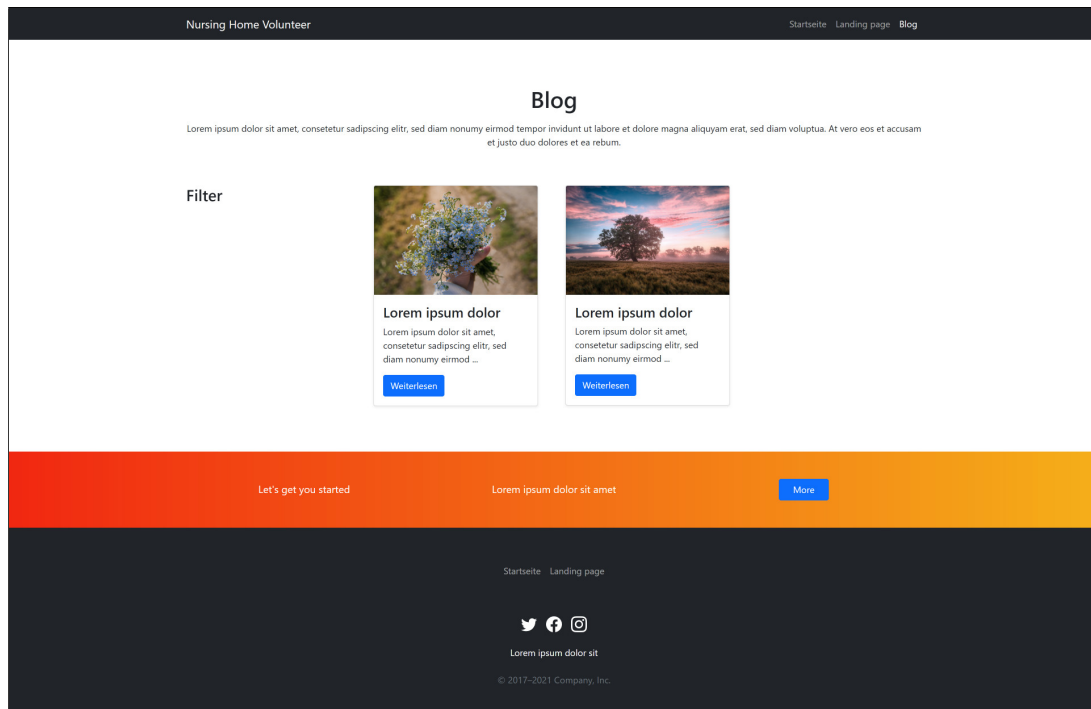


Abbildung 3.5.: Blog des Frontends

### 3.1.4. Blogbeitrag

Nach dem Öffnen eines Beitrags im Blog oder durch manuellen Aufruf der URL, gelangt der Besucher auf die Detailseite für einen Blogbeitrag (siehe Abbildung 3.6). Jeder Beitrag ist über die Route `/blog/<id>` erreichbar, wobei `<id>` durch die eindeutige ID des Beitrags ersetzt werden muss. Beim Aufruf der Seite wird der Blogbeitrag vom Backend abgerufen, falls dieser nicht bereits geladen wurde.

Ein Blogbeitrag besteht aus insgesamt vier Komponenten: Titel, Bild (optional), Beschreibung und beliebig vielen Medienbereichen.

Die Medienbereiche sind, wie in Abbildung 3.6 zu sehen, voneinander getrennte Abschnitte, die mindestens eine Beschreibung bzw. Text beinhalten. Der Titel sowie das Medium sind optional. Diese Bereiche können dazu verwendet werden, um den Blogbeitrag inhaltlich zu gliedern sowie mit weiteren Medien zu erweitern. Mögliche Medientypen sind Bilder, Videos, Audiodateien sowie sonstige Dateien (PDFs, Textdateien etc.).

## 3.1. FRONTEND

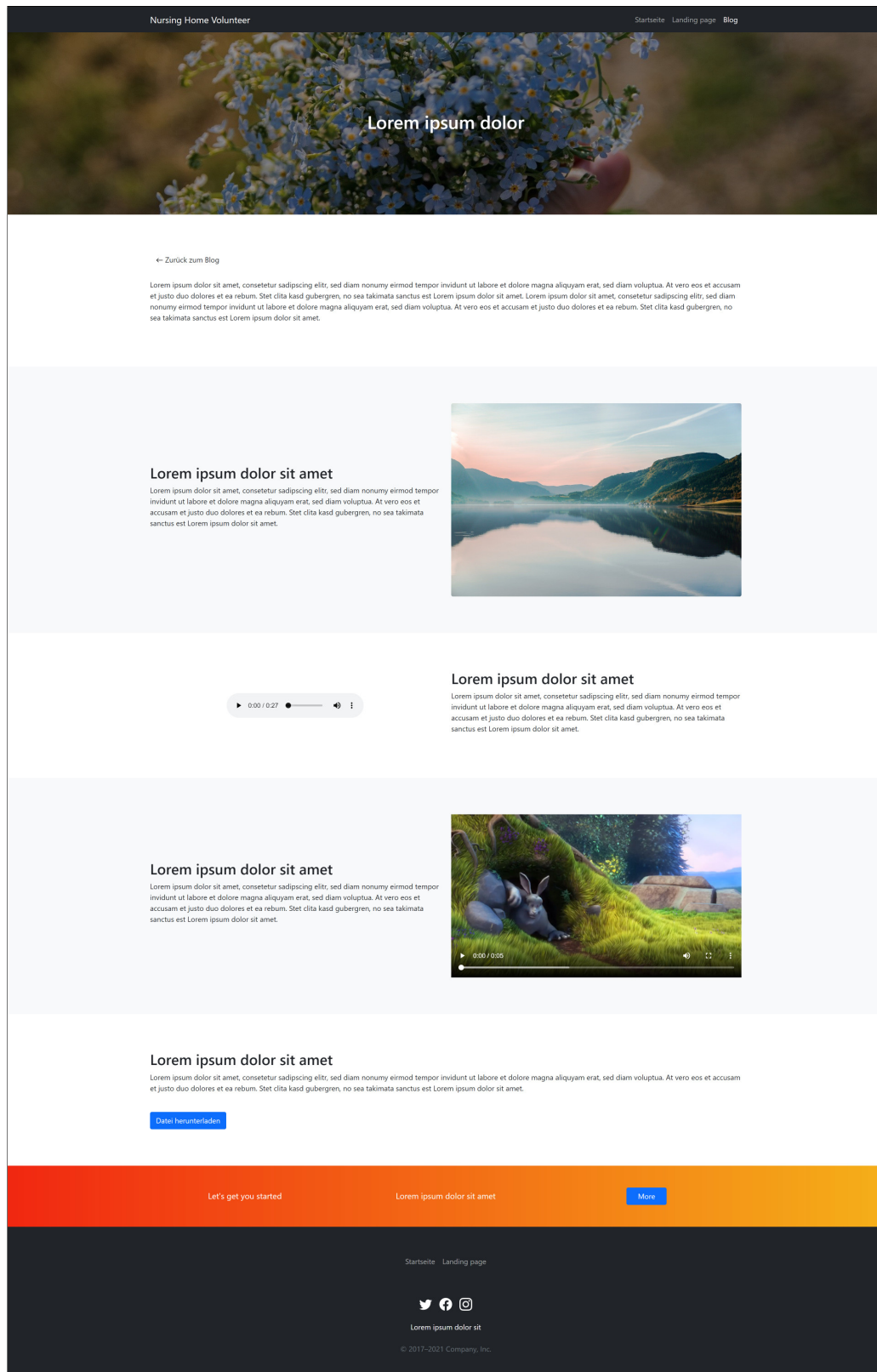


Abbildung 3.6.: Blogbeitrag des Frontends

## 3.1. FRONTEND

### 3.1.5. Code-Struktur

Tabelle 3.1 zeigt die hierarchische Code-Struktur des Frontends sowie die Funktionen der wesentlichen Verzeichnisse und Dateien.

Ordner / Datei	Beschreibung
assets	Bilder/Dateien, die im Frontend verwendet werden (außer Favicons)
axios	REST-Client zur Interaktion mit dem Backend
components	Vue-Komponenten (SFCs)
bootstrap	Komponenten, die mithilfe von Bootstrap erstellt wurden
layouts	Layout- bzw. Struktur-Komponenten (z.B. Header, Hauptinhalt, Footer)
config	Statische Konfiguration des Frontends (z.B. URL des Backends)
router	Vue Router zur Navigation zwischen den Seiten
store	Globaler Zustand / Daten (z.B. Blogbeiträge)
styles	Globales CSS / Sass
types	TypeScript Typ-Definitionen
utils	Globale (wiederverwendbare) Funktionen
views	Vue-Komponenten für die Seiten
App.vue	App-Komponente (Eintrittspunkt / Hauptkomponente des Frontends)

Tabelle 3.1.: Beschreibung der wesentlichen Verzeichnisse und Dateien des Frontends

### 3.1.6. Lokale Entwicklung

Details zur lokalen Entwicklung des Frontends befinden sich in der Datei „`frontend/README.md`“. Nachdem ein Kommandozeilen-Fenster geöffnet und der Pfad zu den Dateien des Frontends geöffnet wurde, muss zuerst durch den Befehl „`npm install`“ die Abhängigkeiten bzw. Pakete installiert werden, die das Frontend verwendet. Anschließend kann das Frontend durch „`npm run dev`“ für die Entwicklung gestartet werden, wodurch es unter der URL „`http://localhost:3000`“ zur Verfügung steht. Bei Änderungen am Code wird die Seite automatisch aktualisiert.

Um das Frontend für den Produktivbetrieb zu kompilieren, muss der Befehl „`npm run build`“ verwendet werden. Hierdurch wird ein „`dist`“-Ordner erstellt, der die kompilierten statischen Dateien des Frontends beinhaltet.

## 3.2. BACKEND

### 3.2. Backend

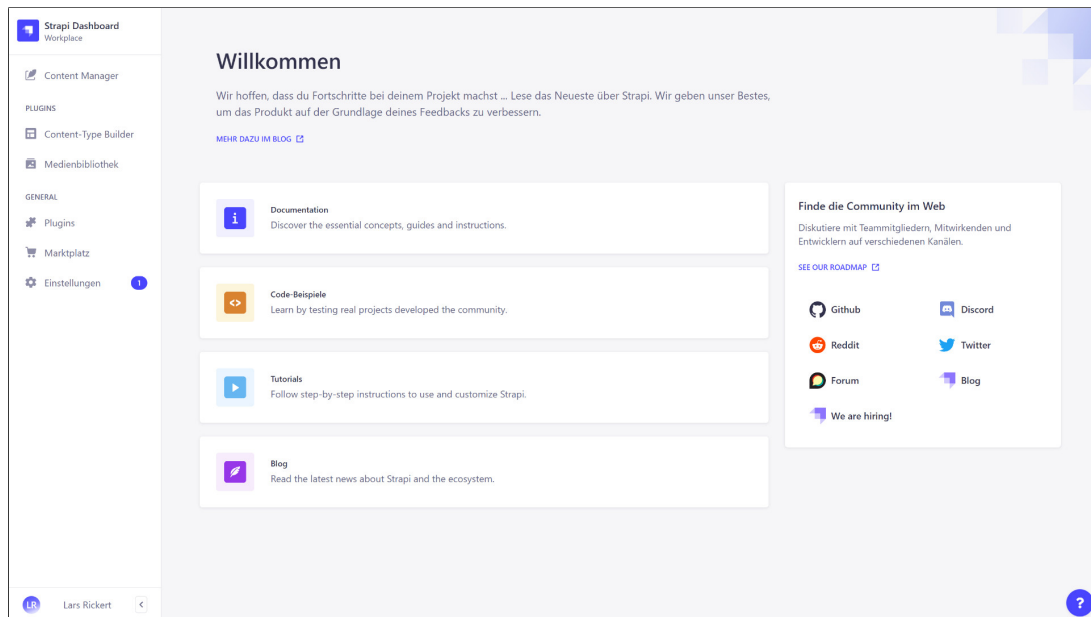


Abbildung 3.7.: Dashboard des Backends

Abbildung 3.7 zeigt das Dashboard bzw. die Startseite des Backends, das über die Route `/admin` erreichbar ist. Unter dem Menüpunkt „Content-Type Builder“ können über die Benutzeroberfläche Inhaltstypen angelegt werden, die dann über die API zugänglich sind (siehe Abbildung 3.8). Im Rahmen dieser Studienarbeit wurde ein Inhaltstyp „Blog Post“ angelegt, der das Erstellen von Blogbeiträgen ermöglicht. Die Inhaltstypen können lediglich während der Entwicklung geändert werden. Im Produktivbetrieb ist dies nicht mehr möglich.

Die konkreten Inhalte für den Inhaltstyp, d.h. die Blogbeiträge können anschließend unter dem Menüpunkt „Content Manager“ verwaltet werden.

## 3.2. BACKEND

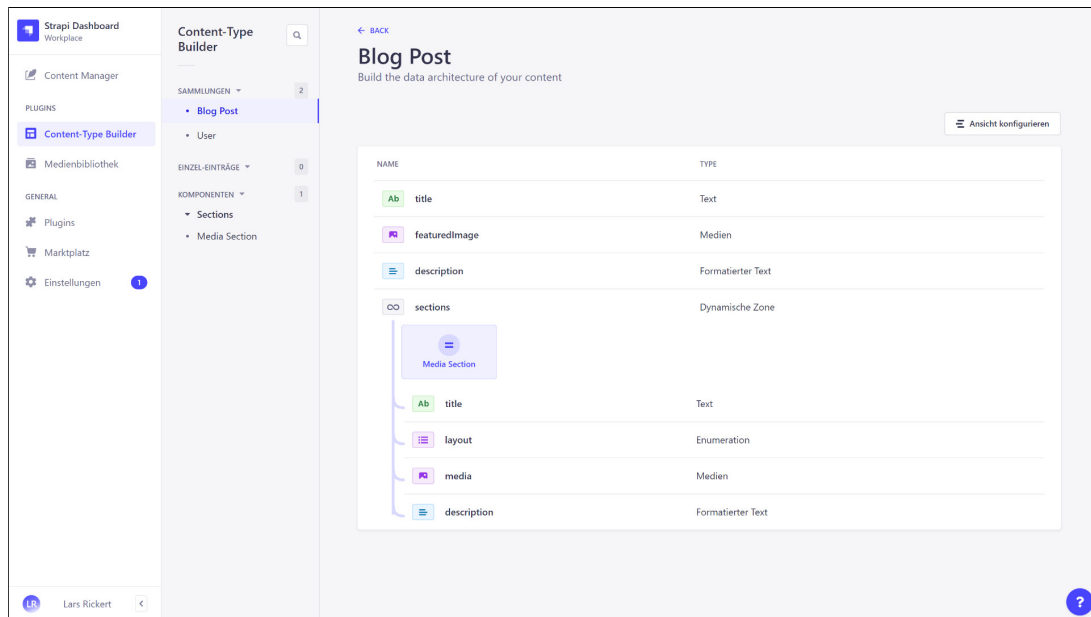


Abbildung 3.8.: Inhaltstyp für die Blogbeiträge

### Lokale Entwicklung

So wie bei der lokalen Entwicklung des Frontends muss zuerst ein Kommandozeilen-Fenster und darin der Pfad zu den Dateien des Backends geöffnet werden. Anschließend müssen ebenfalls die Abhängigkeiten bzw. Pakete, die das Backend verwendet mit „`npm install`“ installiert werden.

Für die lokale Entwicklung des Backends kann dieses mit „`npm run develop`“ gestartet werden. Für den Produktivbetrieb muss es zuerst mit „`npm run build`“ kompiliert und danach mit „`npm start`“ gestartet werden.

Für weitere Informationen über Strapi sei für Interessierte auf [14] verwiesen.



## 4. Deployment

Das Front- und Backend sollen auf einem Linux Server deployed werden. Im Folgenden soll das Aufsetzen des Servers, der Domain sowie die Inbetriebnahme der beiden Anwendungen näher erläutert werden.

### 4.1. Bereitstellen des Linux Servers und der Domain

Zuerst muss ein Linux Server bereitgestellt werden, auf dem später das Front- und Backend betrieben werden sollen. Für diese Studienarbeit wird hierfür ein Virtual Private Server (VPS) der Firma netcup empfohlen. Die VPSs von netcup bieten unter Anderem eine stundenbasierte Abrechnung, DDoS-Schutz, eine Mindestverfügbarkeit von 99,6 % pro Jahr sowie SSD Festplatten mit RAID10. Der angebotene VPS 200 mit einem CPU-Kern, 2GB RAM und 20GB Speicher für monatlich 2,69 € [vgl. 15] ist für diese Studienarbeit ausreichend.

Der VPS kann unter [15] bestellt werden. Sollte noch keine Domain existieren, kann diese ebenfalls bei netcup für 5€ pro Jahr (für eine .de Domain) unter [16] bestellt werden.

#### 4.1.1. Installation von Ubuntu

Sobald der VPS von netcup bereitgestellt wurde, soll Ubuntu als Betriebssystem installiert werden. Standardmäßig wird der Server mit Debian vorinstalliert.

Dazu muss sich unter <https://www.servercontrolpanel.de/SCP/Home> in die Verwaltungsoberfläche des VPS angemeldet werden. Anschließend kann wie in Abbildung 4.1 zu sehen, ein neues Betriebssystem unter dem Menüpunkt „Medien“ installiert werden. Hier sollte die aktuellste Version von Ubuntu ausgewählt und die folgenden Installationsschritte durchgeführt werden.

## 4.1. BEREITSTELLEN DES LINUX SERVERS UND DER DOMAIN

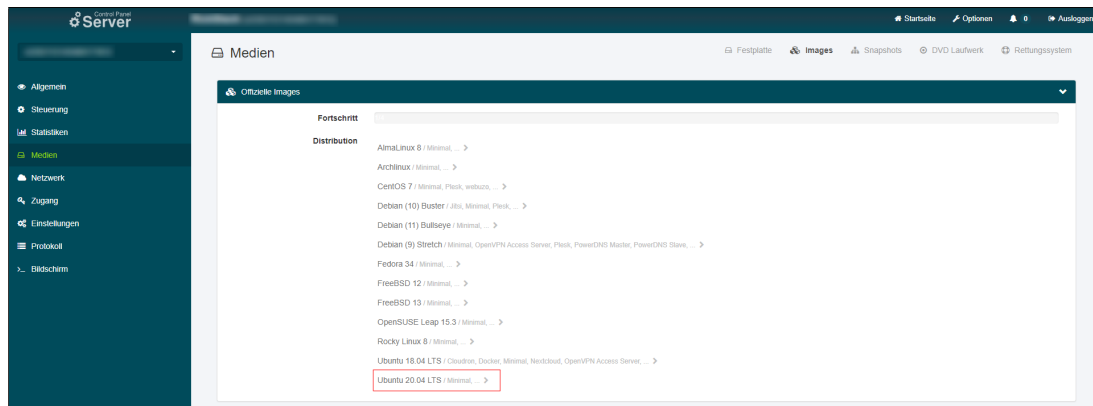


Abbildung 4.1.: Installation von Ubuntu auf dem VPS

### 4.1.2. Zugriff auf den VPS / Einrichtung eines SSH-Clients

Für einen benutzerfreundlichen Zugriff auf den VPS wird die Entwicklungsumgebung Visual Studio Code (VS Code) empfohlen. Nach der Installation mithilfe von [17] muss die „Remote - SSH“ Erweiterung installiert werden (siehe Abbildung 4.2).

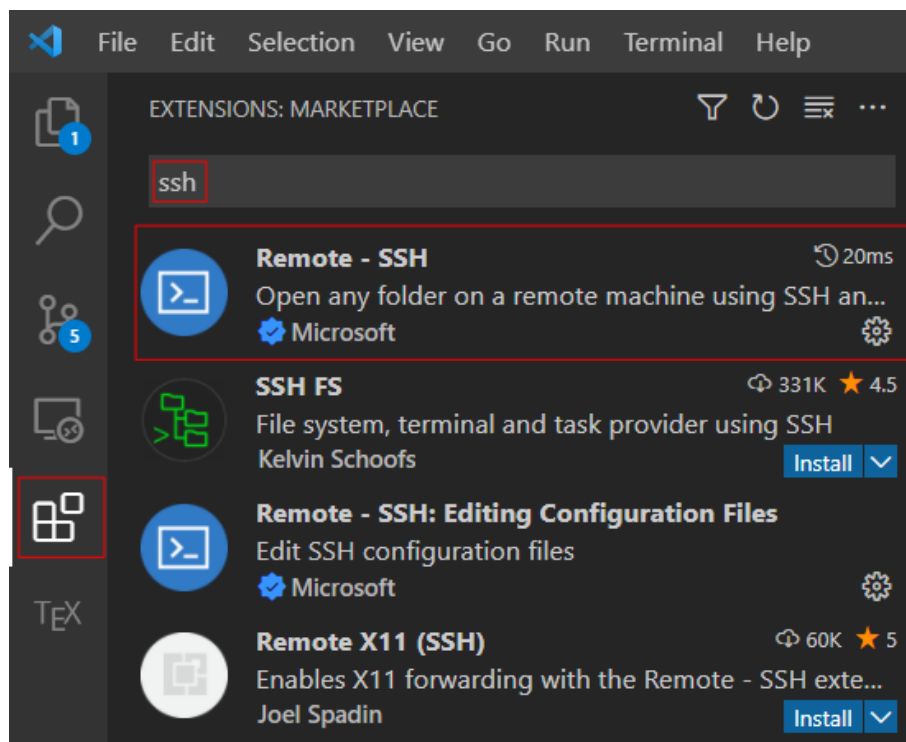


Abbildung 4.2.: Einrichtung von VS Code als SSH-Client

## 4.1. BEREITSTELLEN DES LINUX SERVERS UND DER DOMAIN

Sobald die Erweiterung installiert ist, befindet sich in VS Code unten links ein grünes Symbol, über das eine Verbindung mit dem VPS hergestellt werden kann. Dazu muss auf das Symbol geklickt und in dem sich öffnendem Auswahlménü „Connect to Host“ ausgewählt werden. Anschließend muss angegeben werden mit welchem Host (Server) sich verbunden werden soll. Dazu muss `root@VPS_IP` eingegeben werden, wobei `VPS_IP` durch die IP-Adresse des VPS ersetzt werden muss. Diese findet sich in der E-Mail Bestätigung von netcup oder in der Verwaltungsoberfläche unter dem Menüpunkt „Netzwerk“. Das Passwort für den Zugriff befindet sich ebenfalls in der E-Mail Bestätigung oder auf der Verwaltungsoberfläche nach Installation von Ubuntu.

Nach erfolgreicher Verbindung können mit VS Code über die Benutzeroberfläche Dateien/Ordner erstellt und geöffnet bzw. bearbeitet werden.

### 4.1.3. Konfiguration der Domain / DNS

Sobald die bestellte Domain von netcup bereitgestellt wurde, müssen die DNS Einstellungen der Domain geändert werden, damit sie auf die IP-Adresse des VPS zeigt.

Dazu muss sich unter <https://www.customercontrolpanel.de/> in das Kundenkonto angemeldet werden. Anschließend ist unter dem Menüpunkt „Domains“ die bestellte Domain aufgelistet. Durch einen Klick auf das Lupen-Symbol lassen sich die DNS-Einstellungen der Domain anpassen.

Host	Type	MX	Destination	gültig	löschen
*	A			unknown	<input type="checkbox"/>
@	A			unknown	<input type="checkbox"/>

Abbildung 4.3.: Einrichtung der Domain

Die DNS-Einträge müssen mindestens so wie in Abbildung 4.3 konfiguriert werden, wobei als Destination die IP-Adresse des VPS eingetragen werden muss. Beim Aufruf der Domain in z.B. einem Browser führt diese nun auf den VPS. Die Änderungen der DNS-Einstellungen kann bis zu 48 Stunden dauern. In der Regel sind diese jedoch bereits nach einigen Minuten aktiv.

### 4.2. Einrichtung des nginx-proxy

Nun ist der VPS und die Domain erfolgreich eingerichtet. Im Folgenden wird erläutert, wie das Front- und Backend eingerichtet werden können, damit über die Domain auch auf die Anwendungen zugegriffen werden kann und diese nicht ins leere führt.

Die Anwendungen werden als Docker Container gestartet. Basierend auf dem Docker Container `nginx-proxy` von `jwilder` werden die Anwendungen (Front- und Backend) dann über die Domain erreichbar gemacht und automatisch SSL-Zertifikate angefragt und verwaltet, damit die Daten per `https` verschlüsselt sind. Auf Docker und docker-compose wird im Rahmen dieser Studienarbeit nicht weiter eingegangen.

#### 4.2.1. Installation von Docker und docker-compose

Da sowohl der `nginx-proxy`, als auch das Front- und Backend mit Docker betrieben werden sollen, muss Docker und docker-compose installiert werden. Installationsanleitungen können entsprechend auf [18] und [19] gefunden werden.

Um die Installation zu vereinfachen, kann auch das Skript unter <https://github.com/larsrickert/nginx-proxy/blob/main/utils/install-docker-ubuntu.sh> verwendet werden. Es beinhaltet die offiziellen Installationsschritte zur Installation der aktuellsten Versionen von Docker und docker-compose.

Dazu muss die genannte Datei auf den VPS kopiert werden und das Skript mit

```
bash ./install-docker-ubuntu.sh
```

ausgeführt werden.

#### 4.2.2. Installation des nginx-proxy

Nach der Installation von Docker und docker-compose muss der `nginx-proxy` selbst installiert werden. Dazu kann die Installationsanleitung unter [20] (<https://nginxproxy.larsrickert.de/guide/getting-started.html>) verwendet werden.

Der `nginx-proxy` ist dafür zuständig, jede Docker-Anwendung, die die Umgebungsvariablen `VIRTUAL_HOST` und `LETSENCRYPT_HOST` gesetzt hat, über ebendiesen `host/Domain` zur Verfügung zu stellen. Dabei verwaltet er automatisch SSL-Zertifikate von Let's Encrypt,

### 4.3. STARTEN DES FRONT- UND BACKENDS

um die Verbindung über SSL zu verschlüsseln [21]. Dies ist für das Front- und Backend bereits in der `docker-compose.yml` realisiert, sodass lediglich die Domain in der `.env` Datei eingetragen werden muss (siehe Kapitel 4.3).

## 4.3. Starten des Front- und Backends

Um nun das Front- und Backend dieser Studienarbeit bereitzustellen, muss zuerst mit

```
git clone https://github.com/philippabele/nursing-home-volunteer.git
```

der Code von GitHub heruntergeladen werden.

Danach muss die Datei `.env.example` kopiert und in `.env` umbenannt werden. Diese Datei enthält alle sensiblen Umgebungsvariablen für den Betrieb der beiden Anwendungen. Diese sollten niemals öffentlich geteilt werden. Nach dem Anlegen der `.env` Datei sollten die Werte der Umgebungsvariablen entsprechend geändert werden.

Nun kann das Front- und Backend durch

```
docker-compose up -d
```

gestartet werden. Der zuvor eingerichtete nginx-proxy erkennt die beiden Anwendungen automatisch und stellt sie entsprechend über die `FRONTEND_DOMAIN` und `BACKEND_DOMAIN` bereit, die in der `.env` Datei eingetragen wurden. Es sei angemerkt, dass die Anforderung der SSL-Zertifikate für die Domains ein paar Sekunden dauern kann, sodass direkt nach dem Starten eine Fehlermeldung des Browser erscheinen kann, dass die SSL-Zertifikate nicht gültig sind.

Sollten neue Änderungen für die Anwendungen verfügbar, d.h. der Code auf GitHub geändert worden sein, können sie mit

```
git pull && docker-compose up -d --build
```

aktualisiert werden.

### 4.4. Ersteinrichtung des Backends

Nach erstmaliger Installation des Backends muss ein Administrator-Benutzer angelegt sowie die Berechtigung der Schnittstelle für die Blogbeiträge angepasst werden. Dazu muss zuerst [https://BACKEND\\_DOMAIN/admin](https://BACKEND_DOMAIN/admin) aufgerufen, wobei BACKEND\_DOMAIN durch die Domain ersetzt werden muss, mit der das Backend deployed wurde. Auf der Seite kann anschließend ein Administrator-Benutzer angelegt werden.

Danach muss [https://BACKEND\\_DOMAIN/admin/settings/users-permissions/roles](https://BACKEND_DOMAIN/admin/settings/users-permissions/roles) aufgerufen und auf den Reiter „Public“ geklickt werden. Auf der sich öffnenden Seite muss nun unter dem Reiter „Permissions -> Application“ ein Haken bei „find“ und „findone“ gesetzt werden. So ist die Schnittstelle für die Beiträge öffentlich aufrufbar und benötigt keine Authentifizierung.

### 4.5. Datensicherung und Umzug auf einen anderen Server

Sollte der Bedarf für eine vollständige Datensicherung der Anwendungen bestehen oder diese z.B. auf einen anderen VPS umgezogen werden sollen, ist dies durch die Verwendung von Docker benutzerfreundlich möglich.

Um die Daten der beiden Anwendungen vollständig zu sichern, muss lediglich der Order `docker-data` und die Datei `.env` gesichert werden. `docker-data` wird hierbei beim Start der Anwendungen mit docker-compose erstellt und persistiert die Datenbank des Backends sowie alle hochgeladenen Medien. Das Frontend besteht lediglich aus statischen kompilierten Dateien, sodass dieses nicht gesichert werden muss.

Bei einem Umzug auf einen anderen VPS muss die in diesem Kapitel beschriebene Einrichtung des Servers (inkl. Docker, docker-compose und nginx-proxy) sowie der Domain durchgeführt werden. Anschließend können die gesicherten Daten (`docker-data` und `.env`) auf den neuen Server kopiert und die Anwendungen, wie in Kapitel 4.3 beschrieben, gestartet werden.

## 5. Fazit

Im Rahmen dieser Studienarbeit soll eine Web-Anwendung und ein CMS erstellt werden, um Wissen über Aktivierungen in Pflegeheimen austauschen zu können. Dazu wurde mithilfe von u.a. Vue.js und Bootstrap ein Frontend entwickelt, das neben dem Blog und den Blogbeiträgen zwei statische Seiten zum Darstellen von allgemeinen Informationen über Aktivierungen oder das Projekt besitzt.

Da zum aktuellen Zeitpunkt noch keine konkreten Inhalte feststehen, wurden vorerst Platzhalter-Inhalte eingefügt. So kann möglichen ehrenamtlichen Helfer:innen und Pflegekräften ein frühzeitiger Eindruck des Projekts vermittelt werden.

Die Inhalte für den Blog und die Blogbeiträge wird dynamisch von dem erstellten Backend abgerufen. Dieses wurde mit dem Strapi CMS realisiert und bietet eine separate Benutzeroberfläche, um die Inhalte für den Blog zu verwalten. So bietet es den Vorteil, dass Inhalte eingefügt, angepasst und entfernt werden können, ohne die Programmierung des Frontends anpassen zu müssen. Außerdem wird eine Schnittstelle bereitgestellt, die anderen Anwendungen (außerhalb dieser Studienarbeit) den Abruf der Inhalte ermöglicht, um diese z.B. in einem anderen Blog einzubinden und die Inhalte weiter zu verbreiten.

Um die beiden Anwendungen über das Internet zu veröffentlichen, wird ein selbst gehostetes Deployment mit Docker gewählt. Dieses bietet ein gutes Preis-Leistungsverhältnis sowie hohe Verfügbarkeit und Ausfallsicherheit. Zudem ist das Deployment dadurch flexibel und es können ggf. zukünftig weitere Anwendungen gleichermaßen deployed werden. Durch den Einsatz von Docker sind die Anwendungen zusätzlich von dem Server getrennt, was u.a. Sicherheitsvorteile bietet. Außerdem werden die Anwendungen automatisch neu gestartet, sollte es zu einem Ausfall kommen.

Weiterführend empfiehlt sich das Projekt zukünftig weiter zu entwickeln, um z.B. neue Funktionen, wie eine Filtermöglichkeit der Blogbeiträge, zu implementieren.

# Literatur

- [1] *Anzahl der Pflegebedürftigen in Deutschland in den Jahren 1999 bis 2019*. URL: <https://de.statista.com/statistik/daten/studie/2722/umfrage/pflegebeduerftige-in-deutschland-seit-1999> (besucht am 27.04.2022).
- [2] *Anzahl des Personals in der ambulanten und der stationären bzw. teilstationären Pflege in Deutschland im Zeitraum von 2001 bis 2019*. URL: <https://de.statista.com/statistik/daten/studie/36565/umfrage/personal-in-der-stationaeren-und-ambulanten-pflege/> (besucht am 27.04.2022).
- [3] Microsoft Corporation. *TypeScript: JavaScript With Syntax for Types*. URL: <https://www.typescriptlang.org/> (besucht am 04.03.2022).
- [4] Microsoft Corporation. *TypeScript for JavaScript Programmers*. URL: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html> (besucht am 04.03.2022).
- [5] Sass team. *Sass: Documentation*. URL: <https://sass-lang.com/documentation> (besucht am 07.03.2022).
- [6] Sass team. *Sass: Sass Basics*. URL: <https://sass-lang.com/guide> (besucht am 07.03.2022).
- [7] Inc. Twitter. *Bootstrap · The most popular HTML, CSS, and JS library in the world*. URL: <https://getbootstrap.com/> (besucht am 08.03.2022).
- [8] Evan You. *Introduction*. URL: <https://vuejs.org/guide/introduction.html> (besucht am 04.03.2022).
- [9] Evan You. *Components Basics*. URL: <https://vuejs.org/guide/essentials/component-basics.html> (besucht am 04.03.2022).
- [10] Evan You. *SFC Syntax Specification*. URL: <https://vuejs.org/api/sfc-spec.html> (besucht am 07.03.2022).



## Literatur

- [11] Evan You. *Reactivity API: Core*. URL: <https://vuejs.org/api/reactivity-core.html> (besucht am 07.03.2022).
- [12] Evan You. *Reactivity Fundamentals*. URL: <https://vuejs.org/guide/essentials/reactivity-fundamentals.html> (besucht am 07.03.2022).
- [13] Strapi SAS. *What is a Headless CMS?* URL: <https://strapi.io/what-is-headless-cms> (besucht am 22.03.2022).
- [14] Strapi SAS. *Strapi - Open source Node.js Headless CMS*. URL: <https://strapi.io/> (besucht am 07.03.2022).
- [15] netcup GmbH. *virtuelle Server (VPS)*. URL: <https://www.netcup.de/vserver/vps.php> (besucht am 03.04.2022).
- [16] netcup GmbH. *Unsere Domainangebote*. URL: <https://www.netcup.de/bestellen/domainangebote.php> (besucht am 03.04.2022).
- [17] Microsoft. *Code editing. Redefined*. URL: <https://code.visualstudio.com/> (besucht am 03.04.2022).
- [18] Docker Inc. *Install Docker Engine on Ubuntu*. URL: <https://docs.docker.com/engine/install/ubuntu/> (besucht am 03.04.2022).
- [19] Docker Inc. *Install Docker Compose*. URL: <https://docs.docker.com/compose/install/> (besucht am 03.04.2022).
- [20] Lars Rickert. *Getting Started*. URL: <https://nginxproxy.lars-rickert.de/guide/getting-started.html> (besucht am 20.04.2022).
- [21] Lars Rickert. *Deploy applications*. URL: <https://nginxproxy.lars-rickert.de/guide/deployment.html> (besucht am 20.04.2022).

Teil I.

Datenträger CD