Dokumentation zur betrieblichen Projektarbeit

Erstellen einer individuellen Bewerberplattform auf Basis eines Headless CMS

PRÜFUNGSBEWERBER

Jamal Harris  
Goebensiedlung 14  
56077 Koblenz

Identnummer: 1152593

AUSBILDUNGSBETRIEB

Open New Media GmbH  
Simrockstraße 5  
56075 Koblenz

Tel.: 0261 / 30 380-80  
Fax.: 0261 / 30 380-88

Abgabe: Koblenz, 20.04.2023

Inhaltsverzeichnis

[1. Einleitung 1](#_Toc132891214)

[1.1 Projektbeschreibung 1](#_Toc132891215)

[1.2 Projektbegründung 2](#_Toc132891216)

[1.3 Projektziel 3](#_Toc132891217)

[1.4 Projektabgrenzung 3](#_Toc132891218)

[1.5 Projektumfeld 3](#_Toc132891219)

[1.5.1 Projektschnittstellen 3](#_Toc132891220)

[1.5.2 Entwicklungsumgebung 3](#_Toc132891221)

[1.6 Abweichungen zum Projekt-Antrag 4](#_Toc132891222)

[2. Projektplanung 4](#_Toc132891223)

[2.1 Ressourcen-Planung 4](#_Toc132891224)

[2.2 Entwicklungsprozess 4](#_Toc132891225)

[2.3 Planung von Maßnahmen zur Qualitätssicherung 5](#_Toc132891226)

[2.4 Anstoß-Meeting 5](#_Toc132891227)

[2.5 Auswahl des Headless CMS 5](#_Toc132891228)

[2.6 Auswahl des Formular-Tools 7](#_Toc132891229)

[2.7 Auswahl des Tools für den Mailversand über SMTP 7](#_Toc132891230)

[2.8 Entwerfen von Seitenstruktur & Elementen 7](#_Toc132891231)

[3. Implementierung 8](#_Toc132891232)

[3.1 Installation und Grundeinrichtung des Headless CMS 8](#_Toc132891233)

[3.2 Erstellen einer Vue.JS Applikation 9](#_Toc132891234)

[3.2.1 Installation einer Vue.JS Applikation 9](#_Toc132891235)

[3.2.2 Grundeinrichtung Vue.js Applikation 10](#_Toc132891236)

[3.2.3 Schnittstellen zu Headless CMS einrichten 10](#_Toc132891237)

[3.2.4 Erstellen von Inhaltstypen in CMS 10](#_Toc132891238)

[3.2.5 Auslesen der Seiten-Elemente und Vorbereitung der Daten 11](#_Toc132891239)

[3.2.6 Erstellung von Vue-Komponenten 13](#_Toc132891240)

[3.2.7 Integration von Formtool 14](#_Toc132891241)

[3.3 Mailversand über SMTP 15](#_Toc132891242)

[3.3.1 Problem Mailversand 15](#_Toc132891243)

[3.3.2 Lösung Mailversand 15](#_Toc132891244)

[3.3.3 Grundeinrichtung Laravel Projekt 16](#_Toc132891245)

[3.3.4 Anbinden der Vue Applikation an Laravel 16](#_Toc132891246)

[3.3.5 Abwickeln des Mailversands 17](#_Toc132891247)

[3.4 Umsetzung von Design 18](#_Toc132891248)

[3.5 Routing 19](#_Toc132891249)

[3.6 Abweichungen 20](#_Toc132891250)

[3.7 Maßnahmen zur Qualitätskontrolle 20](#_Toc132891251)

[4. Dokumentation 21](#_Toc132891252)

[5. Projekt Abschluss 22](#_Toc132891253)

[5.1 Soll-Ist-Vergleich 23](#_Toc132891254)

[5.2 Fazit 24](#_Toc132891255)

[6. Ausblick 25](#_Toc132891256)

[7. Anhänge i](#_Toc132891257)

[A1. Tabelle 1: Ressourcen/Kosten Tabelle i](#_Toc132891258)

[A2. Tabelle 2: Produkt Backlog i](#_Toc132891259)

[A3. Gantt-Diagramm ii](#_Toc132891260)

[A4. Use-Case-Diagramm ii](#_Toc132891261)

[A5. Seitenbaum iii](#_Toc132891262)

[A6. Ausschnitt der groben Gestaltung iv](#_Toc132891263)

[A7. Screenshot von Strapi Backend vi](#_Toc132891264)

[A8. Erstellen eines Beispiel-Elements in Strapi vii](#_Toc132891265)

[A9. Beispielseite mit Elementen und Ausschnitt von JSON-Antwort auf Anfrage ix](#_Toc132891266)

[A10. App.js xi](#_Toc132891267)

[A11. ViteConfig.js xii](#_Toc132891268)

[A12. Ausschnitt aus Mail-Model xiii](#_Toc132891269)

[A13. Mail Controller xiv](#_Toc132891270)

[A14. Sequenzdiagramm E-Mails xv](#_Toc132891271)

[A15. Ausschnitt Design-Umsetzung Frontpage xvi](#_Toc132891272)

[A16. Design-Umsetzung Bewerbungsseite xviii](#_Toc132891273)

[A17. Vue Router xix](#_Toc132891274)

[A18. Erstellen der Seitenmap xx](#_Toc132891275)

[A19. Test E-Mail-Versand xxi](#_Toc132891276)

[A20. Unit-Test Überschrift xxii](#_Toc132891277)

[A21. Ausschnitt Kundendokumentation xxiii](#_Toc132891278)

[A22. Ausschnitt der Entwicklerdokumentation xxiv](#_Toc132891279)

[A23. Projektantrag xxv](#_Toc132891280)

[A24. Formblatt für das Projekt xxvi](#_Toc132891281)

# Einleitung

Die Agentur, in der ich meine Ausbildung mache, ist die Open New Media GmbH (ONM). ONM ist in der Webentwicklung tätig, wobei wir Konzeption, Design sowie Realisation übernehmen, um ganzheitliche Lösungen für unsere Kunden zu schaffen. Es wird großer Wert auf Individualisierung gelegt, um verschiedenste Kundenanforderungen zu erfüllen.

Die Agentur hat ihren Hauptsitz in Koblenz und hat insgesamt 18 Mitarbeiter. Den Großteil unseres Kundenstamms macht die Hotellerie aus. Hier bieten wir verschiedene individualisierbare Produkte an, mit denen u. a. Prozesse im Hotel digitalisiert und optimiert werden können. Diese werden von uns als Ergänzung zur bestehenden Softwarelandschaft der Kunden entwickelt.

Ein weiteres Aufgabengebiet der Agentur ist die individuelle Entwicklung und Betreuung von Projekten auf Basis verschiedener Content-Management-Systeme.

## Projektbeschreibung

Um die Rekrutierung neuer Mitarbeiter zu optimieren, soll eine individuelle Bewerberplattform für ONM entwickelt werden, welche im Nachgang auf der TYPO3[[1]](#footnote-1)-Webseite der Agentur verlinkt werden soll, um die bestehende Lösung zu ersetzen. Die Daten sollen über ein Headless CMS[[2]](#footnote-2) bezogen werden. Ein Headless CMS ist ein CMS ohne Frontend, d. h. die Daten werden nur gemanagt und können über Abfragen bezogen werden, es liegt kein User-Interface zur Darstellung der Daten vor.

Eines der Hauptziele dieser Bewerberseite ist es, die Agentur als attraktiven Arbeitgeber darzustellen, ONM optimal zu präsentieren und potenziellen Bewerbern einen ersten Eindruck zu gewähren.

Die Unternehmensdarstellung soll entweder auf 2-5 Seiten oder auf einer Seite, als One-Pager[[3]](#footnote-3), erfolgen. Es sollen mehrere Seitenabschnitte erstellt werden, in denen jeweils Texte, Bilder und Videos integriert werden können. Es soll fest vorgegebene Eingabefelder für Überschriften, Texte und Medien je Seitenabschnitt geben. Diese Seitenabschnitte sollen über das Headless CMS zu verwalten sein.

Zudem soll es eine Übersicht über die verfügbaren Stellenangebote, die nach Beschäftigungsart gruppiert sind, geben. Diese sollen über das Headless CMS angelegt und verwaltet werden. Hier sollen pro Stellenangebot jeweils Titel, Beschäftigungsart, Beginn der Beschäftigung, die Aufgaben des Beschäftigten sowie die Erwartungen an und die Benefits für den Beschäftigten gepflegt werden.

Interessenten sollen auch die Möglichkeit haben, sich direkt über ein Formular bei uns zu bewerben. Die Formulare beinhalten wichtige Felder zur Person, eine Upload-Funktion für Bewerbungsunterlagen und jobspezifische Kenntnisabfragen. Das Bewerbungsformular sowie die hochgeladenen Dateien werden anschließend automatisch per E-Mail an die Personalverantwortliche versendet.

Die Daten auf der Plattform sollen im Headless CMS eingepflegt, und von einer Vue.js Applikation über eine API im JSON Format ausgelesen werden. Die Applikation muss die Daten von den jeweiligen Endpunkten der API auslesen, Daten verarbeiten und daraus eine strukturierte GUI erstellen.

## Projektbegründung

Aktuell sind zahlreiche Kundenprojekte in TYPO3 oder WordPress umgesetzt. Besonders hoch individualisierte TYPO3-Instanzen, können bei Aktualisierungen und Updates enorme Aufwände verursachen. Auf der Kundenseite entstehen somit hohe Kosten, ohne einen offen ersichtlichen Mehrwert. Damit die Kundenzufriedenheit hierunter nicht dauerhaft leidet, wurde über Alternativen zu diesen Systemen diskutiert.

Ergebnis dieser Diskussion war, dass wir künftig auf Headless Content-Management-Systeme (CMS) setzen wollen, um kleinere Projekte, deren Anspruch nicht einer TYPO3- oder WordPress-Instanz entspricht, als schlanke und leicht zu aktualisierende Lösung zu realisieren.

Auch die Webseite der Agentur ist in TYPO3 umgesetzt, und bietet die Möglichkeit, sich über Formulare zu bewerben. Möglichkeiten zur Firmendarstellung sind hier nur bedingt gegeben, der Mailversand funktioniert nur unzuverlässig und die Gestaltung der Nutzeroberfläche entspricht nicht mehr dem Firmenstandard. Eine Erweiterung wäre sehr zeitaufwendig, zudem wird aktuell über einen Relaunch der Webseite nachgedacht.

Um auch in Zukunft wettbewerbsfähig und erfolgreich zu bleiben, ist die Rekrutierung neuer Mitarbeiter für unser Unternehmen von großer Bedeutung. In meinem Projekt soll deshalb eine moderne, eigenständige Bewerberplattform erstellt werden, die im Nachgang auf der Webseite verlinkt wird, um die bestehende Lösung zu ersetzen.

Mein Projekt dient zudem als Pilotprojekt für die Umsetzung von Headless CMS Projekten. Ziel ist es, erste Erfahrungen im Umgang mit dieser Lösung zu sammeln und diese für zukünftige Kundenprojekte zu evaluieren.

## Projektziel

Da die aktuellen Bewerberseiten nicht mehr zeitgemäß sind und eine Erweiterung auf TYPO3-Basis sehr zeitaufwendig wäre, ist das Projektziel das Erstellen einer Bewerberplattform, die sich aus einem Headless CMS, in dem Daten gepflegt werden, und einer Vue.js Applikation, die die Daten dynamisch ausliest und daraus ein Frontend erstellt, zusammensetzt. Diese Lösung soll im Nachgang die bestehende Lösung zur Rekrutierung von Mitarbeitern über die Homepage ersetzen. Des Weiteren besteht das Ziel in einer Evaluation des Einsatzes von Headless CMS in kleineren bis mittelgroßen Projekten, die auf der Auswertung der Ergebnisse basiert. Dies könnte uns langfristig ermöglichen, die Kundenzufriedenheit dadurch zu erhöhen, dass weniger Kosten bei Updates entstehen.

## Projektabgrenzung

Es wird noch nicht das fertige Endprodukt, sondern die Basis für weitere Iterationen geschaffen. Ein Deployment wird in der Projektphase noch nicht stattfinden.

## Projektumfeld

### Projektschnittstellen

Um die im Strapi Backend eingepflegten Daten bereitzustellen, wird von Strapi ein API-Endpunkt bereitgestellt, dieser liefert Daten im JSON Format. Die Applikation kann nun über die Abfrage dieses Endpunktes die Informationen auslesen, und ein Frontend daraus erstellen.

Enge Schnittstellen stellen auch die Betriebsleitung als Auftragsgeber, die Personalverantwortliche sowie die Redakteurin dar, welche über agile Methoden in den Entwicklungsprozess integriert werden sollen. Bei Fragen/Problemstellungen kann auf Kollegen und Ausbilder zugegangen werden.

### Entwicklungsumgebung

Zum Entwickeln wurde ein Desktop-Computer und verschiedene Software verwendet, genauere Spezifikationen sind der Tabelle in Anhang A1 (Tabelle 1: Ressourcen/Kosten Tabelle) der Spalte „Ressource“ unter den Punkten „Hardware“ und „Software“ zu entnehmen.

## Abweichungen zum Projekt-Antrag

Durch Probleme, die in der Entwicklungsphase festgestellt wurden, wurde sich dazu entschieden, die Vue.js Applikation an ein Laravel-Projekt anzubinden, hierauf wird unter 3.3.1 (Problem Mailversand) und 3.3.2 (Lösung Mailversand) genauer eingegangen. Dieser zusätzliche Aufwand konnte jedoch durch Zeitgewinne an anderer Stelle wieder ausgeglichen werden. Diese werden unter 5.1 (Soll-Ist-Vergleich) genauer erläutert. Die Gesamtprojektzeit hat sich nicht geändert.

# Projektplanung

## Ressourcen-Planung

Im Anhang A1 (Tabelle 1: Ressourcen/Kosten Tabelle) befindet sich eine Auflistung aller Ressourcen, die zum Entwickeln der Plattform in dem von der IHK vorgegebenen Zeitraum von 80h verwendet werden. Die Ressourcen werden den entstehenden Kosten gegenübergestellt. Bei der Auswahl der verwendeten Software wurde auf Open-Source-Lizenzen[[4]](#footnote-4) gesetzt, weshalb in diesem Bereich keine weiteren Kosten entstehen.

## Entwicklungsprozess

Zum Entwickeln der Plattform wurde sich für ein inkrementelles Vorgehen entschieden, das an Scrum orientiert ist. Es werden zunächst die allgemeinen Anforderungen an die Plattform in einem Anstoß-Meeting definiert und einem Dokument, angelehnt an den Produkt-Backlog von Scrum, festgehalten. Daraufhin werden Schritte für einen Arbeitszyklus (angelehnt an einen SCRUM-Sprint) geplant, der als erster Zyklus primär alle Hauptanforderungen, sekundär die weiteren Anforderungen der Priorität nach geordnet abarbeitet. Die Ergebnisse werden dann in einer Iteration des Projektes in einem weiteren Meeting vorgestellt und besprochen. Hieraufhin wird aus dem Feedback und den eventuell angepassten allgemeinen Anforderungen, die im Product Backlog ergänzt werden können, der nächste Arbeitszyklus geplant. Dieser kann Anpassungen von bestehenden Elementen (iterativer Charakter, besonders im Bereich der Gestaltung) sowie neue Anforderungen (inkrementell) enthalten. Für den Bearbeitungszeitrum wurden insgesamt drei Meetings eingeplant. Die Zeitplanung wurde in einem Gantt-Diagramm visualisiert, dieses befindet sich im Anhang A3 (Gantt-Diagramm).

Ein agiler Ansatz ist für dieses Projekt besonders gut geeignet, da intern noch keine Erfahrung mit Headless CMS besteht, und es deshalb wichtig ist, auf neue Herausforderungen und Erkenntnisse zu reagieren.

Dass bislang noch nicht alle Anforderungen an die Plattform definiert sind, und diese in der Zukunft noch wachsen werden, wird ebenfalls durch eine inkrementelle/agile Vorgehensweise unterstützt.

## Planung von Maßnahmen zur Qualitätssicherung

Die Qualität soll zum einen softwareseitig durch Tests, als auch durch Code-Reviews durch die Leitung der Anwendungsentwicklung sowie interne Experten in Vue.JS gesichert werden. Zudem soll ein GitHub[[5]](#footnote-5) Repository angelegt werden, das zur Versionskontrolle dient.

## Anstoß-Meeting

In einem Meeting mit allen Beteiligten (Geschäftsführung, Redakteurin, Personalverantwortliche, und dem Entwickler) wurden allgemeine Ideen, Vorstellungen, Inhalte sowie Anforderungen besprochen und diskutiert. Die Ergebnisse wurden in einem Produkt-Backlog zusammengefasst, dieses befindet sich im Anhang A2 (Tabelle 2: Produkt Backlog). Der Produkt-Backlog dient als Sammlung aller Anforderungen, die die Plattform erfüllen soll, und ist nach Priorität sortiert. Auch wurde mir eine Sammlung von Texten/Inhalten für die einzelnen Seiten übergeben. Die Interaktionen zwischen Beteiligten und dem System wurden in einem Use-Case-Diagramm dargestellt. Diese befindet sich im Anhang A4 (Use-Case-Diagramm).

## Auswahl des Headless CMS

Als zu verwendendes CMS wurde sich für Strapi entschieden. Es erfüllt alle Anforderungen, befindet sich unter einer Open-Source-Lizenz (siehe 2.1 Ressourcen-Planung), kann selbst gehostet werden, was ein großer Vorteil in Bezug auf DSGVO-Konformität und Unabhängigkeit von Anbietern mit sich bringt. Zudem bietet es ein Verwaltungstool für hochgeladene Medien und eine intuitive Benutzeroberfläche für Redakteure. Auch werden Daten ausschließlich im JSON-Format[[6]](#footnote-6) bereitgestellt, was sich gut mit Vue.js kombinieren lässt, da Vue.js auf JavaScript basiert.

## Auswahl des Formular-Tools

Nach einer Internetrecherche wurde sich für die Integration von „Formkit“, einem Framework zum Erstellen von Formularen in Vue.js, entschieden. Es kann als Node-Package installiert und in der Vue-Applikation integriert werden. Es deckt alle nötigen Formular-Felder in seiner Funktionalität ab, bietet Multi-Step-Formulare[[7]](#footnote-7), und hat eine ansprechende Funktionsweise. Zudem ist es möglich, eigene Formular-Feld-Validationen[[8]](#footnote-8) zu erstellen.

## Auswahl des Tools für den Mailversand über SMTP

Nach einer Internetrecherche wurde die JavaScript-Library „SMTPJS“ gefunden, und entschieden, diese als einfach zu implementierende Lösung für den Mailversand aus der Vue.js Applikation zu nutzen.

## Entwerfen von Seitenstruktur & Elementen

In Anhang A5 (Seitenbaum) befindet sich ein Entwurf für eine erste simple Seitenstruktur, bei welcher jede Seite als direkte Unterseite der Startseite angelegt wird, in Form eines Seiten-Baumdiagramms.

Auf Basis der vorhandenen Informationen aus dem Product-Backlog, wurden nun grobe Entwürfe für Inhalts- und Seitenelemente angefertigt und gestaltet, ein Ausschnitt hiervon befinden sich in Anhang A6 (Ausschnitt der groben Gestaltung).

# Implementierung

## Installation und Grundeinrichtung des Headless CMS

Die Installation von Strapi wird, wie in der Dokumentation beschrieben, mit dem Kommando „npx create-strapi-app@latest strapiJobs“ in einem CLI[[9]](#footnote-9) ausgeführt. Dies erzeugt eine Strapi Instanz mit dem Namen „strapiJobs“.

Um die Anwendung zu starten und lokal zu hosten, kann man nun den Befehl „npm run develop“ in dem CLI ausführen. Die von diesem Befehl erzeugte Ausgabe wird in Abbildung 1 dargestellt.

|  |
| --- |
|  |

Abbildung 1

Die Applikation ist nun im Browser lokal unter der in dem CLI ausgegeben Adresse erreichbar. Zunächst muss man einen Admin-Nutzer erstellen, mit dem man sich authentifizieren kann. Als Nächstes hat man Zugriff auf das Backend der Applikation, von dessen Startseite sich ein Screenshot im Anhang A7 (Screenshot von Strapi Backend) befindet.

## Erstellen einer Vue.JS Applikation

### Installation einer Vue.JS Applikation

Wie in der Dokumentation beschrieben, kann die Installation mit dem Kommando „npm init vue@latest“ in einem CLI ausgeführt werden. Nun wird in dem CLI der Installationsguide ausgeführt, und ein Ordner erstellt, der die Vue.js Applikation enthält.

Als Nächstes werden mit dem CLI-Kommando „npm install“ Abhängigkeiten installiert. Um das Projekt lokal zu hosten, startet man einen Entwicklungsserver mit dem CLI-Kommando „npm run dev“. Die Ausgabe des Befehls im CLI ist in Abbildung 2 zu sehen.

|  |
| --- |
|  |

*Abbildung 2*

Die Applikation ist nun im Browser unter der im CLI ausgegebenen Adresse erreichbar.

### Grundeinrichtung Vue.js Applikation

Zunächst wurden Views[[10]](#footnote-10) für die Seiten angelegt. Daraufhin wurde der Vue-Router zunächst statisch eingerichtet und die entsprechenden Views über eine statische Navigation verlinkt. Dies diente lediglich als temporäre Lösung.

### Schnittstellen zu Headless CMS einrichten

Um Daten von den Endpunkten des Headless CMS abzufragen, wurde sich für die Verwendung von „Axios“ entschieden. Dieses Node Modul fungiert als http-Client [[11]](#footnote-11), mit dem HTTP Requests aus der App heraus getätigt werden können. Ein Beispiel-Codeauszug ist in Abbildung 3 zu sehen. Hier wird eine URL angefragt, und bei einer Antwort werden die Daten der Seiteninhalte über die Funktion „loadContent“ zu einem Array hinzugefügt.

|  |
| --- |
| axios.get(this.dataSrcURL).then((response) => {  this.pageObject = response.data.data.attributes.PageContent      this.pageObject.forEach(el =>      this.loadContent(el)      )  }) |

Abbildung 3

### Erstellen von Inhaltstypen in CMS

Als Nächstes wurden Teststrukturen angelegt, welche aus Texten und Bildern bestanden. Um diese Teststrukturen abzufragen, muss ein Sammel-Typ[[12]](#footnote-12) erstellt werden. Dieser wurde unter dem Namen „Seite“ angelegt, und kann als Behälter dienen, in dem die erstellten Inhaltselemente in einer dynamischen-Zone[[13]](#footnote-13) platziert werden können.

Hier habe ich Dummy-Daten[[14]](#footnote-14) eingepflegt, um diese testweise auszulesen. Um Bilder aus den dynamischen-Zonen auszulesen, wurde Strapi mit dem CLI Befehl „npm install strapi-plugin-populate-deep“ um ein Plugin erweitert, das es erlaubt, verschachtelte Inhalte aus dynamischen Zonen abzufragen.

Im nächsten Schritt wurden dann die Inhaltstypen auf Basis der entworfenen Elemente (siehe 2.8 Entwerfen von Seitenstruktur & Elementen) im Headless CMS erstellt. Im Anhang A8 (Erstellen eines Beispiel-Elements in Strapi) wird dieser Prozess beispielhaft für eine vereinfachte Überschrift Komponente dargestellt.

Als Nächstes kann die Komponente zu der dynamischen Zone des Seiten-Objektes hinzugefügt, und wie in Abbildung 4 verwendet werden.

|  |
| --- |
|  |

Abbildung 4

Nun kann das Seiten-Element über den entsprechenden Endpunkt mit der Seiten-ID abgefragt werden. Ein Beispiel aus dem Backend von einer Seite mit einem Inhaltselement vom Typ „Überschrift“ sowie einem Inhaltselement vom Typ „Text und Bild“, sowie ein Ausschnitt der Antwort auf die Anfrage des API-Endpunktes ist in Anhang A9 (Beispielseite mit Elementen und Ausschnitt von JSON-Antwort auf Anfrage) zu sehen.

### Auslesen der Seiten-Elemente und Vorbereitung der Daten

Nun wurden innerhalb der Vue.js Applikation die Daten über „Axios“ promise-based[[15]](#footnote-15) abgefragt, die Inhaltselemente verarbeitet und die Daten zu einem von der Vue-Komponente erstellten Datenobjekt mit dem Namen „content“ vom Typ Array hinzugefügt. Um die Inhalte nun auszugeben, wurde ein Vue-for-Loop (v-for=”Element in Array”) mit Konditionen verwendet, es folgt eine vereinfachte Darstellung:

|  |
| --- |
| <div v-for="inhalt in inhaltsArray" >  <div v-if="inhalt['\_\_component']=='beispiel.element'">  //Gib Komponente von beispiel.element mit Informationen  //aus der Abfrage aus  </div>  //...weiter Komponenten  </div> |

So wird später für jedes Inhaltselement die entsprechende Vue-Komponente gerendert, und die Daten können über die „Slot“-Tags zugeordnet, oder als Parameter für konditionelle Zwecke genutzt werden (siehe 3.2.6 Erstellung von Vue-Komponenten).

### Erstellung von Vue-Komponenten

Nun wurden für die Inhaltstypen aus dem Headless CMS (siehe 2.8 Entwerfen von Seitenstruktur & Elementen) in der Vue.js Applikation Komponenten angelegt, die für die Darstellung der Daten als Inhaltselemente im Frontend der App verantwortlich sind. Als Beispiel hierfür befindet sich ein Code-Auszug aus dem Render-Loop und der entsprechenden Komponente für den Inhaltstyp „Inhalts-Überschrift“ in Abbildung 5.

|  |
| --- |
| Header Komponenten Aufruf in Loop    <div v-if="content['\_\_component'] == 'text.content-header'">  <ContentHeader :textAlignCenter="content.alignCenter">    <template #ContentHeader>  {{ content['ContentHeader'] }}  </template>      </ContentHeader>  </div> |
| Header Komponente  <template>      <div class="content-container">          <h3 class="content\_header"  v-bind:class="{ textAlignCenter: textAlignCenter }">              <slot name="ContentHeader"></slot>          </h3>      </div>  </template>  <style scoped>  .content\_header {      font-size: 2rem;      margin-bottom: 3rem;  }  </style>  <script>  export default {      name: 'ContentHeader',      props: ['textAlignCenter'],  }  </script> |

Abbildung 5

### Integration von Formtool

Das ausgewählte Framework für die Formulare (siehe 2.6 Auswahl des Formular-Tool) kann als Node-Package mit dem Befehl „npm install @formkit/vue“ installiert werden.

Es wurden des Weiteren eine Formkit-Erweiterung namens „FormkitMultiStep“ für Multi-Step-Formulare sowie eine Erweiterung mit Themes[[16]](#footnote-16) mit dem Namen „themes“ für ein Design des Formulars hinzugefügt.

Um das Tool in der Vue Applikation verfügbar zu machen, wurde es nun noch im Startpunkt der Vue Applikation integriert, wie in Anhang A10 (App.js) zu sehen ist.

Ein simples, beispielhaftes Formular mit Codeauszug neben der Frontend-Ausgabe ist in Abbildung 6 verdeutlicht.

|  |
| --- |
|  |

Abbildung 6

## Mailversand über SMTP[[17]](#footnote-17)

Nachdem ein simples Testformular angelegt war, war es nun an der Zeit, die Daten aus dem Formular zu versenden. Hierbei konnte leider nicht, wie zunächst geplant, vorgegangen werden.

### Problem Mailversand

Das Node-Package „NPMJS“ (siehe 2.7 Auswahl des Tools für den Mailversand über SMTP) kann in einem CLI mit dem Befehl „npm install smtpjs“ installiert werden. Als während der Konfiguration des Plugins Probleme auftraten, wurde nach einer Internetrecherche festgestellt, dass aufgrund von Spam und Missbrauch nur noch „Elastic-Email“ als Service Provider für SMTP-Dienste zugelassen ist. Da zum einen ein SMTP-Server zur Verfügung steht, und zum anderen Software sowie Lizenzkosten zu vermeiden sind (2.1 Ressourcen-Planung) ist diese Option nicht geeignet.

### Lösung Mailversand

Nach weiterer Internetrecherche und Rücksprache mit Kollegen, stellte es sich als die beste Option heraus, die Vue.js Applikation an ein PHP-Backend anzubinden, über das der Mailversand abgewickelt werden kann. Um das PHP-Backend umzusetzen, wurde sich für das PHP-Framework „Laravel“ entschieden. Neben interner Expertise konnte ich auch selbst in schulischen Projekten schon positive Erfahrungen hiermit sammeln.

### Grundeinrichtung Laravel Projekt

Um das Laravel Projekt zu initialisieren, wurde der Befehl „composer create-project laravel/laravel OnmJobs“ in einem CLI ausgeführt. Nun kann man in das Verzeichnis wechseln und über ein CLI den Befehl „php arstian serve“ ausführen, um das Projekt lokal zu hosten. Die Abbildung 7 zeigt die Ausgabe des CLI. Das Projekt ist jetzt über die in der Ausgabe zu sehende URL zu erreichen.

|  |
| --- |
|  |

Abbildung 7

### Anbinden der Vue Applikation an Laravel

Um die erstellte Vue Applikation an das Laravel Projekt anzubinden, wurde zunächst das Node Package „@vitesjs/plugin-vue“ über ein CLI installiert. Als Nächstes wurde die Datei „Vite.config.js“[[18]](#footnote-18) (siehe A11 ViteConfig.js) um das Plugin und somit eine Vue.js Applikation erweitert.

Hieraufhin habe ich die bereits erstellte Applikation an die in Laravel entstandene Ordner-Struktur angepasst, und in das Laravel Projekt eingefügt. Als Nächstes wurde eine Laravel View erstellt, die einen Container mit einer ID von „App“ enthält. In der Haupt JavaScript Datei des Laravel Projektes wurde nun die Vue.js Applikation auf den Container mit der ID „App“ gemounted[[19]](#footnote-19).

Im Router der PHP Application wurden nun alle Routen auf die Laravel View geleitet, die den Container mit der ID „App“ enthält, und die somit das Frontend der Seite darstellt.

### Abwickeln des Mailversands

Laravel arbeitet nach dem MVC-Prinzip[[20]](#footnote-20), also wurde ein Model, eine View und ein Controller erstellt, um aus einem ausgefüllten Bewerbungsformular, eine E-Mail zu erstellen, welche automatisch an die Personalverantwortliche versendet wird.

Zunächst wurde ein Model mit dem Namen „ApplicationMail.php“ durch das CLI-Kommando „php artisan make:mailable ApplicationMail“ erzeugt. In dieser Klasse werden Inhalte sowie Anhänge über eine erstellte View formatiert zu einem via E-Mail zu versendendem Objekt verarbeitet. Im Anhang A12 (Ausschnitt aus Mail-Model) befindet sich ein Code Ausschnitt dieser Klasse, von der Stelle, an der die Daten an die View bezogen wird.

Um den Mailversand durchzuführen, wurde nun mit dem CLI-Kommando „php artisan make:controller SendMailController“ ein neuer Controller mit dem Namen „SendMailController“ zu dem Laravel Projekte hinzugefügt. Dieser nimmt die Informationen aus dem Bewerbungsformular entgegen, und verwendet das „ApplicationMail“-Model, um eine E-Mail daraus zu erstellen, und zu versenden. Ein Codeauszug der Controller-Klasse befindet sich im Anhang A13 (Mail Controller).

Als Nächstes wurde noch eine Laravel Route hinzugefügt, die auf den „SendMailController“ deutet, und das „action“-Attribut[[21]](#footnote-21) des Testformulars (siehe 3.3 Mailversand über SMTP) wurde auf diese Route gedeutet. So werden die Formulardaten über das POST-Array[[22]](#footnote-22) an den Mail-Controller übergeben.

Nun wurde der Mailversand für einen Test-SMTP Server konfiguriert und darauf getestet.

Daraufhin wurden die Formular-Felder validiert und mit Fehlermeldungen je nach Validations-Ergebnis über Angabe der Validations-Regel angepasst (Abbildung 8) und ein simpler eigener Validator für den Datei-Typ geschrieben. Dieser ist in Abbildung 9 zu sehen und wird über die Datei App.js (siehe A10 App.js) als Validations-Regel integriert.

|  |
| --- |
| <FormKit type="text" label="Vorname" name="firstname" validation="required|alpha"  :validation-messages="{  alpha: 'der Vorname darf keine Zahlen enthalten',  required: 'dies ist ein Pflichtfeld'  }" /> |

Abbildung 8

|  |
| --- |
| var file = function (node) {  return  (node['\_value'][0]['name'].split('.').pop().toLowerCase()=='pdf')  }  file.force = true  export default file |

Abbildung 9

Der Prozess des Mailversands wurde zur Veranschaulichung in einem Sequenz-Diagramm dargestellt, dieses befindet sich im Anhang A14 (Sequenzdiagramm E-Mails).

## Umsetzung von Design

Als Nächstes wurde das Styling für die unter 3.2.6 (Erstellung von Vue-Komponenten) erstellten Komponenten anhand der Entwürfe unter 2.8 (Entwerfen von Seitenstruktur & Elementen) sowie das Design der Seitenelemente umgesetzt. Zu sehen ist ein Ausschnitt des Ergebnisses in den Anhängen A15 (Ausschnitt Design-Umsetzung Frontpage) sowie A16 (Design-Umsetzung Bewerbungsseite).

## Routing

Im nächsten Schritt wurde das Routing in der Vue Applikation so angepasst, dass verschiedenen Seiten nun über die gleiche View in der Vue Applikation dargestellt werden, und eine dynamische Seiten-Navigation eingesetzt werden konnte.

Um den URL-Parameter von einzelnen Seiten anzupassen, wurde im Backend zu dem Seitentyp eine Option für ein wählbares URL-Segment eingefügt, dies ist in Abbildung 10 zu sehen.

|  |
| --- |
|  |

Abbildung 10

Im Vue Router wurde die URL um einen Parameter erweitert. Dieser ist in Anhang A17 (Vue Router) zu sehen. Über den Parameter wird dann die ID des Seiten-Objektes durch eine Map [[23]](#footnote-23) bestimmt. Diese Map wird in der Vue.js Applikation durch eine Abfrage der Seitenelemente erstellt. Wenn im Backend des Headless CMS kein URL-Segment für eine Seite spezifiziert wird, wird der Seiten-Name, welcher einmalig ist, aus den Attributen der Seite verwendet. Dies wird in einem Codeauszug in Abbildung A18 (Erstellen der Seitenmap) verdeutlicht.

Um die Navigation dynamisch zu gestalten, wurde als Nächstes eine Options-Seiten-Komponente als Sammel-Typ erstellt. Hier sollen Elemente/Informationen gepflegt werden, die auf allen Seiten gleich sind (Beispielsweise Logo, Navigation, Footer). Für die Navigation werden Relationen[[24]](#footnote-24) zu bestehenden Seiten als Inhaltselemente verwendet, um diese an die Vue.js Applikation zu übergeben, in welcher daraus Navigations-Menüs erstellt werden.

## Abweichungen

Wie unter 3.3.1 (Problem Mailversand) und 3.3.2 (Lösung Mailversand) erklärt, wurde sich dazu entschieden, die Vue.JS Applikation an ein Laravel Backend anzubinden, um den Mailversand abzuwickeln.

## Maßnahmen zur Qualitätskontrolle

Zum einen wurden Code Tests in Laravel mit dem PHP-Test-Framework „PHP-Unit“ und in Vue.js mit „Vitest“ zur Qualitätssicherung erstellt. Im Anhang A19 (Test E-Mail-Versand) befindet sich ein Beispiel für einen PHP-Unit-Test. Hier wird der E-Mail-Versand als Komponenten-Test auf die Zuordnung der Eingabe-Felder, sowie den korrekten Empfänger kontrolliert. Für die Vue-Komponenten wurden Komponenten-Tests geschrieben, um die korrekte Darstellung und Übergabe der Daten zu gewährleisten, ein Beispiel Codeauszug befindet sich in Anhang A20 (Unit-Test Überschrift).

Es wurde des Weiteren ein Gitlab für die Versionskontrolle bei Anpassungen und Erweiterungen sowie einer übersichtlichen Darstellung von Änderungen im Code, um bei Problemen auf vorherige Stände zurückwechseln zu können, eingerichtet.

Zuletzt hat noch ein Blackbox-Test stattgefunden, welcher von der Redakteurin ausgeführt wurde.

# Dokumentation

Dieses Dokument wurde im Laufe des Projektes angefertigt und nach Abschluss überarbeitet und ergänzt. Die Kundendokumentation wurde im Anschluss unter Zuhilfenahme dieses Dokumentes erstellt. Ein Auszug der Kundendokumentation befindet sich in Anhang A21 (Ausschnitt Kundendokumentation).

Die Entwicklerdokumentation wurde während des Projektes in Form von Kommentaren, und im Anschluss unter Zuhilfenahme dieses Dokumentes verfasst, und mit einer lokalen Installationsanleitung im Gitlab hinterlegt. Ein Ausschnitt der Dokumentation ist in Anhang A22 (Ausschnitt der Entwicklerdokumentation) zu sehen.

# Projekt Abschluss

## Soll-Ist-Vergleich

|  |  |  |
| --- | --- | --- |
| **Arbeitsschritt** | **Ist** | **Soll** |
| **Projektplanung** | | |
| Auswahl Headless CMS | 1 Stunde | 1 Stunde |
| Planung von Seitenstruktur & Einrichtung von Elementen | 3 Stunden | 1.5 Stunden |
| Abstimmung | 1 Stunde | 1Stunde |
| **Basisgestaltung** | | |
| Erstellung von groben Entwürfen für Elemente | 3 Stunden | 3 Stunden |
| Abstimmung | 1 Stunde | 1 Stunden |
| **Einrichten des Headless CMS** | | |
| Grundeinrichtung Headless CMS | 3 Stunden | 3 Stunden |
| Einrichtung von Rollen im Headless CMS | 2 Stunden | 1 Stunde |
| **Erstellen einer Vue.JS Applikation** | | |
| Grundeinrichtung Vue.js Application | 2 Stunden | 1 Stunde |
| Umsetzung der Seitenstruktur | 4 Stunden | 4 Stunden |
| Schnittstellen zu Headless CMS einrichten | 6 Stunden | 4 Stunden |
| Erstellen von Inhaltstypen im Headless CMS | 8 Stunden | 8 Stunden |
| Erstellung von Objekten aus Daten von Headless CMS | 8 Stunden | 8 Stunden |
| Darstellung der erstellten Objekte | 8 Stunden | 8 Stunden |
| Erstellen Laravel Projekt und Anbinden der Vue Applikation | 0 | 2 Stunden |
| Mailversand über SMTP | 0 | 3 Stunden |
| **Tests** | | |
| Funktionstests | 5 Stunden | 3.5 Stunden |
| Modultests | 5 Stunden | 5 Stunden |
| Integrationstests | 5 Stunden | 5 Stunden |
| Dokumentation | | |
| Projektdokumentation | 10 Stunden | 12 Stunden |
| Dokumentation für Entwickler | 3 Stunden | 1 Stunde |
| Kundendokumentation | 2 Stunden | 1 Stunden |
| Zeitplan Ergebnis | | |
| Gesamtzeit | 80h | 80h |

In der Tabelle ist der Zeitplanung aus dem Projektantrag zu sehen. Diese wurde um die Spalte „ist“ erweitert. Zeiteinsparungen sind mit Grün, Zeitüberschreitungen mit Rot markiert.

Bei dem Punkt „Planung von Seitenstruktur & Einrichtung von Elementen“ konnte durch eine simple Seitenstruktur und erfolgreicher Abstimmung im Anstoß-Meeting (siehe 2.4 Anstoß-Meeting) Zeit eingespart werden.

Der Schritt „Einrichtung von Rollen im Headless CMS“ konnte durch die hohe Benutzerfreundlichkeit von Strapi ebenfalls schneller durchgeführt werden als geplant.

Die Schritte „Grundeinrichtung Vue.js Application“ sowie „Schnittstellen zu Headless CMS einrichten“ konnten durch nahtloses Zusammenspiel von Vue.JS und Strapi ebenfalls schneller durchgeführt werden, als in der Planung angegeben.

Als der Mailversand nicht wie geplant durchgeführt werden konnte, stellte es sich als beste Alternative heraus, die Vue.js Applikation an ein Laravel-Backend anzubinden, über welches der Mailversand abgewickelt wird. Hierdurch wurden die Punkte „Erstellen Laravel Projekt und Anbinden der Vue Applikation“ und „Mailversand über SMTP“ hinzugefügt.

Für die Projektdokumentation wurde mehr Zeit aufgewendet als geplant. Diese konnte jedoch bei der Kundendokumentation teilweise wieder eingespart werden, da Abbildungen und Erklärungen teilweise wieder verwendet werden konnten. Durch das Verwenden von Kommentaren im Code während des Entwickelns konnte auch die Entwicklerdokumentation schneller angefertigt werden.

Das heißt, die Gesamtzeit hat sich, trotz dieser Änderungen, nicht verändert.

## Fazit

Mit dem gewählten Entwicklungsprozess war die Umsetzung des Projektes, trotz Komplikationen, ein Erfolg. Die Kommunikation zwischen dem Headless CMS und der Vue.js Applikation ging überraschend leicht von der Hand. Das entstandene Produkt dient der Geschäftsleitung als zufriedenstellende Basis für den zukünftigen Einsatz zur Rekrutierung neuer Mitarbeiter.

Zudem hat sich auch die Erkenntnis ergeben, dass der Einsatz von Headless CMS eine gute Möglichkeit darstellt, um kleine bis mittelgroße Kundenprojekte umzusetzen. Dies kann also künftig das Angebot der Firma erweitern und die Kundenzufriedenheit durch schlanke, leicht zu aktualisierende Systeme erhöhen.

# Ausblick

Die Plattform wird vor allem im Bereich der Inhaltselemente an Gestaltung und Umfang noch erweitert werden. Zunächst wird die Anwendung auf einem Testserver deployt, auf dem die Redakteurin mit der Inhaltspflege experimentieren, die Zuverlässigkeit des Mailversands garantiert werden, und Anpassungswünsche und Erweiterungen geplant und abgestimmt werden können.

Wenn die Plattform bereit für den Livegang ist, wird sie auf der Webseite der Agentur verlinkt, um die bestehende Lösung zu ersetzen.

# Anhänge

1. Tabelle 1: Ressourcen/Kosten Tabelle

|  |  |
| --- | --- |
| **Ressource** | **Kosten** |
| **Hardware** | |
| Desktop Computer (Windows 10 Pro, 64-Bit-Betriebssystem, Intel(R) Core(TM) i5-3570 CPU 3.40GHz Prozessor, 16GB RAM) | keine Kosten |
| **Software** | |
| Visual-Studio Code (v1.72.1) | keine Kosten |
| Node JS (v16.14.2) | keine Kosten |
| Node Package Manager (v6.14.7) | keine Kosten |
| Vue.js (v3.2.45) | keine Kosten |
| **Personal** | |
| Auszubildener (80h) |  |
| Anwendungsentwickler (2h) |  |
| Personalverantwortliche (2h) |  |
| Redaktionsverantwortliche (2h) |  |
| Firmenleitung (2h) |  |

1. Tabelle 2: Produkt Backlog

|  |  |
| --- | --- |
| **Anforderung** | **Priorität (1-5, niedrig zu hoch)** |
| Datenverwaltung über Headless CMS | 5 |
| Umsetzung in Vue.Js | 5 |
| Automatischer Mailversand mit hochgeladenen Anhängen | 5 |
| Einbindung von Bildern und Videos | 4 |
| Gegliederte Übersicht der Stellenangebote | 4 |
| Listendarstellung von Anforderungen und Benefits | 3 |
| Multi Step Formular | 3 |
| Seperater Bereich für Ausbildung | 2 |
| Übersichtliche Gestaltung | 2 |
| Freundlicher Charakter | 1 |
| Pflegbare Navigations Menüs | 1 |
| Trenner Elemente für Übersicht | 1 |
| Logo Gallerie für Kunden-Logos | 1 |

Produkt-Backlog des ersten Inkrementes

1. Gantt-Diagramm

|  |
| --- |
|  |

Gant Diagramm der Projektplanung

1. Use-Case-Diagramm

|  |
| --- |
|  |

Use-Case-Diagramm zur Nutzerinteraktion

1. Seitenbaum

|  |
| --- |
|  |

Seitenbaum der Webanwendung

1. Ausschnitt der groben Gestaltung

|  |
| --- |
|  |
|  |

Screenshot vom Ausschnitt der groben Gestaltung

1. Screenshot von Strapi Backend

|  |
| --- |
|  |

Screenshot von Strapi Backend

1. Erstellen eines Beispiel-Elements in Strapi

|  |
| --- |
|  |

|  |
| --- |
|  |

Screenshots Erstellung Beispiel-Element

1. Beispielseite mit Elementen und Ausschnitt von JSON-Antwort auf Anfrage

|  |
| --- |
|  |
| C:\Users\Cheefa Mali\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Antwort.png |

Screenshot einer Beispielseite mit Überschrift, Text, Bild und JSON-Antwort auf Anfrage

1. App.js

|  |
| --- |
| import { createApp } from 'vue'  import App from './VueApp/App.vue'  import router from './VueApp/router'  /\* import the fontawesome core \*/  import { library } from '@fortawesome/fontawesome-svg-core'  /\* import font awesome icon component \*/  import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'  /\* import specific icons \*/  import { faTwitter, faFacebook, faLinkedin, faXing} from '@fortawesome/free-brands-svg-icons'  import { faX } from '@fortawesome/free-solid-svg-icons'  /\* add icons to the library \*/  library.add(faTwitter, faFacebook, faLinkedin, faXing, faX )  //custom form validation  import file from './VueApp/custom-form-rules/file'  import './VueApp/assets/main.css'  // FormKit imports  import { plugin as formKitPlugin, defaultConfig } from '@formkit/vue'  import { createMultiStepPlugin } from '@formkit/addons'  import '@formkit/themes/genesis'  import '@formkit/addons/css/multistep'  const app = createApp(App).component('font-awesome-icon', FontAwesomeIcon)  app.use(router)  app.use(formKitPlugin, defaultConfig({      plugins: [createMultiStepPlugin()],      rules: { file }  }))  app.mount('#app') |

Codeausszug der App.js-Datei

1. ViteConfig.js

|  |
| --- |
| import { defineConfig } from 'vite';  import laravel from 'laravel-vite-plugin';  //import vue  import vue from '@vitejs/plugin-vue';  export default defineConfig({      define:{          'process.env': {}      },      plugins: [          vue({              template: {                  compilerOptions: {                    isCustomElement: (tag) => {                      return tag.startsWith('Formkit')                    }                  }              }          }),          //ad vue to laravel as plugin          laravel({              input: ['resources/css/app.css', 'resources/js/app.js'],              refresh: true,          }),      ],  }); |

Codeauszug der ViteConfig.js-Datei

1. Ausschnitt aus Mail-Model

|  |
| --- |
| …  /\* Get the message content definition.  \*/  public function content(): Content{  return new Content(      view: 'mail.email',          with: [          'firstname' => $this->firstname,              'lastname' => $this->lastname,              'email' => $this->email,              'phone' => $this->phone,              'html\_css' => $this->html\_css,              'php' => $this->php,              'mysql' => $this->mysql,              'js' => $this->js,              'frontendFrameworks' => $this->frontendFrameworks,              'wordpress' => $this->wordpress,              'typo3' => $this->typo3,              'bonusText' => $this->bonusText,              'jobapplication' => $this->jobapplication          ],      );  }  /\*\*  \* Get the attachments for the message.  \*  \* @return array<int, \Illuminate\Mail\Mailables\Attachment>\*/  public function attachments(): array  {  return [  Attachment::fromPath($this->file->path())          ->as($this->file->getClientOriginalName())              ->withMime($this->file->getClientMimeType())          ];      }  }  … |

Codeausschnitt des Mail-Models

1. Mail Controller

|  |
| --- |
| class SendMailController extends Controller  {      public function sendmail(Request $request)      {          //set in conf          $to\_email = env('MAIL\_TO');          Mail::to($to\_email)->send(new ApplicationMail(              $request->input('firstname'),              $request->input('lastname'),              $request->input('email'),              $request->input('phone'),              $request->input('html\_css'),              $request->input('php'),              $request->input('mysql'),              $request->input('js'),              $request->input('frontendFrameworks'),              $request->input('wordpress'),              $request->input('typo3'),              $request->file('file'),              $request->input('bonusText'),              $request->input('jobapplication')          ));          return redirect('/')->withErrors(['msg' => 'The Message']);}  // The email sending is done using the to method on the Mail // facade  Mail::to('test@onm.de')->send(new ApplicationMail($name, $content));  } |

Codeausschnitt des Mail-Controllers

1. Sequenzdiagramm E-Mails

|  |
| --- |
|  |

Sequenzdiagramm des E-Mail-Versands

1. Ausschnitt Design-Umsetzung Frontpage

|  |
| --- |
|  |
|  |

Ausschnitt Design Umsetzung Frontpage

1. Design-Umsetzung Bewerbungsseite

|  |
| --- |
|  |

Screenshot Design-Umsetzung Bewerbungsseite

1. Vue Router

|  |
| --- |
| import { createRouter, createWebHistory } from 'vue-router'  import HomeView from '../views/HomeView.vue'  import PageNotFound from '../views/PageNotFound.vue'  // these are passed to `createRouter`  const routes = [    // dynamic segments start with a colon    { path: '/:pageName', component: HomeView },    { path: '/', component: HomeView, props: { HomePage: 1 } },    {      path: '/404', component: PageNotFound    },    {      path: '/:pathMatch(.\*)\*', component: PageNotFound    }  ]  const router = createRouter({    history: createWebHistory(import.meta.env.BASE\_URL),    routes: routes  })  export default router |

Codeausschnitt aus von Vue-Router-Datei

1. Erstellen der Seitenmap

|  |
| --- |
| axios.get(this.dataSrcURL).then((response) => {  this.navbarItems = response.data.data  // var map = new Map() in data return object      this.navbarItems.forEach((el) => {  // this.pages[el.id] = el.attributes.PageName or URL slug // if it is set      if (el.attributes.URLSlug != null && el.attributes.URLSlug != "Default") {          this.map.set(el.attributes.URLSlug, el.id);          } else {          this.map.set(el.attributes.PageName, el.id);      }  })  }) |

Screenshot der Erstellung der Seitenmap

1. Test E-Mail-Versand

|  |
| --- |
| {      public function test\_Mail(): void      {          Mail::fake();          $pdfData = UploadedFile::fake()->create('fake.pdf');          $to\_email = "i@onm.de";          $mailable = new ApplicationMail(              'Testfirstname',              'Testlastname',              'Testemail',              'Testphone',              'Testhtml\_css',              'Testphp',              'Testmysql',              'Testjs',              'TestfrontendFrameworks',              'Testwordpress',              'Testtypo3',              $pdfData,              'TestbonusText',              'Testjobapplication'          );          Mail::to($to\_email)->send($mailable);          $mailable->assertTo('i@onm.de');          $mailable->assertSeeInOrderInHtml([              'Testjobapplication',              'Testfirstname',              'Testlastname',              'Testemail', 'Testphone',              'Testhtml\_css',              'Testphp',              'Testmysql',              'Testjs',              'TestfrontendFrameworks',              'Testwordpress',              'Testtypo3',              'TestbonusText'          ]);      }  } |

Codeausschnitt aus Test für E-Mail-Versand

1. Unit-Test Überschrift

|  |
| --- |
| /\*\*   \* @vitest-environment happy-dom   \*/  import { describe, it, expect } from 'vitest'  import { mount } from '@vue/test-utils'  import ContentHeader from './ContentHeader.vue'  describe('bigtext test', () => {      it('should render', () => {          const wrapper = mount(ContentHeader, {              propsData:{                  textAlignCenter: 'true'              }          })   expect(wrapper.find('h3.textAlignCenter').exists()).toBeTruthy()      })  }) |

Beispiel Unit Test für Überschrift

1. Ausschnitt Kundendokumentation

|  |
| --- |
|  |

Screenshot von Ausschnitt der Kundendokumentation

1. Ausschnitt der Entwicklerdokumentation

|  |
| --- |
|  |

Screenshot vom Ausschnitt der Entwicklerdokumentation

1. Projektantrag

…

1. Formblatt für das Projekt

…

1. Name eines Content-Management-Systems [↑](#footnote-ref-1)
2. Conten-Management-System(e) [↑](#footnote-ref-2)
3. Startseite die alles beinhaltet, ohne Unterseiten [↑](#footnote-ref-3)
4. Lizenz mit der Software kostenlos genutzt, verändert und vertrieben werden kann [↑](#footnote-ref-4)
5. Plattform für Kollaboration und Versionskontrolle von Software [↑](#footnote-ref-5)
6. Java Script Objekt Notation, Art von JavaScript Objekte darzustellen [↑](#footnote-ref-6)
7. Formulare, die in mehrere Schritte aufgeteilt sind (wie Teil-Formulare) [↑](#footnote-ref-7)
8. Methodik, um Eingaben der Nutzer auf Plausibilität/Korrektheit zu prüfen [↑](#footnote-ref-8)
9. Command Line Interface, beschreibt die Schnittstelle zwischen Mensch und Computer, um Befehle in Textform einzugeben [↑](#footnote-ref-9)
10. Vue.js Komponente, die via Routing über die URL angesprochen und aktualisiert werden kann [↑](#footnote-ref-10)
11. versendet Http Anfragen und nimmt Antworten entgegen (bsp.Web-Browser) [↑](#footnote-ref-11)
12. Behälter, in dem erstellte und native Komponenten platziert werden können [↑](#footnote-ref-12)
13. Behälter, in dem eine undefinierte Menge von diversen Inhaltselementen in beliebiger Reihenfolge erfasst werden [↑](#footnote-ref-13)
14. Daten ohne Bedeutung, nur zum Testen [↑](#footnote-ref-14)
15. basiert auf dem Konzept, dass etwas zu einem Zeitpunkt einen Wert haben wird (nicht von Anfang an hat) [↑](#footnote-ref-15)
16. Zusammen greifendes Grund Design der Komponenten [↑](#footnote-ref-16)
17. Simple Mail Transfer Protocol, Protokoll zum austausch von Emails [↑](#footnote-ref-17)
18. Konfigurations Datei des Build-Tools [↑](#footnote-ref-18)
19. in einen Behälter „eingefügt“ [↑](#footnote-ref-19)
20. Model-View-Controller Teilt die Logic von der Darstellung und dem zu Erzeugeden Objekt [↑](#footnote-ref-20)
21. entscheided darüber, über welche URL beim versenden des Formulares angeprochen wird [↑](#footnote-ref-21)
22. PHP-Array mit Variabeln die über die POST-Methode versendet werden [↑](#footnote-ref-22)
23. Equivalent zu einem Array, bei dem sowohl Schlüssel als auch Wert zugeordnet wird [↑](#footnote-ref-23)
24. Beziehungen zwischen Objekten [↑](#footnote-ref-24)