Angebot & Konzept

hotelsuite IBE

[Firma]

[Veröffentlichungsdatum]

Inhaltsverzeichnis

[1. Ihre Aufgabenstellung 3](#_Toc100756298)

[2. Lösungsvorschlag / Integration der hotelsuite IBE 4](#_Toc100756299)

[2.2 Integration der IBE 4](#_Toc100756301)

[2.3 Optional: Verfügbarkeitsfarben aus der Suite8 auf Ihrer Webseite 4](#_Toc100756302)

[2.4 Optische Anpassung der IBE 5](#_Toc100756303)

[2.5 Verkürzter Buchungsprozess 6](#_Toc100756304)

[2.6 Integration des Preis Widgets 7](#_Toc100756305)

[2.7 Was ist im Leistungsumfang der IBE und des Gutscheinsystems enthalten? 8](#_Toc100756306)

[3. Unser Angebot 9](#_Toc100756307)

[4. Ihre Ansprechpartner 10](#_Toc100756308)

Inhaltsverzeichnis

1.Ausgangssituation

1.1 Analyse des Projektauftrags (ist analyse, siehe projektnatrag) [1.1 Projektbeschreibung]

1.2 Ableitung der Projektziele und Projektabgenzung [1.2/1.3/1.4]

1.3 Darstellung des Projektumfeldes und der betrieblichen schnittstellen [1.5]

2.Resourcen/-Ablaufplan

2.1 Projektmanagement, Terminplanung, Ablaufplan (inkl. Meilensteine) [2.1/2.2]

2.2 Plaunung der Qualitätssicherung (projektbezogen und technisch) [2.3]

(Soll Zustand)

3. Durchführung und Auftagsbeschreibung

3.1 Prozess-Schnitte und vorgehensweise, was habe ich gemacht? [6]

( Progarmmablaufplan, ER-Diagramme usw.)

3.2 Abweichunge Anpassung, Entscheidungen [6.7]

3.3 Maßnhmen zur Qualitätskontrolle (Projektbez. Und tech.)

4.Projektabchluss

4.1 Soll-ist-Vergleich (Abweichung, Anpassungen)

4.2 Reflexion/Fazit

4.3 Optimierungsmöglickeiten/Ausblick

# 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 ersetzten. Die Daten sollen über ein Headless CMS[[2]](#footnote-2) bezogen werden. Ein Headless CMS ist ein CMS ohne Frontend (Daten werden nur gemanaged 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 „Onepager“, 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 wird, 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 der Agentur an 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 dem Firmenstandard. Eine Erweiterung wäre sehr zeitaufwendig, zudem ist ein Relaunch der Webseite in Sprache.

Da mein Unternehmen hohen Wert auf Rekrutierung neuer Mitarbeiter legt, soll in meinem Projekt eine eigenständige Bewerberplattform erstellt werden, die im Nachgang auf der Webseite verlinkt wird, um die Bestehende Lösung zu ersetzten.

Mein Projekt dient des weiteren, als das erstes Projekt das auf Basis eines Headless CMS umgesetzt werden soll, als Pilot, um Erfahrung im Umgang mit dieser Lösung sammeln, und diese für zukünftige Kundenprojekte evaluieren zu können.

## Projektziel

Da die auf der Typo-3-Webseite der Agentur bestehende Lösung zur Rekrutierung neuer Mitarbeiter zum einen nur einen unzuverlässigen Mailversand, durch den Bewerbungen teilweise verloren gehen, sowie keine ansprechende Nutzer-Oberfläche bietet, und eine Erweiterung der bestehenden-Typo3 Instanz sehr Zeitaufwendig wäre, ist das Ziel zum einem das Erstellen einer Bewerberplattform, die sich aus einem Headless CMS, in dem Daten gefpflegt werden, und einer Vue.js Applikation, die die Daten Dynamisch ausließt und daraus ein Front-End erstellt, zusammensetzt. Diese Lösung soll im Nachgang die bestehende Lösung zur Rekrutierung von Mitarbeitern über die Homepage erstetzten. Des weiteren besteht das Ziel einer Evaluation des Einsatztes 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, sonder 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 diese Endpunktes die Informationen auslesen, und ein Frontend daraus erstellen.

Enge Schnittstellen stellen auch die Betriebsleitung als Auftragssgeber, die Personalverantworliche 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 Entwicklen wurde eine Desktop-Computer und verschiedene Software verwendet, genauere Spezifikationen sind der Tablle in Anhang (impr) der Spalte „Resource“ unter den Punkten „Hardware“ und „Software“zu entehmen.

## Abweichungen zum Projekt-Antrag

Durch Probleme die in der Entwicklungsphase festgestellt wurden, wurde sich dazu entschieden die Vue.Js Applikation an ein Laravel-Projekt anzubindne, hierauf wird unter 3.3.1() und 3.3.2() genauer eingegangen. Dieser zustäliche Aufwand konnte jedoch durch Zeitgewinne, diese werden unter 6.XY (titel) genauer erläutert, wieder ausgeglichen werden, weshalb die Gesamtprojektzeit sich nicht geänder hat.

# Projektplanung

## Resourcen Planung

Im Anhang [\*impr] befindet sich eine Auflistung aller Ressourcen, die zum Entwickeln der Plattform in dem von der IHK vorgegebenen Zeitraum von 80h verwendet werden. Die Resourcen werden den enstehenden Kosten gegenüber gestellt. Bei der Auswahl der Verwendeten Software wurde auf Open Source Lizenzen[[3]](#footnote-3) gestetzt, weshalb in diesem Bereich keine weiteren Kosten entstehen.

## Entwicklungsprozess

Zum entwicklen der Platfform wurde sich für ein Inkrementelles Vorgehen entschieden, dass an Scrum orientiert ist. Es werden zunächst die allgemeinen Anforderungen an die Plattform in einem Anstoß-Meeting definert und einem Dokument, angelehnt an den Produkt-Backlog von Scrum, festgehalten. Darauf hin werden Schritte für einen Arbeitszyklus (angelehnt an einen SCRUM-Sprint) geplant, der als erster Zyklus primär alle Haupt-Anforderungen, 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. Hierauf hin 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 Beriech der Gestaltung) sowie neue Anforderungen (inkrementell)enthalten. Für den Bearbeitungszeitrum wurden insgesamt drei Meetings eingeplant. Die Zeitplanung wurde in einem Gantt-Diagramm visualisert, dieses befindet sich im Anhang [\*impr].

Ein Agiler Ansatz ist für dieses Projekt besonders gut geeignet, da intern noch keine Erfahrung mit Headless CMS beteht, und so deshalb wichtig ist auf neue Herausforderungen und Erkenntinsse zu reagieren.

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

## Planung von Maßnahmen zur Qualitätssicherung

Die Qualität soll zum einen Software-seitig durch Tests, als auch durch Code-Reviews durch die Leitung der Anwendungsentwicklung sowie interne Experten in Vue.JS gesichert werden. Zu dem soll ein GitHub[[4]](#footnote-4) Repository angelegt werden, das zur Versionierungskontrolle dient.

## Anstoß-Meeting

Hier wurde ein Meeting mit allen beteiligten (Geschäftsführung, Redakteurin, Personalverantwortliche, und dem Entwickler) gehalten, wobei allgemeine Ideen, Vorstellungen, Inhalte sowie Anforderungen besprochen und diskutiert wurden. Die Ergebnisse wurden in einem Produkt-Backlog zusammengefasst, dieses befindet sich im Anhang [\*impr ProduktBacklog]. 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 Iinteraktionen zwischen Beteiligten und dem System wurden in einem Use-Case-Diagramm dargestellt. Diese befindet sich im Anhang [insert]

## Auswahl von Headless CMS

Als zu verwendendes CMS wurde sich für Strapi entschieden. Es erfüllt alle Anforderungen, befindet sich unter einer Open-Source-Linzens [2.1 Resourcen Planung], kann selbst gehostet werden, was ein großer Vorteil im Bezug auf DSGVO-Konformität und Unabhängigkeit von Anbietern mit sich bringt, bietet ein Verwaltungstool für hochgeladene Medien und eine Intuitive Benutzeroberfläch für Redakteure. Zudem werden Daten ausschließlich im JSON-Format [[5]](#footnote-5)Bereitgestellt, was sich gut mit Vue.js kombinieren lässt, da Vue.js auf JavaScript basiert.

## Auswahl von Formular Tool

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[[6]](#footnote-6), und hat eine ansprechende Funktionsweise. Zudem ist es möglich eigene Formular-Feld-Validationen[[7]](#footnote-7) zu erstellen.

## Auswahl von Tool für Mailverand über SMTP

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

## Entwerfen von Seitenstruktur & Elementen

In Anhang [\*link] befindet sich ein Entwurf für eine erste simple Seitenstruktur, bei welcher jede Seite als direkte Unterseite der Startseite angelegt wird ,in Form eines Baumdiagrammes.

Auf Basis der vorhanden Informationen aus dem Product-Backlog, wurden nun grobe Entwürfe für Inhalts- und Seitenelemente angefertigt und gestaltet, ein Ausschnitt hiervon befinden sich in den Anhängen 4 und 5 [\*link].

# Implementierung

## Installation und Grundeinrichtung des Headless CMS

Die Installation von wird wie in der Dokumentation beschrieben mit dem Kommando „npx create-strapi-app@latest strapiJobs“ in einem CLI [[8]](#footnote-8) ausgeführt. Dies erzeugt eine Strapi Instanz mit dem Namen „strapiJobs“.

Um die Anwendung zu starten und local 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.

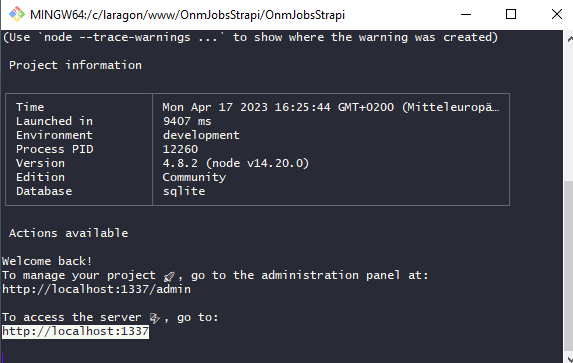


Abbildung 1

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 6 [\*link] 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“ Abhängikeiten 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 im Abbildung 2 zu sehen.

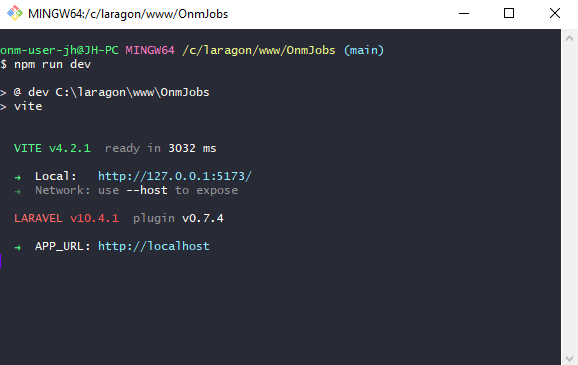


Abbildung 2

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

### Grundeinrichtung Vue.Js Applikation

Zunächst wurden Views für die Seiten Angelegt. Daraufhin wurde der Vue-Router zunächst Statisch eingerichtet, und die enstprechenden Views Über eine Statische Navigation verlinkt. Dies diente lediglich als Temporärer 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 [[9]](#footnote-9)mit dem HTTP Requests aus der App heraus getätigt werden können. Eine Beispiel Codeauszug ist in Abbildung 3 zu sehen. Hier wird die URL angefragt, und bei einer Antwort werden die Daten über die Funktion „loadContent“ zu einem Array hinzugefügt.

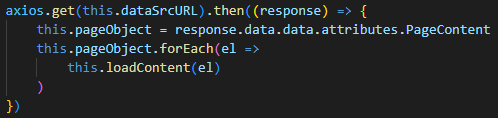


Abbildung 3

### Erstellen von Inhaltstypen in CMS

Als nächstes wurden Teststrukturen angelegt, welche aus Texten und Bildern bestanden. Um diese Teststrukturen abzufragen, muss ein Sammel-Typ[[10]](#footnote-10) erstellt werden. Dieser wurde unter dem Namen „Seite“ angelegt, und kann als Behälter dienen, in dem die erstellten Inhaltselemente in einer dynamischen-Zone[[11]](#footnote-11) platziert werden können.

Hier habe ich Dummy-Daten[[12]](#footnote-12) 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 Inhalstypen auf Basis der Entwurfenen Elemente (siehe 6.3) im Headless CMS erstellt. Im Anhang 7 [\*link] wird dieser Prozess beispielhaft für eine Vereinfachte Überschrift Komponente dargestellt.

Als nächstest kann die Komponente zu der dynamischen-Zone des Seiten-Objektes hinzufügen, 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 8[\*link] zu sehen

[\*create Abbildungen]

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

Nun wurden innerhalb der Vue.js Applikation die Daten über „Axios“ promise-based[[13]](#footnote-13) abgefragt, die Inhalstelement 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=”Inhalt in Array”) mit Conditionen verwendet, es folgt eine vereinfachte Darstelltung:

<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

Nun wurde für die Inhaltstypen aus dem Headless CMS [siehe 2.8] 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 „Überschrift“ in Abbildung 5.

|  |
| --- |
| Header Komponenten Aufruf in Loop  C:\Users\onm-user-jh\AppData\Local\Microsoft\Windows\INetCache\Content.Word\HeaderInLoop.png |
| Header Komponente |

Abbildung 5

### Integration von Formtool

Das Ausgewählte Framework für die Formulare [siehe 2.7] kann als node package mit dem Befehl „npm install @formkit/vue“ installiert werden.

Es wurden des weiteren ein Formkit-Addon names „FormkitMultiStep“ für Multi-Step-Formulare und sowie ein Formkit -Addon mit Themes [[14]](#footnote-14)mit dem Namen „themes“ für ein Design des Formulares hinzugefügt.

Um es in der Vue Applikation verfügbar zu machen, wurde es nun noch im Startpunkt der Vue Applikation integriert, wie in Anhang Einstiegspunkt der VueApplikation app.js (Anhang 9)[\*link] zu sehen ist.

Eine simples Beipielhaftes Formular mit Codeauszug neben der Frontend-Ausgabe ist in in Abbildung 6 verdeutlicht.

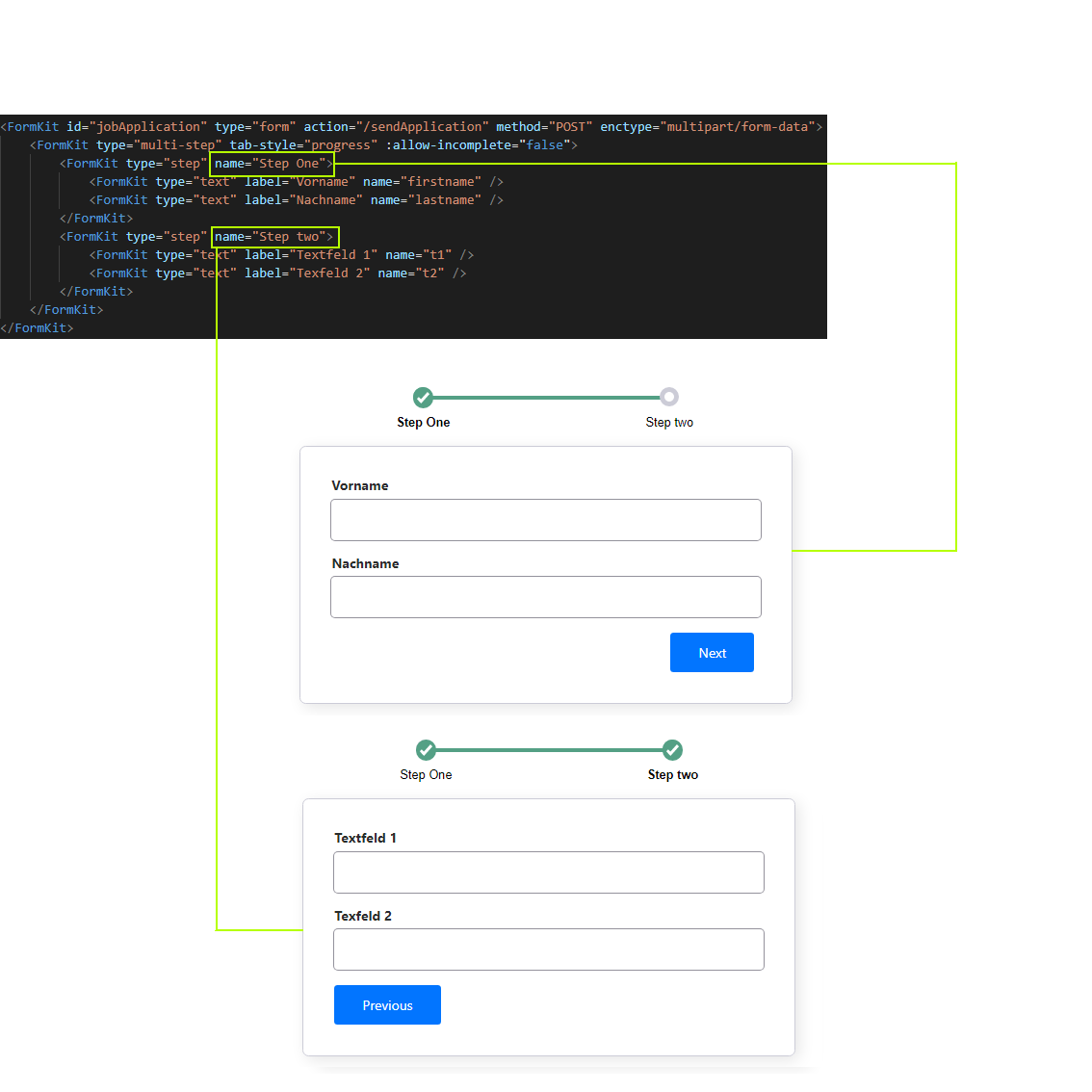


Abbildung 6

## Mailversand über SMTP [[15]](#footnote-15)

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, herauf wird unter [3.3.1 link ] genauer eingegangen.

### Versand aus Vue.js Applikation (problem)

Das node Package „NPMJS“ [5.3] kann in einem CLI mit dem Befehehl „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 Lizenkosten zu vermeiden sind [3.1] ist diese Option nicht geeinget.

### Versand aus Vue.js Applikation (Lösung)

Nach weiterer Internetrecherche und Rücksprache mit Kollegen stelle 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. Hierin besteht intern Expertise und auch ich konnte hiermit in Schulischen-Projekten schon positive Erfahrungen 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 local zu hosten. Die Abbildung 7 zeigt die Ausgabe des CLI. Das Projekt ist nun über die in der Ausgabe zu sehende URL zu erreichen.



Abbildung 7

### Anbinden von Vue Applikation an Laravel

Um die Erstellete Vue Applikation an das Laravel Projekt anzubinden wurde nun zunächst das Node Package „@vitesjs/plugin-vue“ über ein CLI installiert. Nun wurde die Datei „Vite.config.js“[[16]](#footnote-16) (siehe Anhang [\*link]) um das Plugin und somit eine Vue.js Applikation erweitert.

Hieraufhin habe ich die bereits Erstelle 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 [better expr.] wurde nun die Vue.js Applikation auf den Container mit der ID „App“ gemounted[[17]](#footnote-17).

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 Fronted der Seite darstellt.

### Abwickeln des Mailversands

Laravel arbeitet nach dem MVC-Prinzip [[18]](#footnote-18), also wurde ein Model, eine View und ein Controller erstellt.

Zunächst wurde eine Model mit dem Namen „ApplicationMail.php“ durch das CLI-Kommando „php artisan make:mailable ApplicationMail“ erzeugt. In dieser Klasse werden Inhalte sowie Anhäge über eine erstellte View formatiert zu einem via Email versendbarem Objekt verarbeitet. Im Anhang befindet sich ein Code Ausschnitt dieser Klasse, und der Stelle an der die Daten an die View [\*link] übergeben werden.

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[ref] entgegen, und verwendet das „ApplicationMail“-Model, um eine Email darus zu erstellen, und zu versenden. Ein Codeauszug der Controller-Klasse befindet sich im Anhang[\*link].

Als nächstes wurde noch eine Laravel Route hinzugefügt, die auf den „SendMailController“ deutet, und das „action“-Attribut[[19]](#footnote-19) des Testformulars [ref formerstellung] wurde auf diese Route gedeutet. So werden die Formulardaten über das POST-Array[[20]](#footnote-20) an den Mail-Controller übergeben.

Als nächstest wurde der Mailvesand für einen Test-SMTP Server konfiguriert und darauf getestet.

Nun wurden die Formular-Felder validiert, und mit Fehlermeldungen je nach validations Ergebniss über Angabe der Validations-Regel angepasst, wie in im Abbildung 8 zu sehen ist , und ein simpler eigener Valdiator für den Datei-Typ geschrieben. Dieser ist in Abbildung 9 zu sehen und wird über die Datei App.js [link] als Validationsregel integriert.

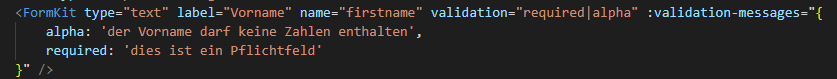
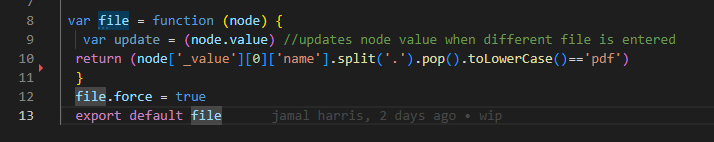


Abbildung 8

Abbildung 9



Der Prozess des Mailversands wurde zur Veranschaulichung in einem Sequenz-Diagramm dargestellt, diese befindet sich im Anhang [link]

## Umsetzung von Design

Als nächstes wurde das Styling für die unter [ref erstellung vue komponenten] erstellen Komponenten anhand der Entwürfe unter [erstllung entwurf fuer content], sowie das Design der Seitenelemente umgesetzt. Zu sehen ist ein Ausschnitt des Ergebnisses in Anhang [\*link].

## 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.

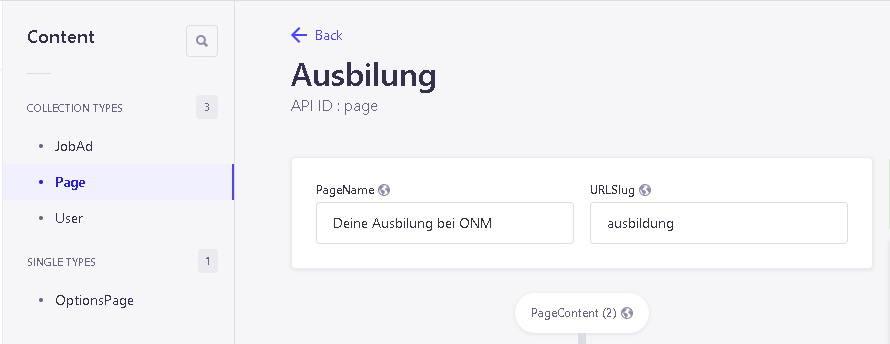


Abbildung 10

Im Vue Router wurde die URL um einen Parameter erweitert. Dieser ist in Anhang [\*link] zu sehen. Über den Paramter wird dann die ID des Seiten-Objektes über eine in der Vue.js Applikation durch eine Abfrage der Seitenelemente erstelle Map [[21]](#footnote-21) bestimmt. Wenn im Backend des Headless CMS kein URL-Segment für eine Seite spezifiert wird, wird der Seiten-Name, welcher Einmalig ist, aus den Attributen der Seite verwendet. Dies wird in einem Codeauszug in Abbildung [\*link] verdeutlicht.

Um die Navigation Dynamisch zu gestalten wurde als nächstest eine Options-Seiten-Objekt erstellt. Hier sollen Elemente/Informationen gepflegt werden, die auf allen Seiten gleich sind (Beispielsweise Logo, Navigation, Footer). Für die Navigation werden Relationen[[22]](#footnote-22) zu bestehenden Seiten als Inhalstelemente verwendet, um diese an die Vue.js Applikation zu übergeben, in welcher daraus Navigations Menüs erstellt werden.

## Abweichungen

Wie unter [link] erklärt, wurde sich dazu entschieden die Vue.JS Applikation an ein Laravel Backend anzubinden, um den Mailversand abzuwickeln.

## Maßnahmen zur Qualitätskontrolle

[Todo: Code tests, code review durch die Leitung der Anwendungsentwicklung und Vue Experten] wie in planung [verlinken] angesetzt...

Zudem wurde ein Gitlab für Versionskontrolle bei Anpassungen und Erweiterungen, sowie einer Übersichtlichen Darstellung von Änderungen im Code ,um bei Problemen auf vorherigen Stand zurück wechseln zu können, eingerichtet.

# Dokumentation

|  |  |  |
| --- | --- | --- |
| Arbeitsschritt | Zeitangabe Projektantrag | Zeit verwendet |
| 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 |
| Test | | |
| 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 |

Dieses Dokument wurde im laufe des Projektes angefertigt, und nac Abschluss überarbeitet und ergänzt. Die Kundendokumentation wurde im Anschluss unter Zuhilfenahme dieses Dokuementes erstellt. Ein Auszug der Kundendokumentation befinden sich in Anhang (create).

Die Entwicklerdokumentation wurde im während des Projektes in Form von Kommentaren, und im Anschluss unter Zuhilfenamhe dieses Dokumentes verfasst, und mit einer lokalen Installationsanleitung im Gitlab hinterlegt. Ein Ausschnitt der Dokumentation ist in Anhang [\*create] zu sehen.

# Projekt Abschluss

## Soll-ist-Vergleich

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

Wie in 5.1 Zusehen ist, gab es zeitliche Abweichungen. Diese werden im folgenden begründet.

Bei dem Punkt „Planung von Seitenstruktur & Einrichtung von Elementen“ konnte durch eine simple Seitenstruktur und erfolgreicher Abstimmung im Anstoßmeeting [\*link] Zeit eingespart werden.

Der Schritt „Einrichtung von Rollen im Headless CMS “ konnte durch die hohe Benutzerfreundlichkeit von Strapi ebenfalls schneller durchgeführt 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 bei 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 den 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. Hierdurch konnte bei der Kundendokumentation Zeit eingespart werden, Abbildungen und Erklärungen Teilweise wieder verwendet werden können. Durch das verwenden von Kommentaren im Code während des Entwickelns konnte ebenfalls die Entwickler Dokumentation schneller angefertigt werden.

Die Gesamtzeit hat sich trotz dieser Änderungen nicht verändert.

## Fazit

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

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

# Ausblick

Die Plattform wird vorallem im Bereich der Inhaltselement an Gestaltung und Umfang noch erweiterte werden. Zunächst wird die Anwendung auf einem Testserver deployed, auf dem die Redakteurin mit der Inhaltspflege experimentieren, die zuverlässigkeit des Mailverands garantiert werden, und Anpassungswünsche und Erweiterungen geplant und Abgestimmt werden könnnen.

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

Anhänge

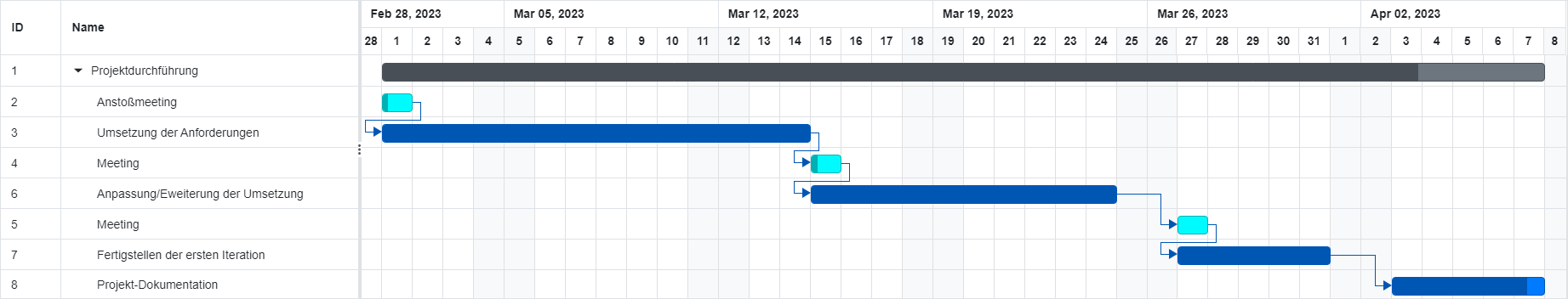
Tabelle 1: Ressourcen/Kosten Tabelle

|  |  |
| --- | --- |
| Resource | Kosten |
| **Hardware** | |
| Desktop Computer (Windows 10 Pro, 64-Bit-Betriebssystem, Intel(R) Core(TM) i5-3570 CPU 3.40GHz Prozessor, 16GB RAM) | Stromverbrauch ausrechnen? |
| **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) |  |
| Personalverantworliche (2h) |  |
| Redaktionsverantwortliche (2h) |  |
| Firmenleitung (2h) |  |

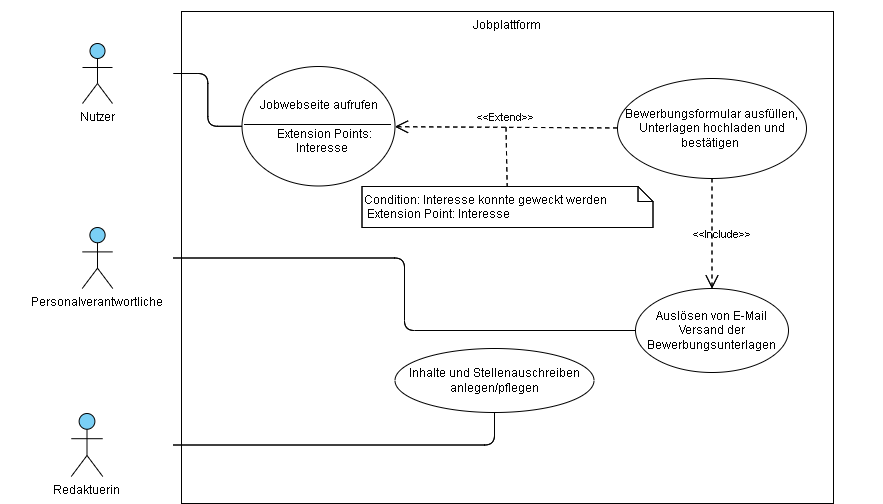
Tabelle 2: Produkt Backlog

|  |  |
| --- | --- |
| **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 |
| Multi Step Formular | 3 |
| Seperater Bereich für Ausbildung | 2 |
| Übersichtliche Gestaltung | 1 |
| Freundlicher Charakter | 1 |

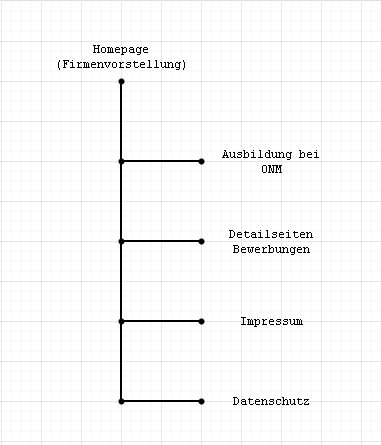
Anhang 1: Gantt-Diagramm



Anhang 2: Use-Case-Diagramm



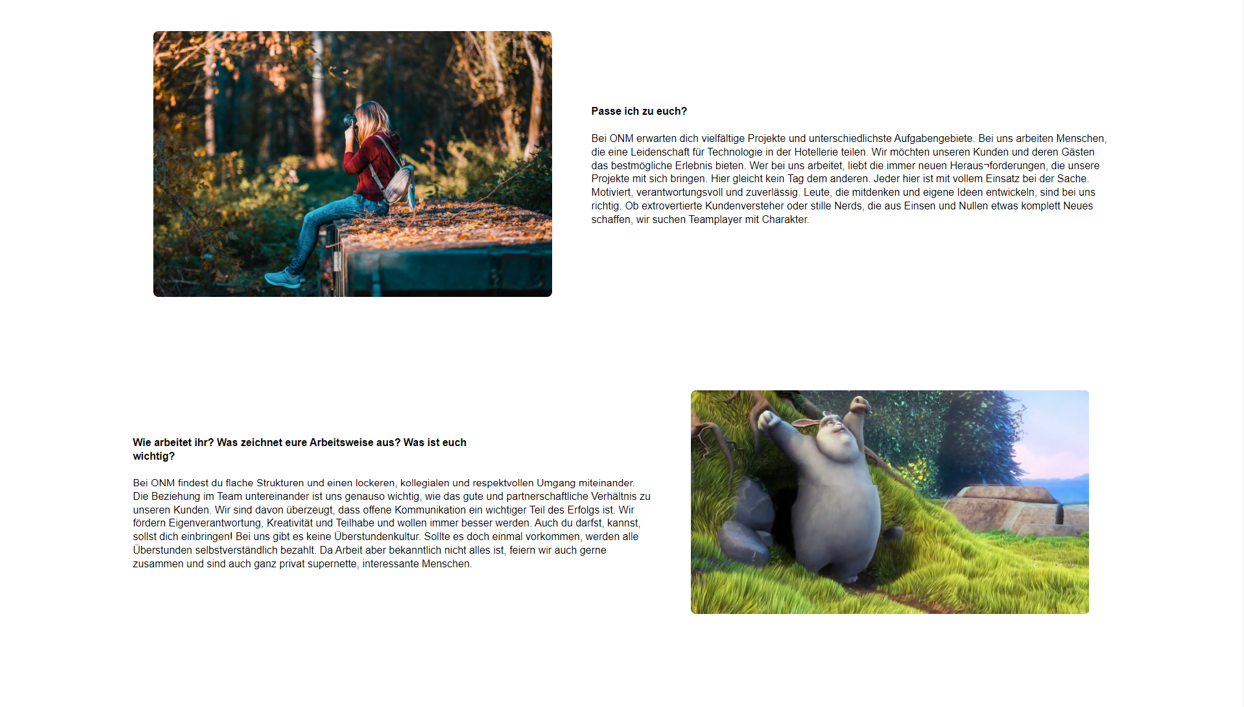
Anhang 3: Seitenbaum



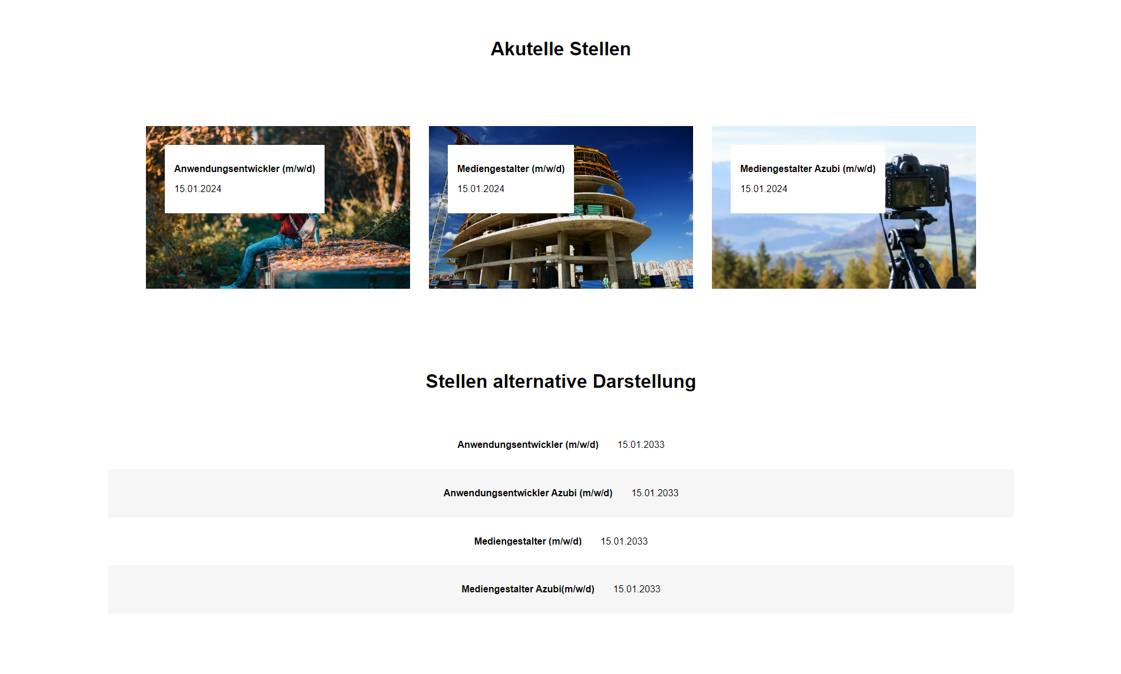
Anhang 4: Ausschnitt der groben Gestaltung (Teil 1)

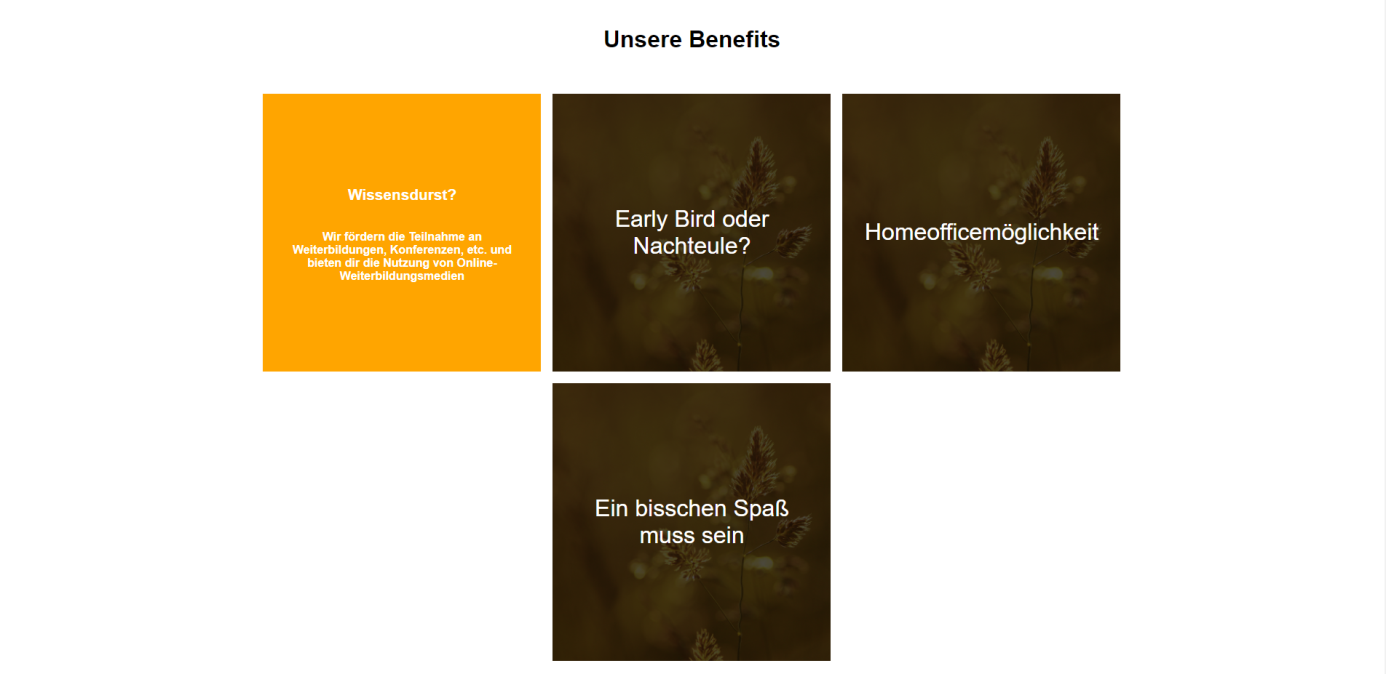


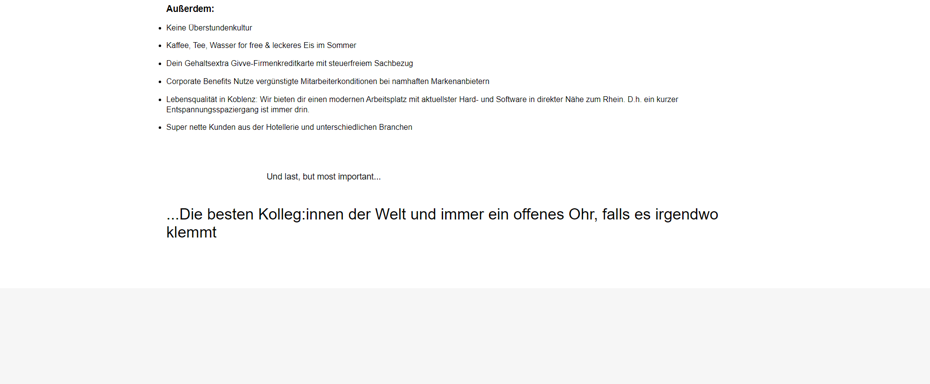




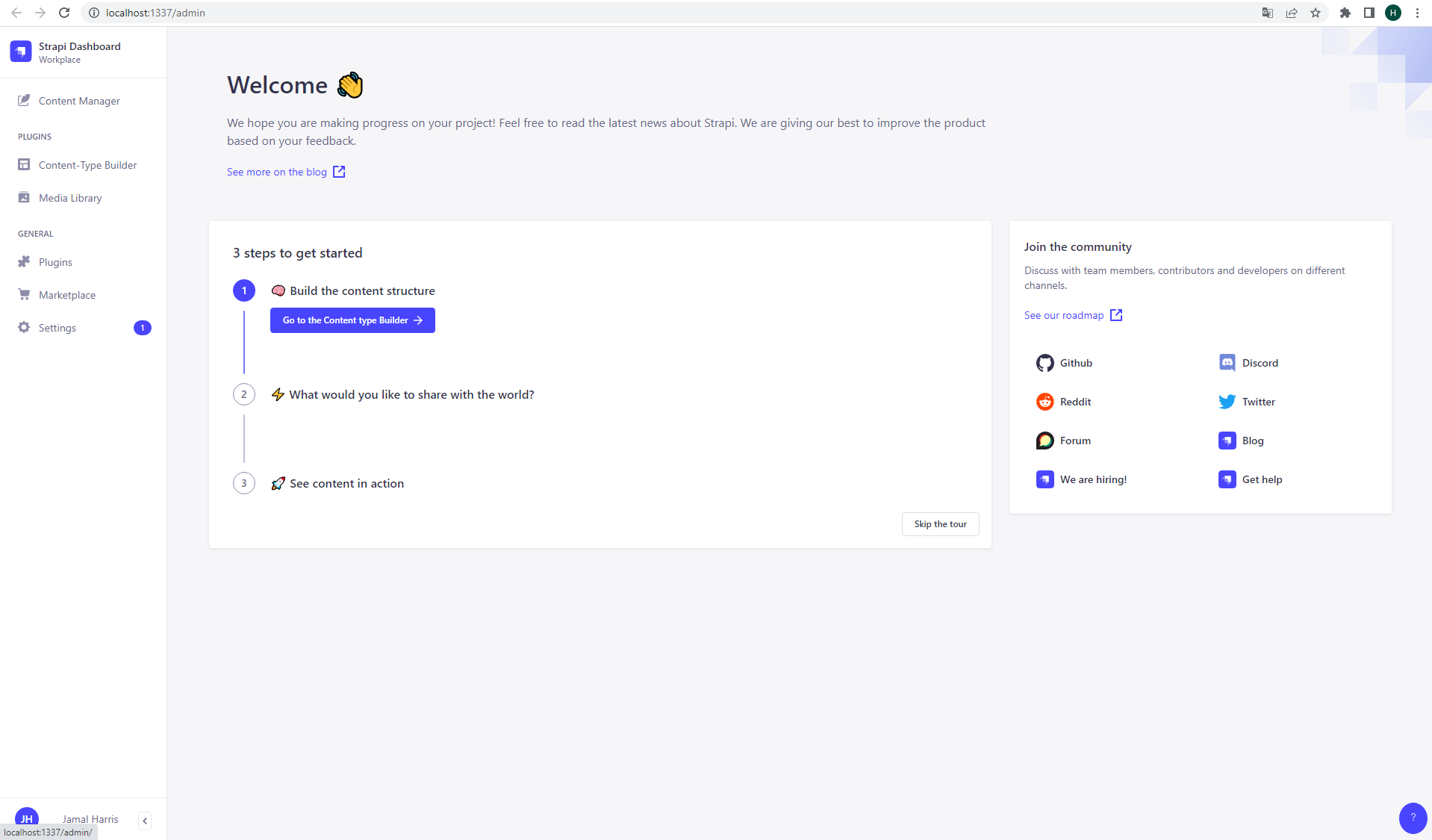
Anhang 5: Ausschnitt grober Gestaltung (Teil 2)



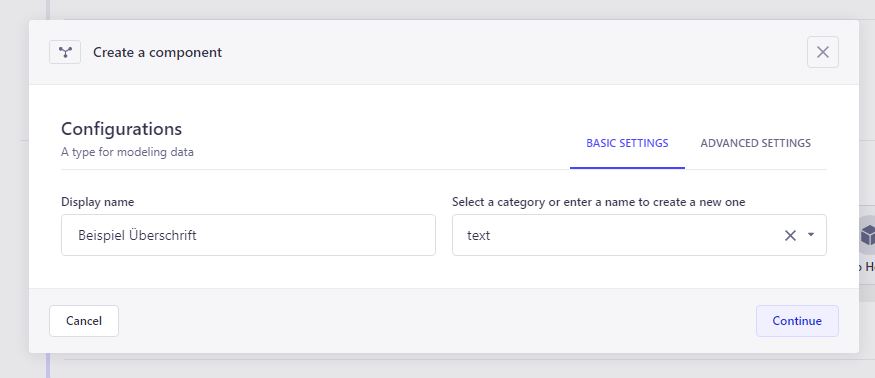
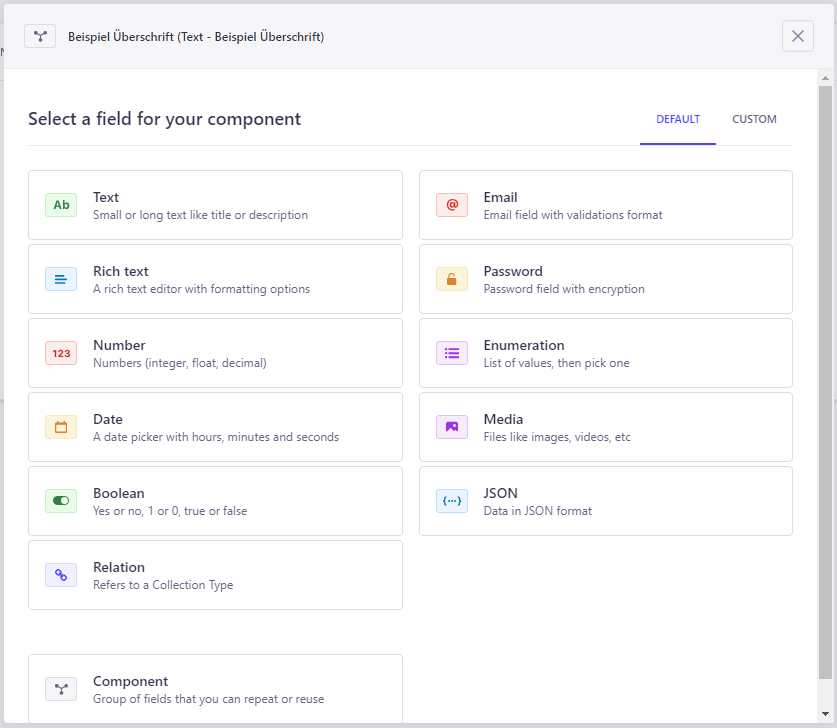


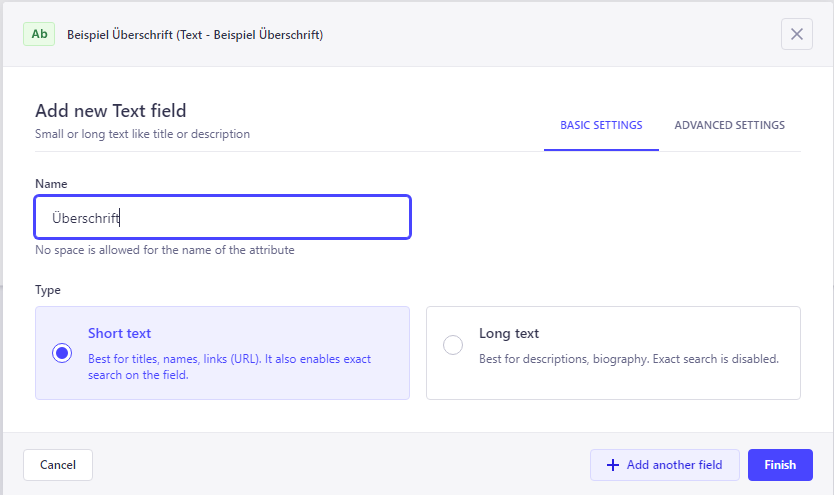


Anhang 6: Strapi Backend

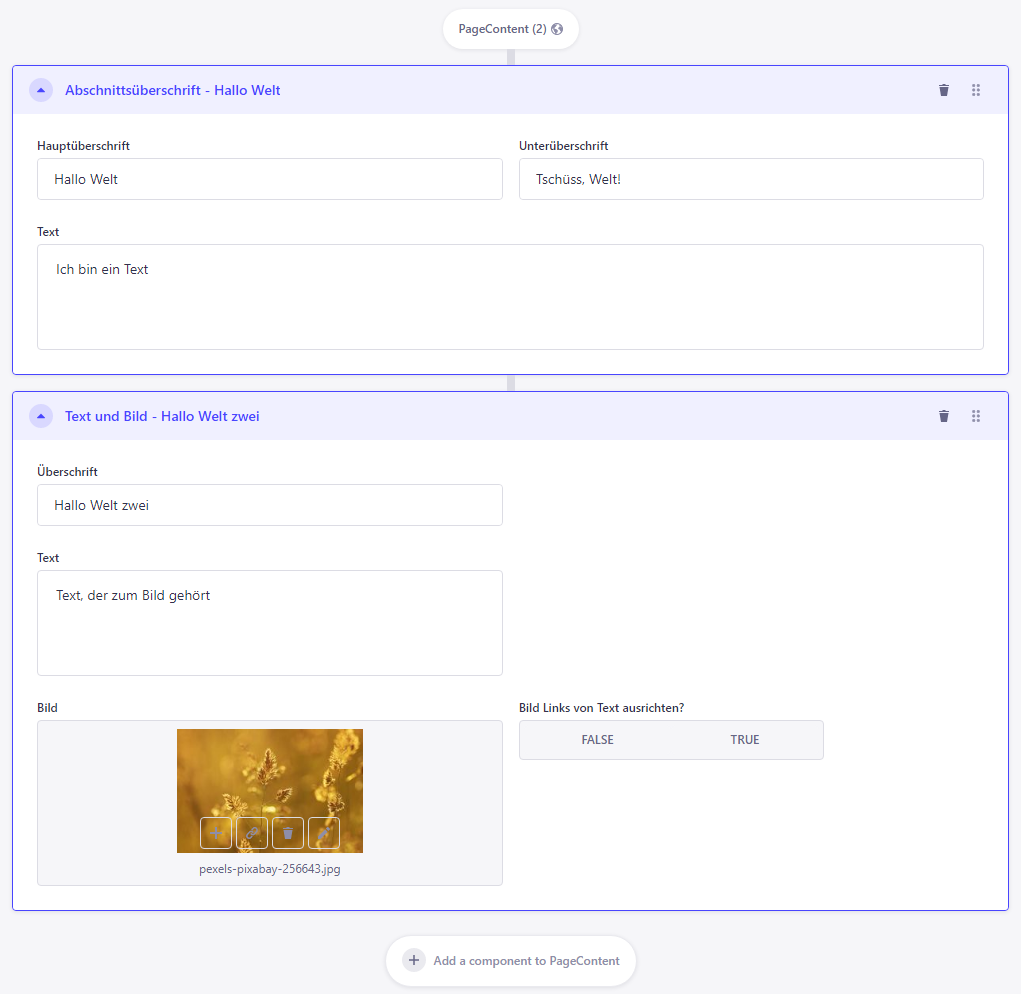


Anhang 7: Erstellen von Beispiel Element in Strapi

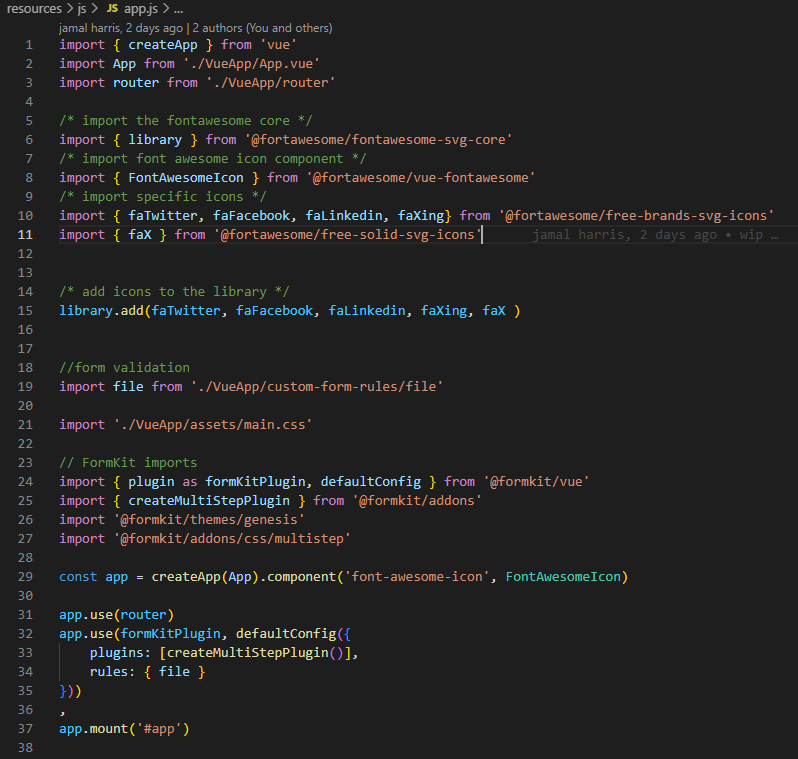
 



Anhang 8: Beispiel Seite mit Überschrift, Text und Bild BE und Ausschnitt von JSON-Antwort



Anhang 9: App.js



Anhang 10: ViteConfig.js



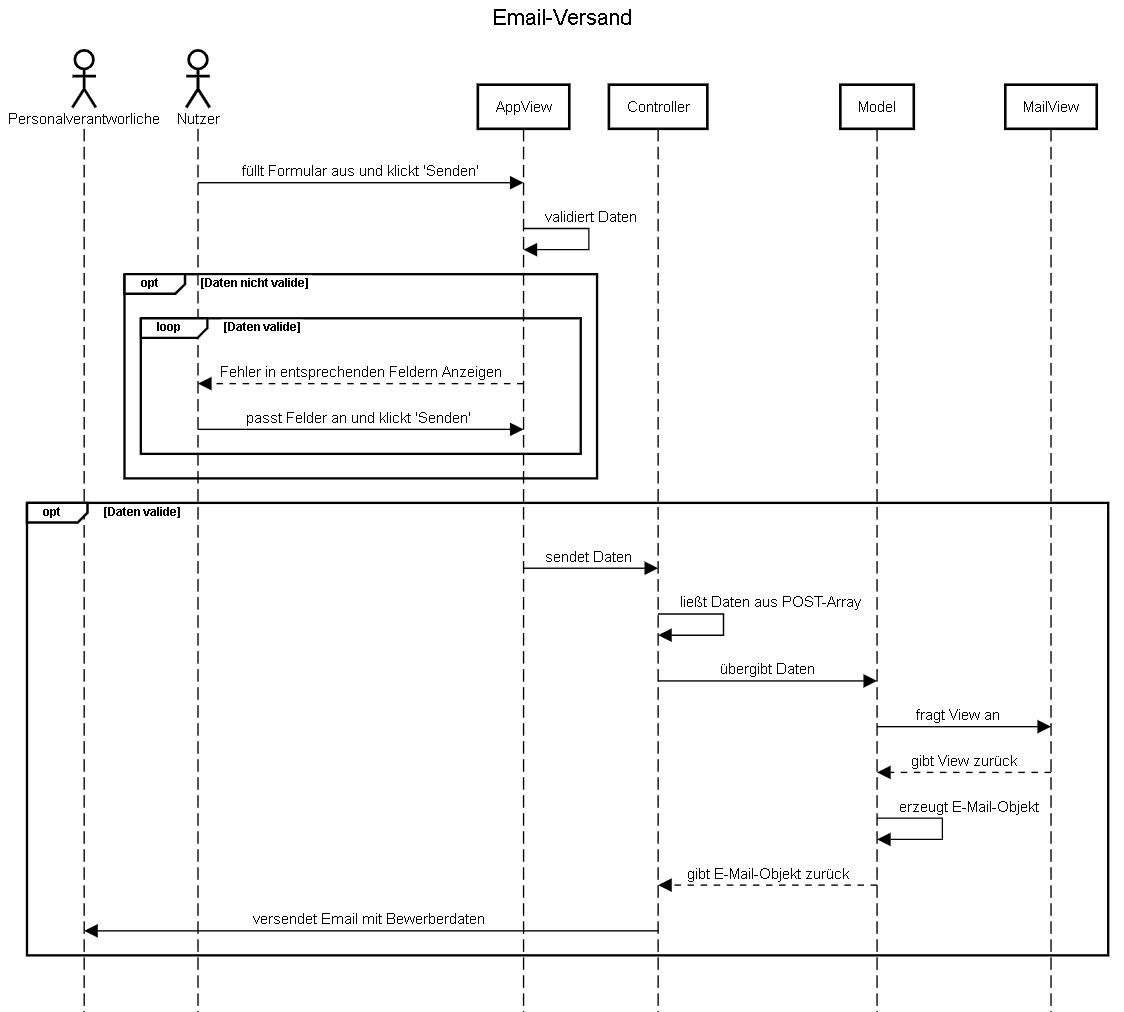
Anhang 11: Ausschnitt aus Mail-Model



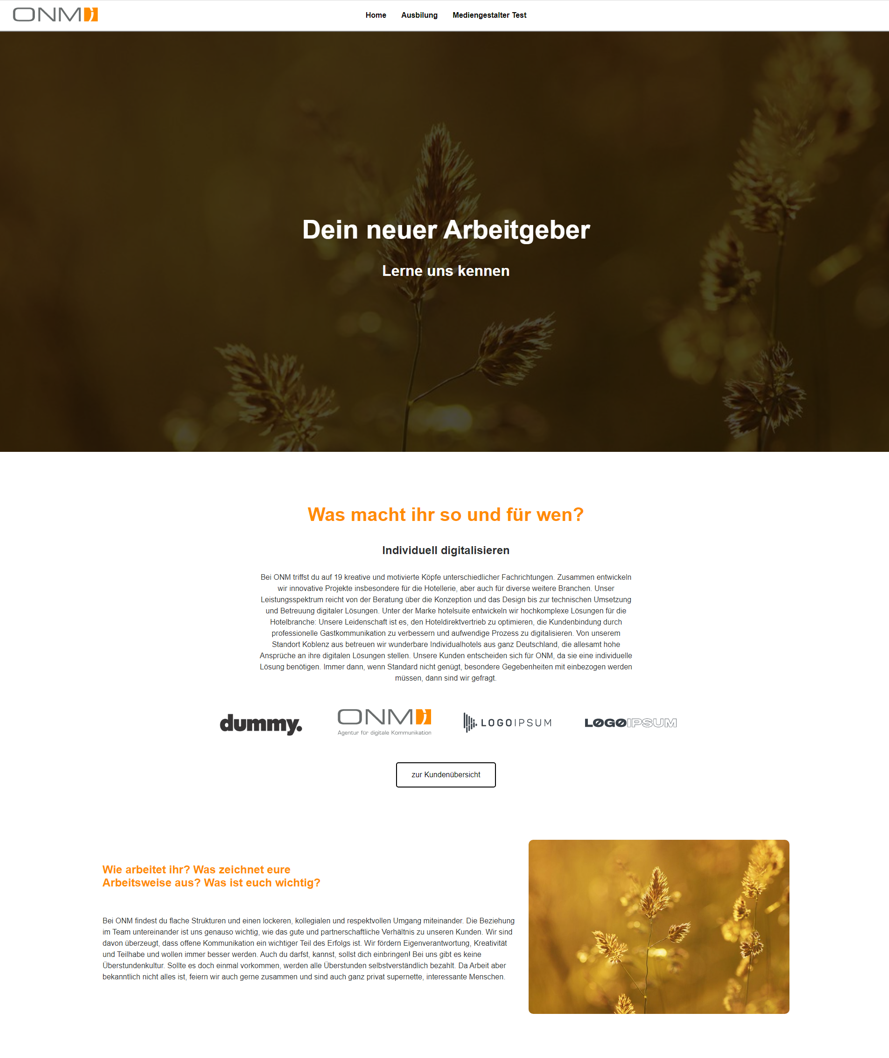
Anhang 12: Mail Controller

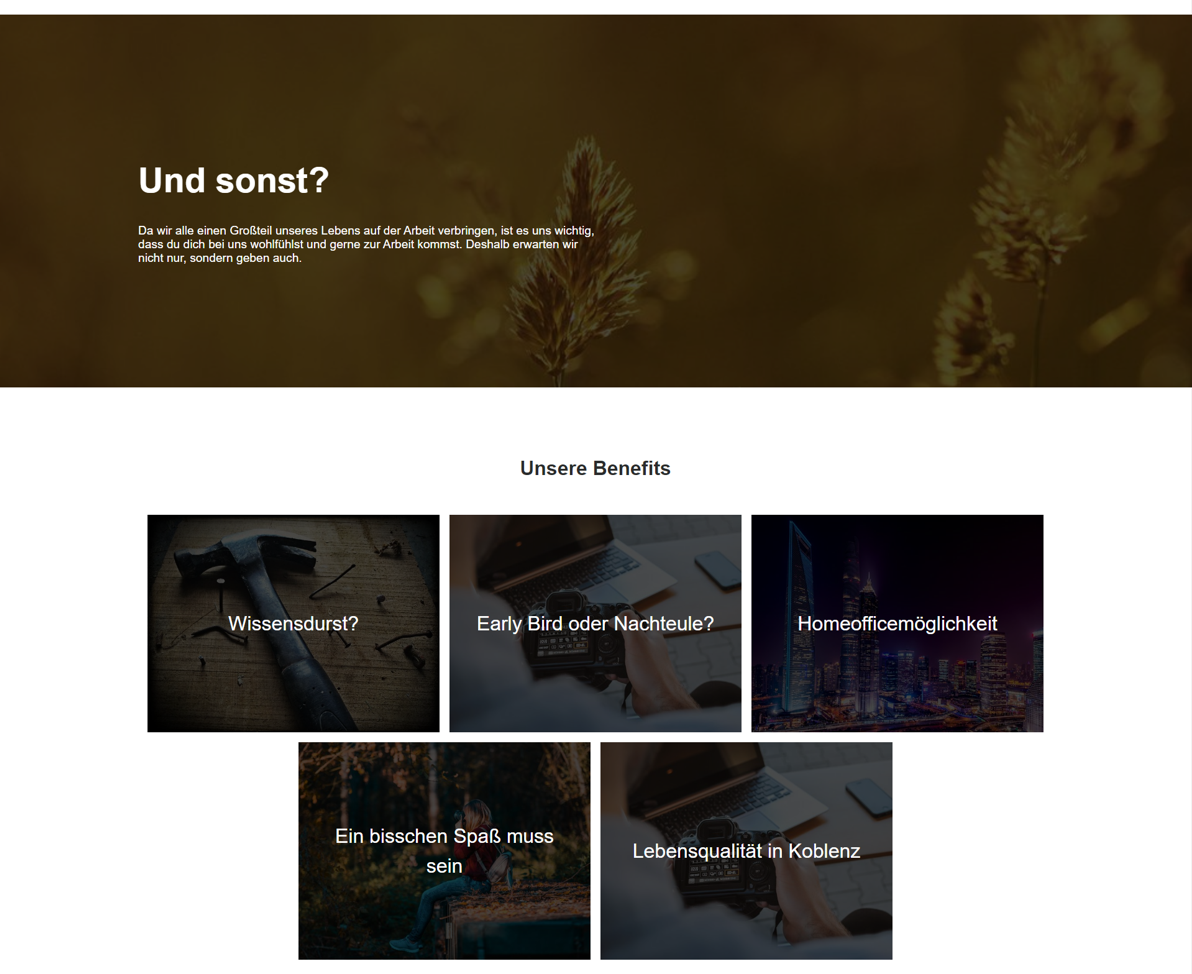


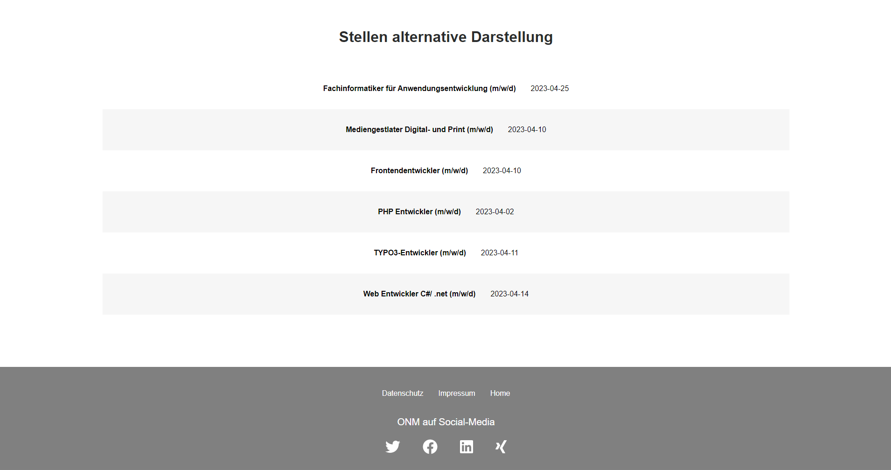
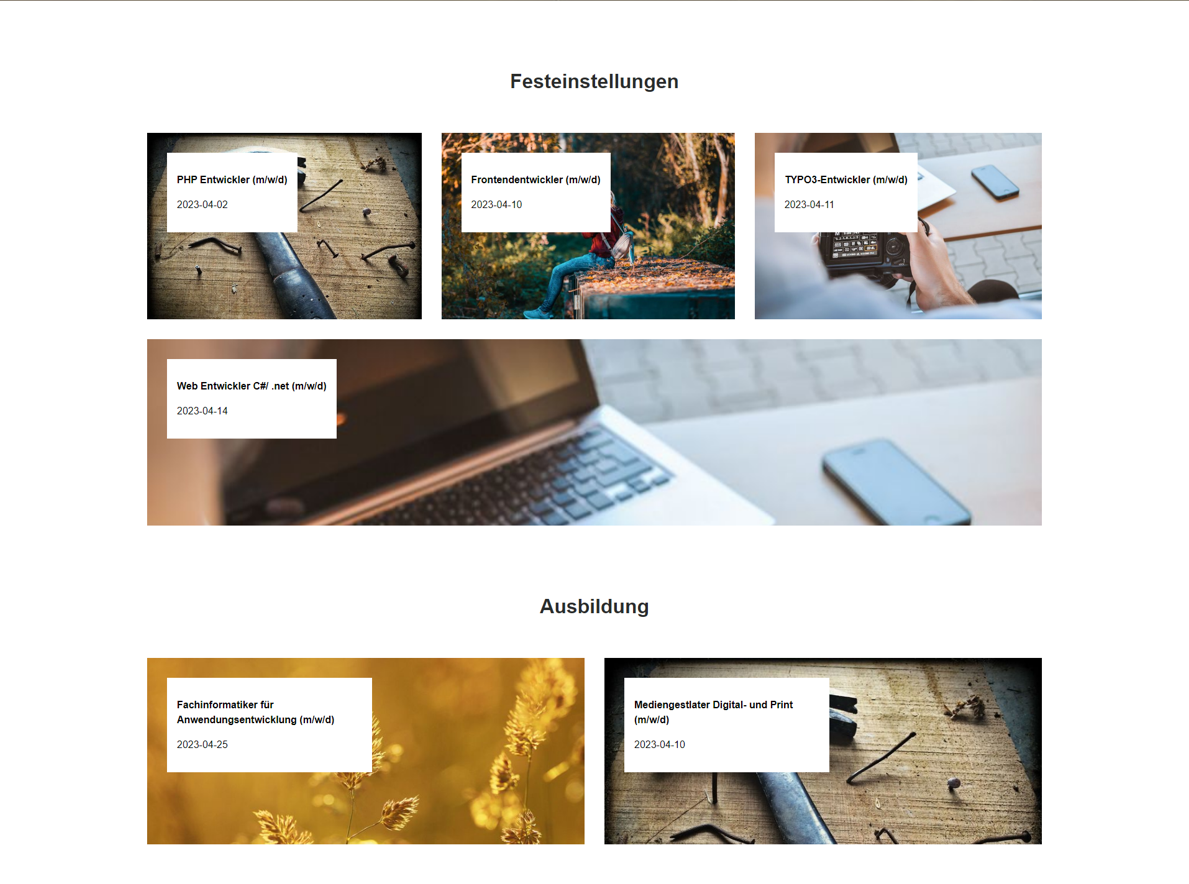
Anhang 13: Sequenzdiagramm E-Mails

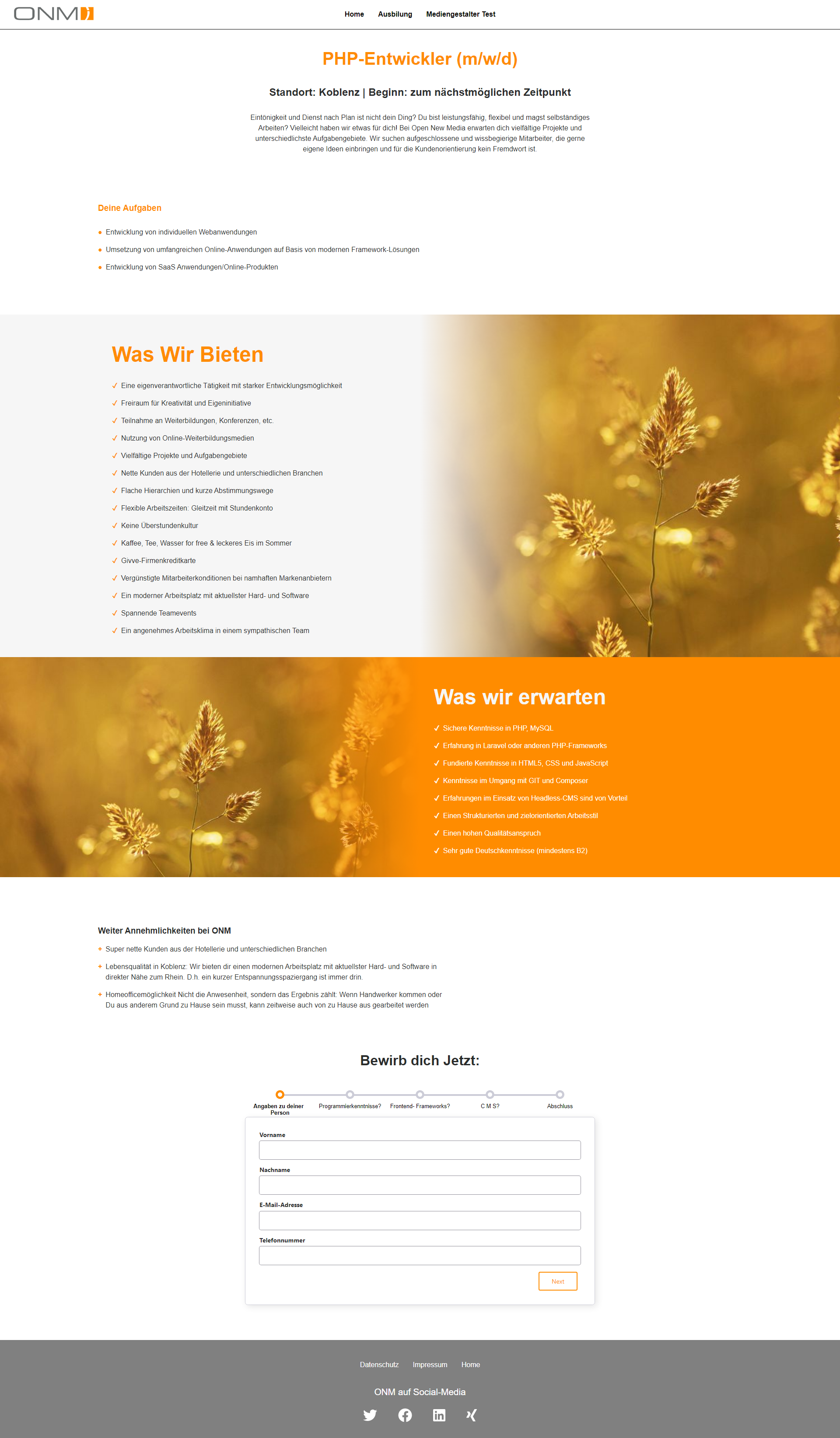


Anhang 14: Ausschnitte Design Umsetzung (Teil 1)

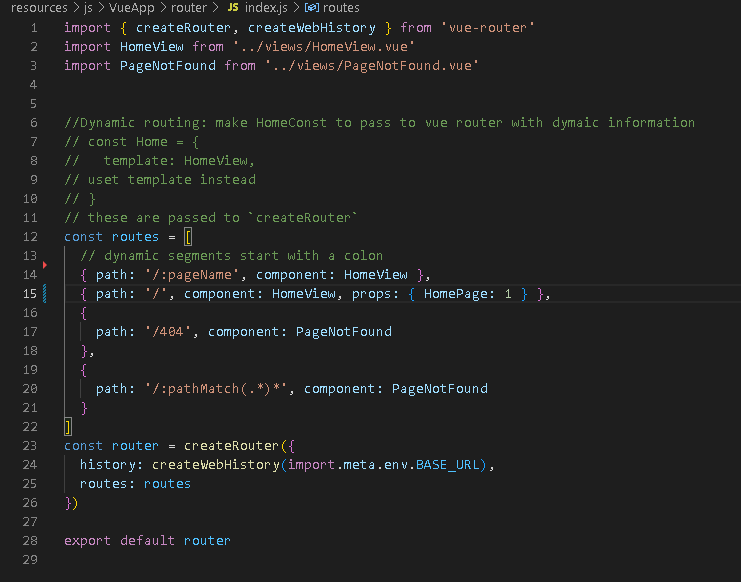




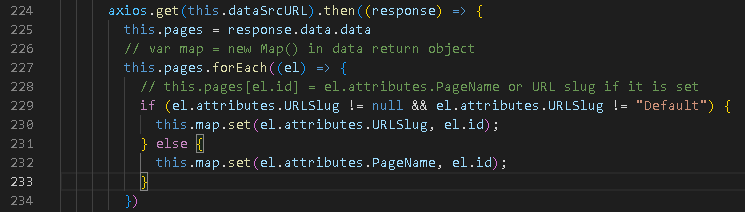




Anhang 15: Vue Router



Anhang 16: Erstellen von Seiten Map



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