Technische Hochschule
Georg-Simon Ohm Nürnberg
Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Projekt OHM News

Projektstudienarbeit
B-ME 5

OHM News - See only what matters

Wintersemester 2018/2019

Projekt-Roadmap

Tätigkeit	Dokument	Beteiligung
Konzeption	Projektidee	Alle
Organisation und Pflege	Trello	Erik
Organisation und Pflege	Bitrix24	Erik
Protokollierung der Treffen	Handschriftlich	Erik, Vivianne
Transkription der Protokolle	LaTeX Protokolle	Erik
Organisation und Pflege	Wiki	Vivianne
Erstformulierung	Pflichtenheft	Alle
Erste Entwürfe des App UI	Scribbles, Skizzen	Alle
Namensüberlegungen	Notizen zur Namensfindung	Alle
Corporate Design	Styleguide	Xenia
Entwicklung	Erstellung der Mockups	Xenia, Edwina
Entwicklung	Frontenddesign mit Bootstrap	Xenia, Edwina
Entwicklung	Single-Page mit Vue.js	Xenia, Edwina
Entwicklung	ServiceWorker	Erik
Entwicklung	Manifest mit Json	Erik
Entwicklung	Meta-Tags mit HTML	Erik
Entwicklung	Erstellung der API-Schnittstelle	Senta, Edwina, Vivianne, Xenia
Dokumentation RESTful API	Digital mit Swagger.io	Erik
Entwicklung	Server mit Node.js/Express.js	Vivianne
Entwicklung	Datenbank mit MongoDB	Senta
Entwicklung	DB-Anbindung mit Mongoose	Senta
Designentwürfe Plakat	Plakatgestaltung	Vivianne
Design und Ausformulierung	Projektpräsentation	Edwina, Vivianne

Tabelle 1: Projekt-Roadmap Tabelle

Inhaltsverzeichnis

Einleitung

ToDo

Vorwort zur Projektidee

Die erste Konzeptversion zu der Projektidee von dem Projekt ”Ohm Management App”, dass im Verlauf der ersten Projektphase in ”OHM-News - See only what matters” umbenannt wurde, wurde von Erik Römmelt erstellt. Nach gemeinsamer Überarbeitung durch Prof.Dr. Matthias Hopf, dem zukünftig betreuenden Professor und Erik Römmelt wurde die zu Projektbeginn gegebene Version des Projektkonzeptes ausformuliert.

Projektidee

Die Projektidee basiert darauf, dass eine plattformunabhängige App für Studierende an der TH Nürnberg entwickelt werden soll, die Kurznachrichten filtert und in einem Nachrichtenfeed anzeigt. Das Filtern funktioniert durch Tags. Es können beim Nachrichten erstellen mehrere, jedoch mindestens eine eingegeben werden. Diese werden bei einer Überschneidung bei der Suche durch eine logische UND-Verknüpfung eingeschränkt. Angezeigt werden die Nachrichten abhängig von ihrer Priorität und Aktualität.
Dazu können in einer Dateiablage von Professoren und Befugten wichtige Links oder andere Dateien gesammelt werden oder als Referenz darauf verwiesen werden. Damit es den Status ”Befugte” gibt, muss es zudem eine Rollenverteilung geben, wem welche Rechte zu stehen oder beispielsweise wer nur Nachrichten empfangen kann. Zudem ist eine Speicherung einzelner Nachrichten aus dem Feed möglich, welche im Nachhinein in den Bookmarks oder auch Lesezeichen einsehbar sind.
Des Weiteren können Push-Benachrichtigungen angezeigt werden, sodass das Risiko, wichtige Nachrichten zu übersehen, minimiert wird. Zuletzt gibt es noch ein Nutzerprofil. Hier können Daten zur Person eingetragen und angezeigt werden wie zum Beispiel das Profilbild, deren Status an der technischen Hochschule oder die Matrikelnummer. Für den Optimalfall einer Übernahme der App von der Hochschule soll modulorientiert und wartungsaufwandsarm gearbeitet werden, um die spätere Verwaltung zu erleichtern. Auch wird Wert gelegt auf eine nutzerfreundliche und möglichst intuitive Bedienoberfläche.

Technische Umsetzung

Zur Umsetzung des Projektes fiel die Zielsetzung auf die Entwicklung einer progressiven Web-App. Aufgrund ihrer Plattformunabhängigkeit und Kompatibilität mit den TH-Servern (VPN und Eduroam) und laut Erfahrungen von Professor Dr. Hopf stellt dies eine gute Lösung und sichere Wahl dar. Ebenfalls, was die Datenschutzvorkehrungen der Hochschule betrifft.
Es werden die Codesprachen HTML, CSS und JavaScript verwendet und als Frameworks Vue.js, Bootstrap und node.js.
Als Datenbank dient MongoDB und das zugehörige Framework mongoose.

Projektorganisation

Wöchentlich fand ein Treffen, zum Austausch und zur Besprechung zurückliegender und anstehender Aufgaben, statt. An diesem Treffen nahmen in der Regel alle Mitglieder dieser Projektgruppe und der Projektbetreuer Prof.Dr. Matthias Hopf teil. Konnten Teammitglieder, gleich welcher Gründe, an einem Treffen nicht teilnehmen, so haben alle sich unverzüglich über Absprachen und den neu anstehenden Aufgaben selbstständig erkundigt.

Trello

Zu Beginn des Projekts wurde entschieden, dass zur Verbesserung des Arbeitsablaufs ein Organisationstool hilfreich wäre. So fiel ist Trello in Gespräch gekommen.
Trello ist ein graphisch sehr gut aufbereitetes Projektmanagement-Tool in Form von Kanban-Boards. In der kostenfreien Version, ist es möglich ”unbegrenzt [viele] Boards, Listen, Karten, Mitglieder, Checklisten und Anhänge” hinzuzufügen und zu nutzen**trello:price**. Zur typografischen Gestaltung der Karten, um zum Beispiel eine Liste einzufügen ist die Nutzung von einigen Markdown-Markup Elementen möglich.
Dem Projekt zugute kommend ist die sehr einfache Bedienbarkeit, Plattform unabhängige Verfügbarkeit, sowie die gute Übersichtlichkeit der Boards. Nachteilig ist, dass es nur sehr eingeschränkt die Zeitkomponente abbilden kann. Dies ist nur über die sogenannten ”Plugins” möglich, wobei in der kostenfreien Version nur ein Plugin je Board zugleich genutzt werden kann. So kann einem Board entweder die Funktion ”Gantt-Diagramme nutzen” oder ”Storypoints den Karten hinzufügen” verwendet werden.

Diese Einschränkung führte zur der Entscheidung gegen die Nutzung von Trello im Team.

Bitrix24

Nach dem Ausschluss von Trello begann die Umsicht nach einem anderen Tool, dass kostenfrei mit einer Gruppe von fünf Personen genutzt werden kann. Und es sollte die Funktionen Gantt-Diagramme erstellen, Aufgaben an Mitglieder verteilen und den Gewichtungs-Punkte oder ”Storypoints” an Aufgaben zuweisen erfüllen.
Bitrix24 hat diese Anforderungen erfüllt, war jedoch bereits auf den ersten Blick deutlich unübersichtlicher und komplexer in der Handhabung. Da es sich bei Bitrix24 in erster Linie um ein umfangreiches CRM-System handelt, mit vielen weiteren Funktionen wie ”Aufgabenverwaltung für Gruppen”, ”Projektplanung und -management” und ”Team-Chat”**bitrix:feature**.
Auch nach der Deaktivierung und Ausblendung sämtlicher Funktionen, die die Nutzung von Bitrix24 komplizierter machten, blieb die Benutzeroberfläche noch unübersichtlich. Was die Umsetzung der Aufgaben als Kanban-Karten mit Gewichtungs-Punkten und der zeitlichen Darstellung der Aufgaben in einem Gantt-Diagramm betrifft, so war dies sehr angenehm und vollends zufriedenstellend möglich.
So wurden zahlreiche organisatorische Fristen, sowie Aufgaben organisatorischer und entwicklungstechnischer Natur in Bitrix24 erfasst und anschließend in einem wöchentlichen Treffen vorgestellt.
Im Weiteren wurde die Nutzung und Pflege des Tools an alle Mitglieder übertragen.
Es stellte sich am Ende des ersten Projektabschnittes heraus, dass das Tool Bitrix24 aufgrund der komplexen und unübersichtliche Benutzeroberfläche kaum genutzt wurde.

Entwicklerblog

Die Wiki Blogeinträge in dem hochschulinternen github Repository dienen dazu, die Projektfortschritte transparent zu gestalten und eine kurze prägnante Zusammenfassung für Professoren und Studierende zu gewährleisten. Einzusehen sind die Beiträge auf der git efi Seite **gitefi**.

Protokollierung

Für die Protokollierung wurden in regelmäßigen Intervallen Dokumentationen über die Tätigkeiten aller Teammitglieder geführt und diese folglich in Schrift festgehalten. Enthalten sind die Auf-

gabenaufteilung und die bisherige Umsetzung aller Projektteilnehmer in einer bestimmten Zeit, welche anschließend gebündelt zusammengefasst wurden.

Blogentwurf

Da das git efi nicht nur für Professoren sondern auch Studierenden der Fakultät efi zugänglich ist, werden die Vorgänge des Projekts in kurzen Sätzen und möglichst verständlich wiedergegeben. So können sich auch Studierende außerhalb des Fachgebietes ein ungefähres Bild von dem Fortschritt der OHM News-App machen. Nach dem Umformulieren und Reduzieren auf die Kernaussagen wurden Einträge in kontinuierlichen Abständen auf dem Hochschul github**gitefi** zum Abrufen hochgeladen. Diese berufen sich auf etwa ein Monats-Abstände oder kürzer. Bei neuen Einträgen wird Professor Dr. Hopf informiert, sodass er stets den Verlauf des Projektes nachvollziehen kann. In zeitlich chronologischer Reihenfolge aufgelistet, vereinfacht dies den Überblick. Insgesamt wurden im Laufe des OHM News-App Projektes sechs Wiki Blogeinträge erstellt.

Pflichtenheft

Das Pflichtenheft hat die Gruppe gemeinsam zusammengestellt. Dabei sollte der Datenzugriff per MongoDB erzeugt werden, alternativ könnte auch eine Schnittstelle zur Verfügung gestellt werden. Die Grafische UI der PWA sollte einen Login, ein Menü, eine Suche, ein Profil, einen Filter nach Hashtags, ein Dashboard, Einstellungen, das Nachrichtfeld und ein Formular zum Verfassen von Nachrichten umfassen. Ein Abhängigkeitsdiagramm soll erstellt werden. Die Anwendungslogik sollte im Bereich des Routings, der Darstellung und Speicherung von Nachrichten und dem LDAP-Login umgesetzt werden. Codetests sollten durchgeführt werden und das Rollenkonzept der Nutzer sollte festgelegt werden.
Des Weiteren sollte ein Anwendungsname gefunden und ein Logo erstellt werden. Ein UX Konzept sollte aufgestellt werden. Wenn es zeitlich noch möglich wäre, hätte man die Funktionen umsetzen sollen um die Datenbankabrufe zu filtern und Kanälen zu folgen. Ebenso hätte man die Anzeigekriterien festlegen können.

Konzepterstellung

Der erste Schritt zur Definition eines Layoutentwurfs war die Erstellung von Scribbles. Unter einem Scribble versteht man einen groben ersten Entwurf, der zur Ideenfindung beitragen soll. Nach Festlegung der grundsätzlichen Projektidee, zeichnete jedes unserer Teammitglieder Scribbles für verschiedene Seiten der App, die wir im Anschluss zusammen diskutierten. Wichtig war uns hierbei, dass jedes Teammitglied erst eigene Ideen entwickelte, um so viele verschiedene Aspekte zu sammeln.

Wir versuchten alle Ideen zu vereinen und einigten uns auf einen Layoutentwurf, welcher im Anschluss in einem detaillierten Screendesign umgesetzt wurde. In Abbildung 1 ist das Screendesign für die Startseite zu sehen. Wir legten vor allem den Fokus auf die Startseite, da diese der erste Baustein der App werden sollte. Die Navigation auf die anderen Seiten erfolgt über eine Leiste am unteren Displayrand. Die weiteren Seiten legten wir zu Beginn fest, arbeiteten diese allerdings erst mit der Zeit detaillierter aus. Ein genauer Aufbau der App wird unter 9.1. beschrieben.

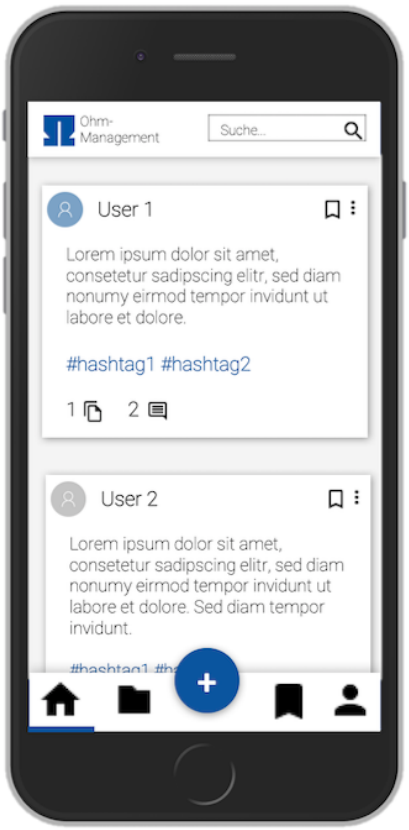


Abbildung 1: Layout

Namensfindung

Zu Anfang haben wir uns für den Titel OHM-Management entschieden, mit dem Wissen, dass es höchstwahrscheinlich nicht dabei bleibt, da das Wort Management mit unserer angestrebten Funktionalität der App nur einschränkend passte. Die Verwendung des Wortstammes Kommunikation war für unsere Benennung auch nicht zutreffend, insofern der Austausch von Information nur in eine Richtung erfolgt. Zusätzlich gestaltete sich die Namensfindung etwas schwierig, da wir sozusagen eine App für die TH-Nürnberg entwickelten, die man auch in Verbindung mit dem OHM-Zeichen kennt. Diesbezüglich wollten wir uns bei der Benennung auch an dieses anlehnen. Um dies tun zu können, mussten wir uns erst einmal informieren, ob es ähnliche Namen bereits gibt die man aus diesem Grund dementsprechend nicht verwenden kann.

Recherche

Bevor also der erste Schritt in Richtung Namensfindung gemacht werden konnte, recherchierten wir über bereits vorhandene Namen, die von der TH-Nürnberg genutzt werden. Neben Namen wie InfoTHek, OHMdoc und OHM-Shop, welche für unser Thema eher irrelevant wirkten, fanden wir auch den Titel OHM-News. Dies war ein Online Journal, welches jedoch, wie sich später rausgestellt hatte, eingestellt wurde.

Brainstorming

In der Zwischenzeit setzten wir jedoch die Ideensammlung weiter fort und präsentierten in der Gruppe einzelne Namensvorschläge. Hieraus entstand eine Liste an kreativen Gedanken. Unsere Favoriten :

- **ON AIR**
  - **O**HM **N**ews **A**pp for **I**nformation and **R**ediscovery/**R**ecommendations
  - **O**HM **N**ews Stream for **A**ctivities, **I**nformation and **R**ecommendations
- **ON STREAM**
  - **O**HM **N**ews **Stream**
- **OHM News**

- **TH i s**
- **INFOhm**

Alle Vorschläge waren sehr durchdacht und kreativ. Jedoch hatten einige nicht ganz die Wirkung die wir uns erhofften, denn wir waren der Meinung, dass man das Wort On Air eher mit einem Radiosender verbindet. Ebenfalls der Nachteil bei der Idee On Stream war es, dass man stark an Streaming Dienste wie zum Beispiel Netflix erinnert wird.

Ergebnis

Nach langem Überlegen kamen wir auf den Namen OHM News zurück und dieser überzeugte uns im Endeffekt alle. Dieser Titel war recht kurz, er ist leicht zu verstehen ohne viel nachzudenken, man verbindet ihn auf Anhieb mit der TH-Nürnberg auf Grund des Wortes OHM und er verweist auf die Funktionalität die unsere App bereitstellt, die da wären Neuigkeiten, sprich News, für die Studenten der Hochschule bereitzustellen.

Claim

Parallel zur Namensfindung, befassten wir uns auch mit dem Claim unseres Produktes. Da ein Claim die Funktion unserer App auf den Punkt bringen soll, und dabei kurz, leicht zu merken und aussagekräftig sein sollte, entschieden wir uns für eine Formulierung der Art ”nur relevante Themen werden angezeigt”. Nach einer kurzen Denkphase kamen wir auf den Claim ”See only what matters”, beziehungsweise ”Only see what matters” vor. Dies kam bei uns allen sehr gut an, jedoch wussten wir nicht, welche von den beiden Optionen die erhoffte Bedeutung am besten hervorbringt. Sie haben zwar die selbe Bedeutung, trotz alle dem empfanden wir, dass die Betonung des Wortes „Only“ auf zwei verschiedene Faktoren des Satzes liegt. Nach reichlicher Überlegung entschieden wir uns also für die Formulierung ”See only what matters” als Claim für unsere App.

Corporate Design

Um das ganze Konzept stimmig zu gestalten, wurde ein Corporate Design erstellt. Wichtigste Grundlage hierfür ist das Farbkonzept. Da die App im Hochschulrahmen verwendet werden soll,

wurde für die Grundfarbe der Blauton des Hochschullogos gewählt, umso eine Zugehörigkeit zu symbolisieren. Ergänzend zu diesem Blauton werden noch zwei dezente Grautöne verwendet, wie in Abbildung 2 zu sehen ist.

Abbildung 2: Farbkonzept

Für die Schrift wird in der gesamten App die von Google Material Design vorgegebene Schriftart Roboto verwendet. Diese ist eine serifenlosen Schriftart und deshalb durchweg gut lesbar. Auch ein Logo ist wichtig, um einen gewissen wiedererkennungswert für die App zu schaffen. Da wir uns in der Gruppe für den App-Namen ”Ohm-News” entschieden haben, war der erster Gedanke eine Zeitung für das Logo herzunehmen (siehe Abbildung 3). Die Grafik war allerdings schwierig in das Produkt einzubinden, weil sie sehr detailreich ist.

Abbildung 3: Erster Logoentwurf

Ein zweiter etwas minimalistischer Entwurf ist in Abbildung 4 zu sehen, für welchen sich das Projektteam schlussendlich auch entschied. Es zeigt ein Globussymbol mit dem Hochschullogo im Zentrum, um auch hier wieder die Verbindung zur Hochschule schaffen.

Abbildung 4: Finales Logo

Das Logo wurde dann auch zu einem App Icon erweitert. In Abbildung 5 ist das Icon für Android sowie IOS Geräte zu sehen. Das Android Icon wurde analog den Material-Design Richtlinien gestaltet und besitzt deshalb einen leichten Schatten. Bei dem IOS Icon wurde das Logo lediglich auf einen einfarbigen Hintergrund gesetzt.

Abbildung 5: App Icons

Erstellung der Mockups

Auf Basis des Screendesigns wurden erste Mockups erstellt, welche mittels HTML und CSS umgesetzt wurden. Um die Elemente entsprechend den Material-Design Richtlinien zu gestalten, wurde das CSS-Framework Bootstrap mit eingebunden, das eine Auswahl an vordefinierten Gestaltungselementen und Stilen in Form von CSS-Stylesheets bereitstellt. Dazu gehören Elemente wie Buttons, Formulare, Navigationszeilen und viele mehr. Wir verwendeten Bootstrap Dateien, die auf Material Design abgestimmt sind. So wird ein einheitliches Design gewährleistet und die Styles müssen nicht selbst erstellt werden. Ein weitere Vorteil von Bootstrap, ist, dass es ein

responsives Webdesign unterstützt und man deshalb plattformunabhängig entwickeln kann. Dafür sorgt ein Grid-Layout, welches den Benutzerbildschirm in 12 Spalten teilt und die Elemente, je nach verfügbarer Auflösung, anzeigt. Im ersten Projektabschnitt legten wir den Fokus besonders auf eine mobile Ansicht. Eine Desktopansicht wird ebenfalls unterstüzt, kann allerdings noch stark optimiert werden. **bootstrap**

Durch die Ergänzung von eigenen CSS-Dateien konnten Änderungen an den vordefinierten Stilen vorgenommen werden. Sowohl unsere als auch die von Bootstrap bereitgestellten Dateien, wurden auf Grundlage von Less-Stylesheets erstellt. Dabei wird auf effiziente Weise CSS generiert. Variablen sowie Funktionen sind möglich und Vererbung wird durch verschachtelte Selektoren dargestellt. **less**

Wir habe für jeden Menüpunkt eine HTML-Seite erstellt, welche im Unterpunkt genauer beschrieben werden.

Aufbau der App

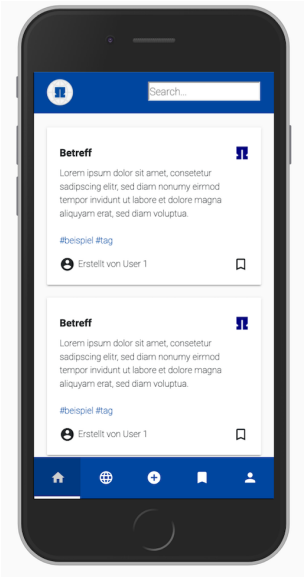


Abbildung 6: Startseite

Auf der Startseite der App gelangt der Nutzer als erstes auf einen Newsfeed. Der Newsfeed besteht aus einzelnen Nachrichten in Form von Karten und soll für jeden Nutzer nur relevante Nachrichten anzeigen. Jede Nachricht besitzt einen oder mehrere Tags. Eine Nachricht wird für einen Nutzer als relevant eingestuft, wenn dieser mindestens einen Tag aus der Nachricht abonniert hat. Somit erhält jeder Nutzer eine individuelle Startseite. Zusätzlich befindet sich auf der Nachrichtenkarte oben rechts ein Icon, welches die Nachricht zu einer Fakultät zuordnet. Unten rechts kann das Lesezeichen-Icon geklickt werden, um eine Nachricht zu speichern. Auf diese Weise können die Studierenden wichtige Informationen schnell wiederfinden.

Im Header ist die Suche zu finden, in der Tags explizit gefiltert werden können. Der Header ist von jeder Seite aus zugängig. Im Footer befindet sich die Navigationszeile, von wo aus der Nutzer auf die weiteren Seiten gelangt.

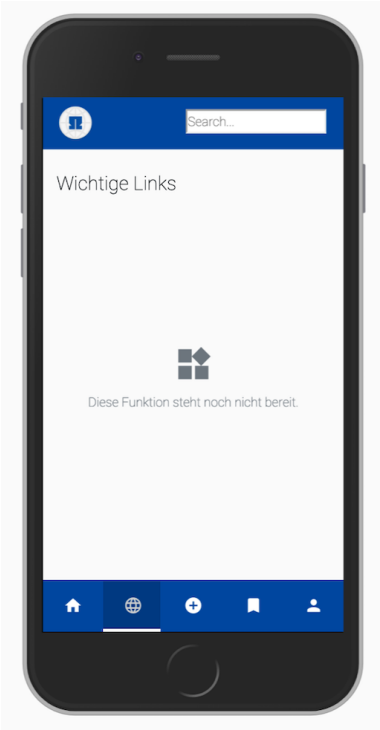


Abbildung 7: Infoseite

Der zweiten Menüpunkt, der durch ein Globus-Icon dargestellt wird, soll eine Informationsseite beinhalten. Das Konzept hierfür ist allerdings noch nicht konkret ausgearbeitet, da wir uns zuerst auf die Mitteilungsfunktionen konzentrieren wollten. Aktuell wird ein Empty State angezeigt.

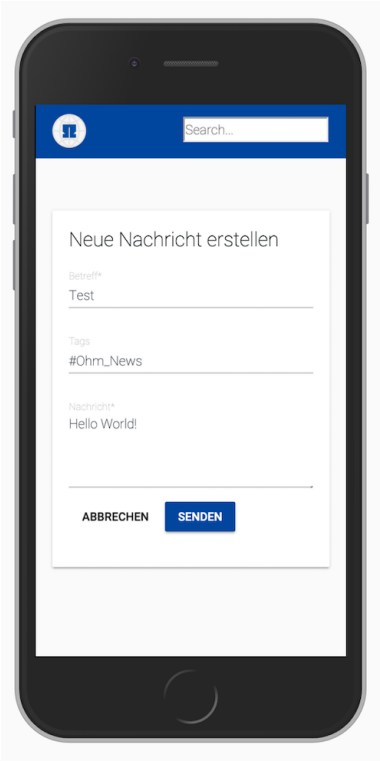


Abbildung 8: Formular

Über das Plus-Icon, in der Mitte der Navigationszeile, gelangt man zu einem Formular. Hier können neue Nachrichten erstellt werden. Nachrichten sollen nur von ausgewählten Nutzergruppen, wie Fakultäten, Studienbüro und ähnlichen gesendet werden können. Für eine Nachricht wird ein Betreff, mindestens ein Tag und der Nachrichteninhalt benötigt. Es soll eine Auswahl an vordefinierten Tags geben, damit am Ende nicht jeder Verfasser wahllos Tags setzten kann und Duplikate vermieden werden. Dieser Menüpunkt ist der einzige in dem keine Navigationszeile angezeigt wird. Nach Absenden einer Nachricht, gelangt man zurück auf die Startseite. Über den Button 'Abbrechen' wird man auf den vorherigen Menüpunkt zurückgeleitet.

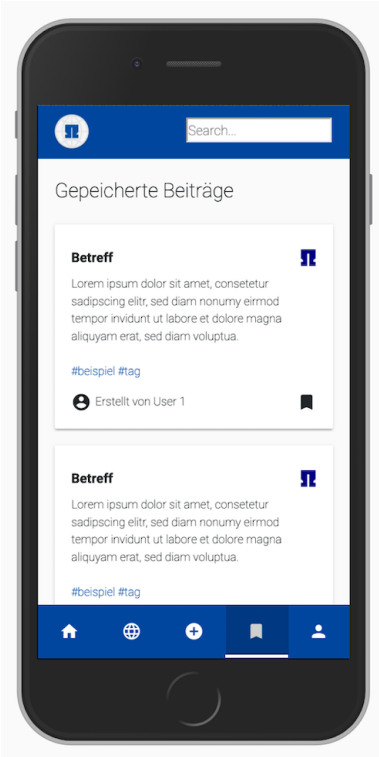


Abbildung 9: Gespeicherte

Der nächste Menüpunkt wird über das Lesezeichen-Icon erreicht und zeigt die gespeicherten Nachrichten an. Diese Seite ist ähnlich aufgebaut wie der Newsfeed der Startseite, beinhaltet allerdings nur die vom Nutzer gespeicherten Nachrichten.

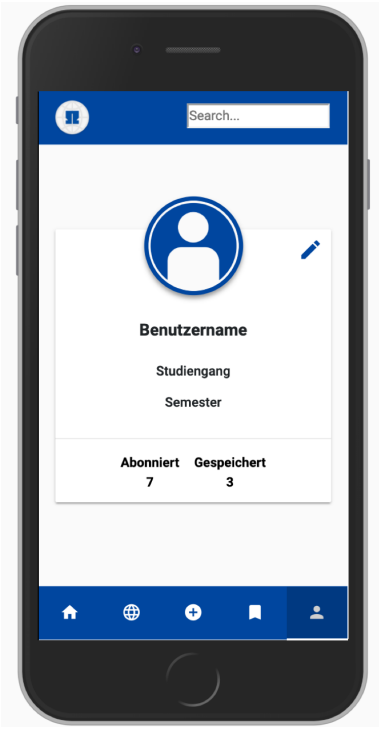


Abbildung 10: Profil

Der letzte Menüpunkt ist das Profil.

Oben sieht man zuerst das Profilbild, hier verwendeten wir die runde Form anstatt der üblichen Quadratischen. Darunter sieht man dann die im Vorfeld angegebene Information des Benutzers. Am Ende der sogenannten Karte befinden sich zwei Knöpfe, die Anzeigen wie vielen Kanälen man folgt und wie viele Beiträge man bis jetzt gespeichert hat. Auf der Karte rechts oben im Eck findet man den Bearbeitungs-Knopf, in Form eines kleinen Stiftes, dies ermöglicht dem Benutzer, die Seite gemäß seinen Wünschen anzupassen.

Single-Page Entwicklung

Neben klassischen Webanwendungen gibt es ebenfalls Single-Page Webanwendungen. Bei klassischen Webapplikationen gibt es mehrere untereinander verlinkte HTML-Dokumente, die dann gemeinsam eine komplette Anwendung ausmachen. Der Nachteil hierbei ist, dass die Seite beim Wechsel durch die erneute Kommunikation mit dem Server neu heruntergeladen werden muss. SPA's jedoch bestehen nur aus einer HTML-Datei. Dies bewirkt, dass neue oder veränderte Daten dynamisch geladen werden und somit die Kommunikation zwischen Server und Client verringert wird. Es ermöglicht dem Benutzer, nach dem einmaligen laden der Seite, diese auch weiterhin im offline Modus zu steuern. Sollte also die Interaktion mit dem Server unterbrochen werden, werden die zwischengespeicherten Daten einfach aus dem Webbrowser verwendet.**spa:1spa:2**

Warum Vue.js?

Neben dem Vue.js Framework gibt es noch weitere wie zum Beispiel react.js, angular.js, meteor.js und viele mehr. Warum wir uns jedoch für das Vue.js Framework entschieden haben, werden wir Ihnen nun erläutern. Vue.js das jüngste von den drei größten Frameworks, es hat jedoch den Vorteil, dass es aus den Fehlern der Vorgänger lernen konnte und diese schon im Vorfeld überdenken und ausmerzen konnte. Da es auf Sprachen wie HTML, CSS und JavaScript aufbaut, lässt es sich problemlos in schon bestehende Projekte einbauen. Die Syntax ist hierbei, im Vergleich zu anderen, besser lesbar. Hierfür werden nämlich sogenannte "Single File Components" verwendet.**vue:warum** Sogenannte Komponenten sind wiederverwendbare Vue-Instanzen mit einem Namen, die als benutzerdefiniertes Element in einer Vue-Stamminstanz verwendet werden. Wie so eine erstellte Komponente aussieht, werden wir Ihnen im folgenden erörtern.

Vue.js in vorhandene Mockups einbinden

Nach reichlicher Recherche und dem einlesen in das uns vorher unbekannte Framework, war es an der Zeit unsere HTML-Seiten in Vue-Komponenten umzuwandern. Wir setzten uns erstmal zusammen und erstellten die Startseite. Im Folgenden wird der Aufbau der Home-Komponente näher erläutert. In Listing 1 ist der dazugehörige Quellcode zu sehen. Nicht enthalten sind zur Komponente zugehörige Methoden. Die Home-Komponente wird in unserem Projekt als Home-Router betitelt.

```
const HomeRouter = {
    template: `
<div id="om-msg-cards">
    <MsgCard
        v-for="id in messagelist.slice().reverse()"
        :key="id"
        :msg="messages[id] || {}"
    ></MsgCard>
</div>`,
//Restliche Komponente
        ...
```

Listing 1: HomeRouter-Komponente

Im Newsfeed, welcher der Hauptbestand der Startseite ist, wird eine Liste von Nachrichten dynamisch generiert, sodass jeder Nutzer später einmal eine individuelle Ansicht erhält, abhängig von den Tags, die dieser abonniert.
Jede Komponente besitzt ein Template in Form eines Strings, welcher HTML-Code beinhaltet. Die Komponente HomeRouter ist verschachtelt und enthält die Komponente MsgCard. Der Quellcode für diese Komponete ist in Listing 2 zu sehen.

```
Vue.component('MsgCard', {
    template: `<div class="om-card card">
                <h6 class="msg-head">
                <b>{{ msg.subject }}</b>
                <img src="favicon.ico" width=20px height=20px>
                </h6>
                {{ msg.message }}<br><br>
                <a href="#">{{ msg.tag }}</a></p>
                <div class="om-card-footer"> <div class="om-user-line">
                <i class="material-icons">account_circle</i>
                Erstellt von {{ msg.user }}</div>
                <i class="material-icons">bookmark_border</i>
```

```
13                 </div></div>`,
14     props: ['msg']
15 });
```

Listing 2: MsgCard-Komponente

Eine MsgCard stellt eine einzelne Nachricht in Form einer Karte dar. In der HomeRouter-Komponete geht eine For-Schleife alle vorhandenen Nachrichten-Ids durch. Dabei werden der MsgCard-Komponente alle relevanten Informationen übergeben, die für eine Darstellung benötig wird. Am Ende stellt die App eine fortlaufende Liste mit allen Nachrichten dar.

Da wir nun ein bisschen geübter im Umgang mit Vue waren, teilten wir uns die restlichen Seiten auf. Dabei gingen wir ähnlich vor, wie bei der Home-Komponente.

Routing

Vue.js bietet eine Reihe von Funktionen, mit denen Sie wiederverwendbare Webkomponenten erstellen können. Routing ist eine dieser Methoden. Der Benutzer kann somit zwischen den Seiten wechseln, ohne diese erneut zu aktualisieren. Dies ermöglicht dem Benutzer eine einfache und dynamische Navigation durch die Webanwendung. Wir werden Ihnen nun, anhand unseres Codes erklären, wie ein Vue.js-Router funktioniert.**vue:router**
Zunächst richten wir hierfür die Navigation über die Navigationsbar ein, die wir mittels einem Router-Link-Element erstellt haben. Dieser gibt an, auf welche der angegebenen JS-Datei zugegriffen wird sobald man auf das ausgewählte Item klickt.

```
1 <nav class="nav nav-tabs nav-justified om-nav" v-if="$route.path !=='/
      createMessage' ">
2     <router-link to="/home" class="nav-item nav-link">
3         <i class="material-icons">home</i></router-link>
4     <router-link to="/files" class="nav-item nav-link">
5         <i class="material-icons">language</i></router-link>
6     <router-link to="/createMessage" class="nav-item nav-link outlined">
7         <i class="material-icons">add_circle</i></router-link>
8     <router-link to="/bookmark" class="nav-item nav-link">
```

```
9         <i class="material-icons">bookmark</i></router-link>
10    <router-link to="/profil" class="nav-item nav-link">
11        <i class="material-icons">person</i></router-link>
12 </nav>
```

Listing 3: Navbar

Im Folgenden werden Komponenten definiert und dann in ein neues Array von Objekten, so genannte Routen, eingefügt. Zu beachten ist die Zuordnung eines URL-Pfads zu der dazugehörigen Komponente. Daraufhin wird ein Vue-Router definiert, der an ein neues Vue-Objekt übergeben wird. Wenn nun beispielsweise ”/home” geladen ist, wird die Home-Komponente in der Router-Ansicht gerendert und aufgerufen. Die erste Zeile mit dem ”/” steht für die Startseite, auf die man gelangt, sobald man die App öffnet.**vue:router3**

```
1  const routes = [
2      { path: "/", component: HomeRouter },
3      { path: "/home", component: HomeRouter },
4      { path: "/files", component: FileRouter },
5      { path: "/createMessage", component: CreateMsgRouter },
6      { path: "/bookmark", component: BookmarkRouter },
7      { path: "/profil", component: ProfilRouter },
8  ];
9  const router = new VueRouter({
10     routes
11 });
12 var app = new Vue({
13     router,
14     el: '#api',
15     methods: {
16         ...
17     },
18 });
```

Listing 4: Routing

Service Worker

Der ServiceWorker ist eine JavaScript Datei, die üblicherweise im root-Verzeichnis der Webanwendung liegt. Für die Funktionsweise ist die Namensgebung der Datei nicht maßgebend. Damit der ServiceWorker überhaupt funktionieren kann, wird eine SSL-Verbindung benötigt. Ist eine sicher Verbindung gegeben, kann der ServiceWorker (im Weiteren auch mit ”SW” abgekürzt) nach erfolgreichem Laden der Webseite im Browser des Clients registriert werden. Nachdem der SW zeitlich unabhängig, also asynchron im Browser läuft, muss bei der Entwicklung und Umsetzung des ServiceWorker-Codes darauf geachtet werden, dass in Promises geschrieben wird. Da Promises ein wenig schwieriger nachzuvollziehen sind und insbesondere das Problem der sogenannten Callback-Hell nicht lösen, bietet es dich an den SW in Async/Await Struktur zu schreiben. Diese basiert auf dem asynchronen Prinzip der Promises und lässt sich dennoch lesen wie synchrone Funktionen.
Der Code für die Registrierung des ServiceWorkers im Browser:

```
1  /* Promise / Then */
2  if ('serviceWorker' in navigator) {
3      window.addEventListener('load', function() {
4          navigator.serviceWorker.register('/serviceWorker.js').then(function(
               registration) {
5              // Registration was successful
6          }, function(err) {
7              // registration failed
8          });
9      });
10 }
```

Listing 5: Register, Promise/Then-Codestruktur

```
1  /* Async / Await */
2  const registerServiceWorker = async () => {
3      try {
```

```
4         const registration = await navigator.serviceWorker.register('/
              serviceWorker.js', {scope: '/'});
5         // Registration successful
6     } catch (err) {
7         // Registration failed
8     }
9     return;
10 }
11 if ('serviceWorker' in navigator) {
12     window.addEventListener('load', registerServiceWorker());
13 }
```

Listing 6: Register, Async/Await-Codestruktur

Einmal in der Promise.then Schreibweise und einmal in der Async/Await Schreibweise. Beide Codes machen exakt das selbe.

Nach der Registrierung folgt die Installation. Das ist der Code dazu:

```
1  /* Promise / Then */
2  const cacheName = 'appname-v1-0';
3  const filesToCache = [ '/index.html' ];
4  self.addEventListener('install', function(event) {
5      console.log('[ServiceWorker] Install');
6      event.waitUntil(
7          caches.open(cacheName).then(function(cache) {
8              console.log('[ServiceWorker] Caching app shell');
9              return cache.addAll(filesToCache);
10         })
11     );
12 });
```

Listing 7: Install, Promise/Then-Codestruktur

```
/* Async / Await */
const installNewCache = async (event) => {
    const cacheName = 'appname-v1-0';
    const filesToCache = [ '/index.html' ];
    const cacheStatic = await caches.open(cacheName);
    cacheStatic.addAll(filesToCache);
    console.log('[ServiceWorker] Cache static files.');
    return;
}
self.addEventListener('install', event => {
    // don't wait
    self.skipWaiting();
    // cache static files
    event.waitUntil(installNewCache());
    console.log('[ServiceWorker] Install');
});
```

Listing 8: Install, Async/Await-Codestruktur

Während des ’install’ Events wird eine neue Version des ServiceWorkers im Browser installiert, sofern eine Netzwerkverbindung besteht. Der nächste Schritt ist die Aktivierung:

```
/* Promise / Then */
self.addEventListener('activate', function(event) {
    // ServiceWorker activated
    event.waitUntil(
        caches.keys().then(function(cacheKeyList) {
            return Promise.all(
                cacheKeyList.map(function(cacheKeyList) {
                    if (cacheWhitelist.indexOf(cacheKeyList) === -1) {
                        // Old Cache removed
                        return caches.delete(cacheKeyList);
                    }
                })
```

```
            );
        })
    );
    return self.clients.claim();
});
```

Listing 9: Activate, Promise/Then-Codestruktur

```
/* Async / Await */
const cacheCleanUp = async () => {
    const cacheKeyList = await caches.keys();
    const deletions = cacheKeyList
    .filter(key => key.startsWith(appPrefix) && !cacheKeyList.includes(key))
    .map(key => {
        caches.delete(key)
        // Old Cache removed
    });
    for (const success of deletions) {
        await success;
    }
    return;
}
self.addEventListener('activate', event => {
    event.waitUntil(cacheCleanUp());
    clients.claim();
    // ServiceWorker activated
});
```

Listing 10: Activate, Async/Await-Codestruktur

Hier werden sämtliche alte ServiceWorker Dateien aus dem Cache gelöscht und alte SW deinstalliert und unregistriert. Nun bleibt nur noch ein Schritt aus, das ist der ’fetch’:

```
/* Promise / Then */
self.addEventListener('fetch', function(event) {
    /* We should only cache GET requests */
    if (event.request.method !== 'GET') {
        /* If we don't block the event as shown below,
        then the request will go to the network as usual. */
        console.log('[ServiceWorker] Fetch event ignored.',
        event.request.method, event.request.url);
        return;
    }
    event.respondWith(
    caches.match(event.request)
    .then(function(response) {
        return fetch(event.request)
        .then(function(response) {
            // Check if response is valid, status is 200, response type is basic
            // (indicates request is from origin, means that requests to third
                party
            // assets aren't cached as well.
            if(!response || response.status !== 200 || response.type !== 'basic')
                {
                return response;
            }
            caches.open(CACHE_NAME)
            .then(function(cache) {
                // We have to clone the response here because request bodies can
                    only
                // be read once. Placing a response in the cache counts as a read.
                cache.put(event.request, response.clone());
            });
            // Cache hit - return response
            if (response) return response;
                return response;
            });
```

```
32          })
33      .catch(function(err) {
34          // Report a lack of connectivity to the user.
35      });
36  });
```

Listing 11: Fetch, Promise/Then-Codestruktur

```
1   /* Async / Await */
2   self.addEventListener('fetch', event => {
3       /* We should only cache GET requests */
4       if (event.request.method !== 'GET') {
5           /* If we don't block the event as shown below,
6           then the request will go to the network as usual. */
7           console.log('[ServiceWorker] Fetch event ignored.',
8           event.request.method, event.request.url);
9           return;
10      }
11      event.respondWith(async function update() {
12          try {
13              var requestURL = new URL(event.request.url);
14              // Start the network request as soon as possible.
15              const networkPromise = fetch(requestURL);
16              const cachedResponse = await caches.match(event.request);
17              const networkResponse = await networkPromise;
18              // Check if response is valid, status is 200, response type is basic
19              // (indicates request is from origin, means that requests to third
                    party
20              // assets aren't cached as well.
21              if (!networkResponse || networkResponse.status !== 200
22              || networkResponse.type !== 'basic') return networkResponse;
23              const cache = await caches.open(staticCacheKey);
24              // We have to clone the response here because request bodies can only
25              // be read once. Placing a response in the cache counts as a read.
```

```
26              cache.put(event.request, networkResponse.clone());
27              if (cachedResponse) return cachedResponse;
28              // ServiceWorker fetch
29              return networkResponse;
30          } catch (err) {
31              // Report a lack of connectivity to the user.
32          }
33      }());
34  });
```

Listing 12: Fetch, Async/Await-Codestruktur

Während des 'fetch' Events werden die Dateien im Cache überprüft und gegebenenfalls erneuert, falls keine Netzwerkverbindung besteht, wird eine Version aus dem Cache angezeigt. Falls diese ebenfalls nicht vorhanden ist, dann kann leider nichts angezeigt werden.
Die Bildausschnitte mit dem Code wurden teils von mir erstellt, teils stammen diese von dieser Internetseite**dev:codeSourcesw:cacheThenNetwork**.

"Add-To-Homescreen"-Funktion

Für die Implementierung der "Add-to-Homescreen"-Funktionalität sind für Android, iOS und die diversen Browser unterschiedliche Schritte nötig.
Bei Android genügt es eine manifest.json Datei im root-Verzeichnis der App abzulegen. Diese Manifest-Datei enthält Meta-Informationen von App-Icon, über App-Name, über die Start-URL bis hin zu der Theme-Farbe für den Hintergrund des App-Icons oder dem Navigationsmenü im Browser. Wenn ein Nutzer im Abstand von 5 Minuten (Standard-Einstellung) die Webseite zweimalig aufruft, so wird dieser gefragt, ob er die App zum Homescreen hinzufügen möchte. Ab da kann er diese über das App-Icon als auch über den Browser starten.
Für iOS müssen die Meta-Informationen im Bereich des Head-Elements, der index.html eingefügt werden. So wird dort das Aussehen von mehreren App-Icon Versionen definiert. Dies wird benötigt, da die meisten Apple-Geräte unterschiedliche Density (Pixel per Inch) haben. Des Weiteren werden Konfigurationseinstellungen aktiviert oder deaktiviert, als auch das Aussehen des Ladescreens für iOS Geräte festgelegt.

Auch die Windows-Mobilgeräte muss in der index.html Datei im Head-Bereich mittels Meta-Informationen mitgeteilt werden, wie der Ladebildschirm oder das Appicon aussehen soll (vgl.**dev:homescre**

Erstellung der API-Schnittstelle

Nachdem Server und die Clientseite der PWA funktionsfähig waren, war es noch relevant, dass zwischen diesen beiden kommuniziert werden kann. Dafür verwendet man die API.**api** API steht für ”Application Programming Interface” und diese Programmierschnittstelle kann wie in diesem Projekt zum Zugriff auf die Datenbank verwendet werden. Dabei handelt es sich in diesem Fall um einen RESTful Webservice.**rest**

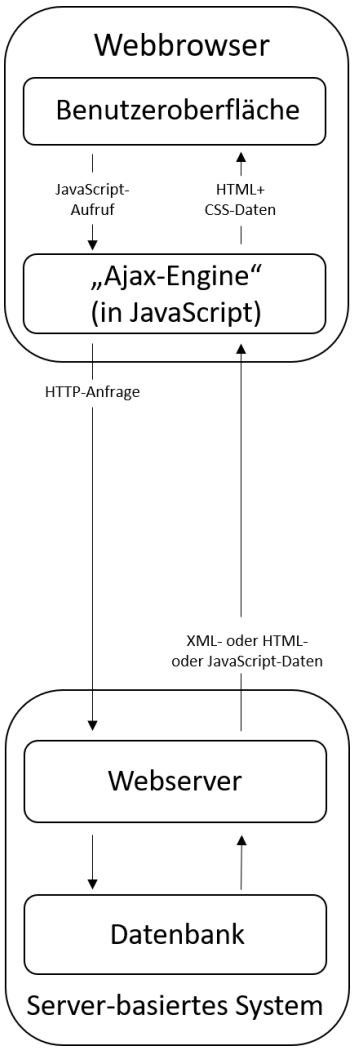


Abbildung 11: Ajax Modell einer Webanwendung. Quelle: nach Haischt, Daniel; https://de.wikipedia.org/wiki/Ajax_(Programmierung)#/media/File:Ajax-vergleich.svg

Das asynchrone Datenübertragungskonzept AJAX wurde zur Kommunikation zwischen Browser und Server verwendet.**ajax1 ajax2 ajax3** Dabei werden wie in Abbildung 11 zu sehen ist HTTP Requests von dem Browser an den Server geschickt. Diese werden daraufhin von dem Server verarbeitet und es werden entweder Daten zurückgegeben oder ein HTTP Response mit einer Rückmeldung. Mit der Verwendung von AJAX hat man einige Vorteile, zum Beispiel, dass man die Seite nicht neu laden muss, damit die neuen Inhalte erscheinen und dass der Browser bereits weiter Requests schicken kann, ohne auf die Responses des Servers warten zu müssen. Da die gesamte PWA des Projekts in JavaScript geschrieben ist, ist es zudem vorteilhaft, dass die AJAX-Engine in JavaScript geschrieben wird. Die Daten aus der Datenbank werden im JSON-Format transportiert.
Um die Funktionsweise einer derartigen Schnittstelle zu verstehen, wurde der Code einer PWA die Professor Matthias Hopf geschrieben und der Gruppe zur Verfügung gestellt hat, analysiert. Dieser war jedoch etwas zu komplex für einen Einsteiger in das Thema. Deswegen wurde auf simplere Beispiele in Tutorials zurückgegriffen. Diese Beispiele konnten nun auf das eigene Projekt übertragen und getestet werden. Nach einer ausführlichen Erläuterung durch Professor Hopf anhand eines Beispiels im Projektzusammenhang, war ein grundsätzliches Verständnis über die Schnittstelle vorhanden. Nun konnte die konkrete Umsetzung der API beginnen.
Die erste Methode die umgesetzt wurde war ”list_messages”. Diese zeigt nun alle Nachrichten der Datenbank auf der Startseite der PWA an. Dazu musste zuerst vom Server die IDs aller Nachrichten abgefragt werden. Man schreibt eine API Funktion, die ein GET Request an den Server schickt. Dieser sucht nach den IDs per find Anfrage in der Datenbank und antwortet mit einem HTTP Response, der das Array mit den IDs enthält. Auf der Client Seite wird nun für jede ID per GET Request alle Informationen der Nachricht angefragt. Der Server sucht die gefragten Information und schickt diese zurück mit dem dazugehörigen Nachrichtentext an den Browser.

```
1 list_messages: function () {
2     $.ajax({url: "api/ids", method: "GET"})
3      .done(jd => {
4         _messagelist.splice(0, _messagelist.length);
5         _messagelist.push.apply(_messagelist, jd);
6         console.log("jd: "+jd);
```

```
7         for (var e in jd) {
8             if (!_messages[jd[e]]) {
9                 get_insert_message(jd[e]);
10            }
11        }
12    }).fail(function (e, f, g) {
13        console.log("err: " + e + f + g);
14    });
15}
```

Listing 13: ”list_messages”-Methode

Was für Vorgänge beim Senden einer Nachricht in einer API getätigt werden, ist im Folgenden aufgeführt. Hierfür muss die Mongodb heruntergeladen und angebunden sein.
In der Server.js Datei wird durch die ”App.post”-Methode eine Nachricht erstellt. Dabei wird in ”createMsg” die Funktion aufgerufen, in einen String umgewandelt und die Nachricht anschließend in der Datenbank gespeichert. Mit der POST-Methode können Nachrichten in die Datenbank gespeichert werden und anhand von einer GET-Methode angezeigt werden. Nun müssen in der createMsg.js alle Werte einsetzbar gestaltet werden. Wichtig dabei ist, sich an das vorgegebene Mongo Schema zu halten. Durch die ajax-Methode ”post” können große Datenmengen an den Server geschickt werden. Zuletzt soll die Konsole auch einen Error anzeigen, falls der Abruf der Funktion erfolglos war.
In Abbildung 12 kann man diese Vorgänge nachverfolgen. Diese beiden Funktionen haben am Ende funktioniert und können zusammen verwendet werden.

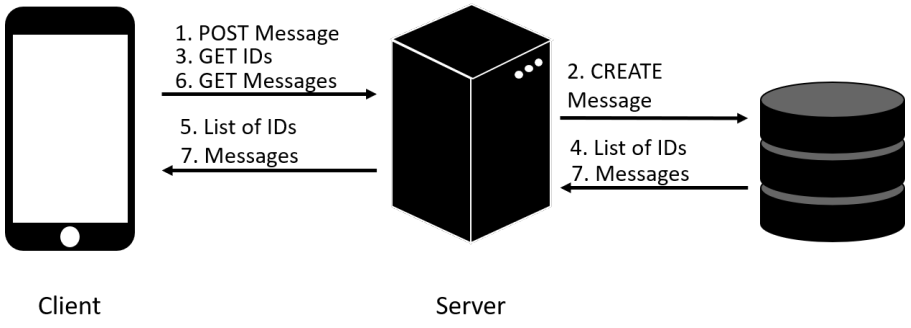


Abbildung 12: Erstellen einer Nachricht und Anzeigen aller Nachrichten

Aufsetzung des Servers

Zunächst muss ein http port eingerichtet werden. Über einen lokalen Server über node.js ist die Ohm News-App abrufbar. Mit der createServer Methode

```
1 http.createServer(app).listen(http_port, function () {
2     console.log("Express http server listening on port " + http_port);
3 });
```

Listing 14: ”list_messages”-Methode

lässt sich ein Server aufsetzen. Hier wird auch der Listener für den Port impliziert und auf der Konsole ausgegeben, wenn die Verbindung erfolgreich war.
Um die wichtigsten Fehlermeldungen und Bugs auszugeben, wird für unsere node.js App der Logger ”Morgan” verwendet. Zusätzlich wird eine try-catch Funktion eingeordnet, da Zertifikat und Schlüssel noch nicht zertifiziert sind (siehe SSL und Keys) und somit vorerst ein Fehler angezeigt wird.

SSL und Keys

Um die Applikation für die Nutzer sicher zu gestalten und eine Verschlüsselung sensibler Daten zu gewährleisten, sorgt ein SSL(Secure Socket Layer). Dadurch wird ein sicherer Kanal zwischen Web-Server und Client bereitgestellt und aus der http- wird eine https-Verbindung. Übergangsweise werden ein selbst verifizierter Schlüssel und Zertifikat erstellt, welche später durch echte ausgetauscht werden. In einem neu erstellten Ordner ”keys” werden das Zertifikat und der Schlüssel platziert.

Datenbankanbindung

Für diese Anwendung wird eine Datenbank benötigt, weil die geschriebenen Nachrichten gespeichert werden müssen. Später sollen auch User mit ihren Userinformationen gespeichert werden. Im Falle dieses Projekts wurde sich für eine MongoDB entschieden.
MongoDB ist eine dokumentenorientierte NoSQL Datenbank. Diese ist für diese Anwendung sinnvoll, da sie im Vergleich zu SQL Datenbanken flexibler handhabbar ist und somit auch viele

Einstellungen noch im Nachhinein geändert werden können.**mongo**
Um den Umgang mit MongoDB zu vereinfachen wurde Mongoose, eine Object Data Modeling Bibliothek, verwendet.**mongoose**
Da man zu Beginn des Projekts noch nicht mit NoSQL Datenbanken gearbeitet hatte, mussten zuerst Tutorials und Erklärungen gelesen werden. Anhand von Beispieltests mit einer lokalen Datenbank konnte die grundsätzliche Funktionsweise bald verstanden werden. Daraufhin wurde das grobe Schema für eine Nachricht aufgebaut. Eine Nachricht sollte einen Betreff, eine Nachricht, den schreibenden User und die Tags beinhalten wie man in Listing 15 an einer beispielhaften JSON-Datei sehen kann. Die Datenbank musste konfiguriert und gestartet werden.

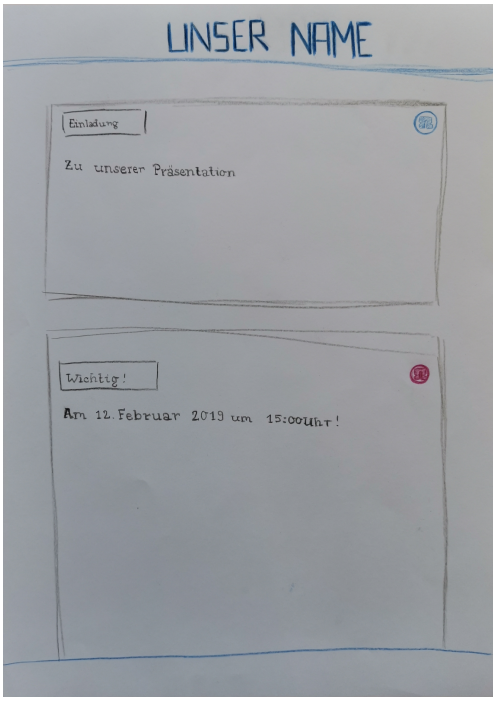
```
1 {
2     "messages": [{
3         "subject": "Test"
4         "message": "Hallo Welt!"
5         "tag": "#OHM_News"
6         "user": "User1"
7     }]
8 }
```

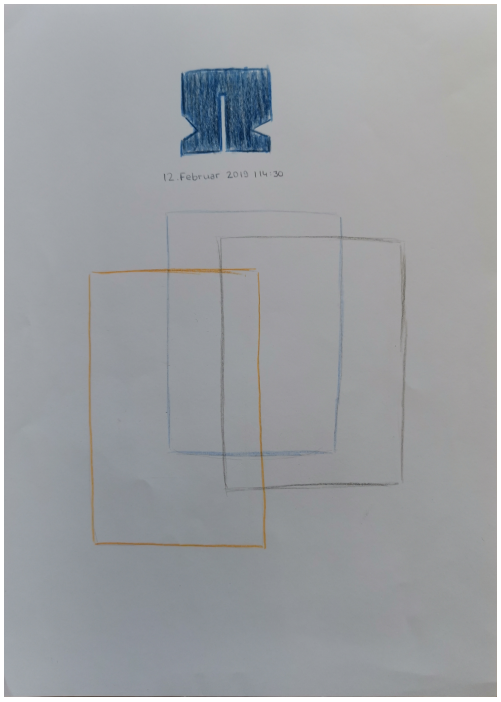
Listing 15: Nachricht im JSON-Format

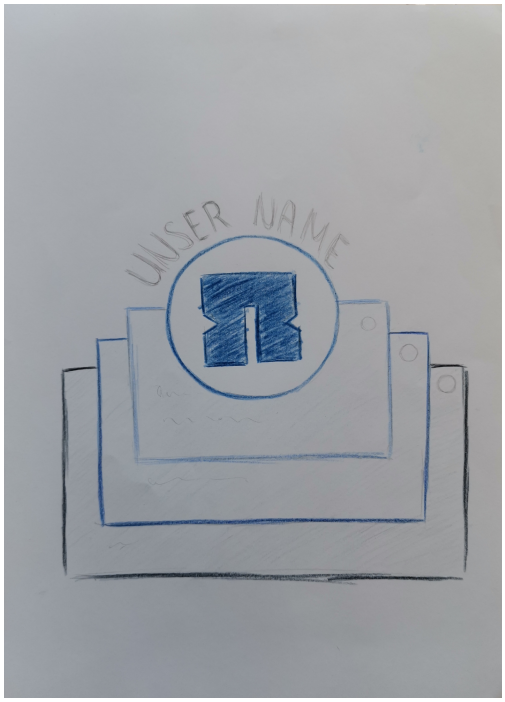
Die serverseitigen Codeteile zur API und der Code für die Datenbank sind momentan zur Vereinfachung in der server.js Datei. Später sollen diese zur Übersichtbarkeit und Wartbarkeit ausgelagert werden. Das Schema für die Nachricht und die Konfiguration der Datenbank sind zu diesem Zweck bereits in eigenen js-Dateien zu finden.
Sobald all das lokal funktioniert hat, wurde die Portierung auf einen Server der Hochschule vorgenommen.

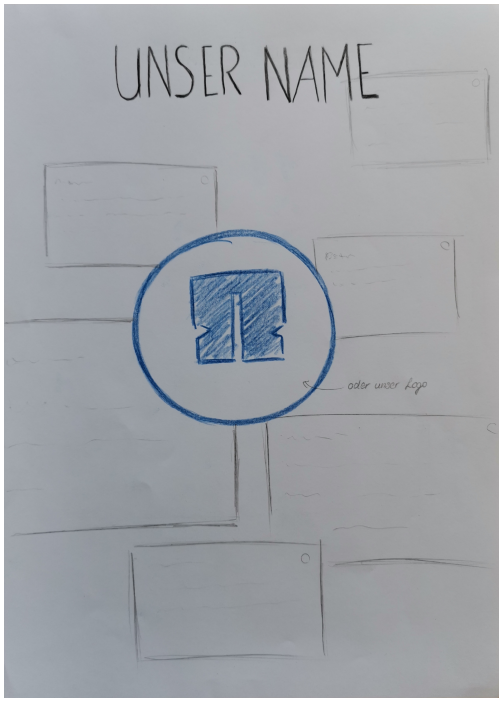
Plakatgestaltung

Zu dem Plakat für die OHM News App wurde eine Ideensammlung in Form von Skizzen angefertigt.



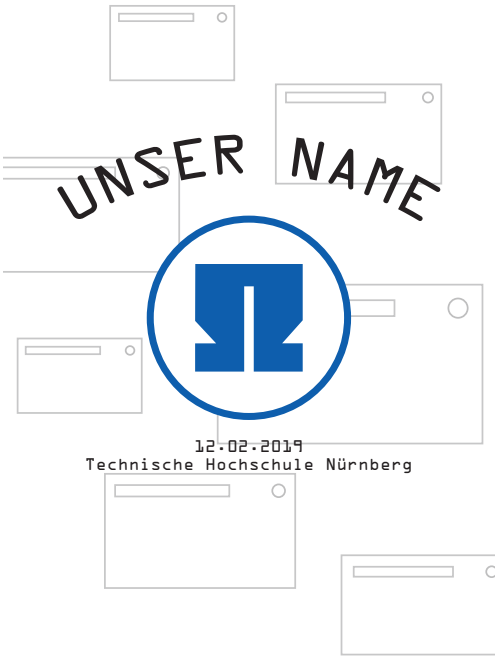






Die Umsetzung derer folgte in Form von ersten Entwürfen in Illustrator, welche der gesamten Gruppe vorgestellt wurden.

Zu den hier unten aufgeführten Plakatentwürfen wurden jeweils zwei Versionen erstellt. Diese zwei kamen jedoch in die engere Auswahl und sollten noch weiter ausgearbeitet werden.

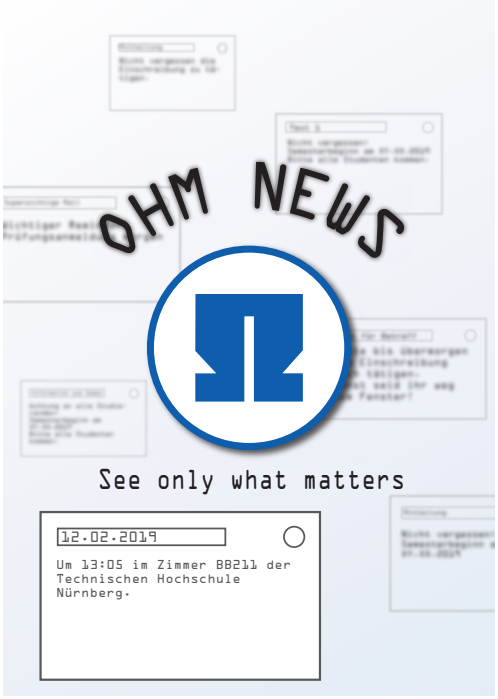


Bei der nächsten Vorauswahl an fünf OHM News Plakat-Versionen wurde daraufhin auf zwei reduziert, welche hier zu sehen sind.

Diese wurden weiter bearbeitet und Änderungen in Absprache mit der Projektgruppe vorgenommen. Auch die Meinung von Professor Dr. Hopf beeinflusste die Plakatgestaltung.

Einige Ansätze später und nach etlichem Experimentieren stand die Endauswahl bei diesen nächsten zwei Exemplaren. Besonderen Zeitaufwand nahm die Suche nach einer geeigneten

Schrift für die zweite Version in Anspruch. Es gestaltete sich als äußerst schwierig, den künstlerischen leicht verspielten Ausdruck des Plakatlayouts mit der Seriösität des Hochschulcharakters in einer Schrift zu vereinen.



Zuletzt fiel die Wahl auf das linke mit blauem Verlauf versehene Plakat. Denn es vermittelt genau das, was der Slogan des Projektes aussagt. Der Fokus liegt auf der Nachrichtenbox, also das was wichtig ist, und der Rest verschwimmt im Hintergrund.
Bei den wöchentlichen Treffen wurde im Allgemeinen der Fortschritt des Plakatdesigns analysiert und mit konstruktiver Kritik versehen. Dieses Feedback wurde im Rahmen des Möglichen umgesetzt und so kristallisierte sich die finale Version heraus. Nach gründlichem Finetuning und einer Umformatierung von DIN A3 auf DIN A1 wurde die Endfassung per Mail eingeschickt und schließlich ausgedruckt. Im Anhang ist sie in voller Auflösung aufzufinden.

PowerPoint Präsentation

Als sich das Projekt für dieses Semester zum Ende hinneigte, war es an der Zeit sich für die Projektpräsentation vorzubereiten. Hierfür erstellte ich für die Gruppe eine PowerPoint Präsentation. Zunächst besprachen wir Gemeinschaftlich, wie diese ungefähr aussehen sollte. Wir waren alle einstimmig der Meinung, dass wir die Präsentation sehr einfach gestalten wollen, ohne zu lange Textpassagen mit einzubinden. Wir wollten es nämlich verhindern, dass das Publikum von zu

vielen Faktoren in der Darstellung abgelenkt werden, zum Beispiel durch lesen von langem Fließtext. Durch den Gebrauch von bildlicher Veranschaulichung, versuchten wir es den Zuschauern zu ermöglichen, das Verständnis zu erleichtern. Auch hatten wir die Idee, eine fiktive Persona zu erstellen, zu dem der Betrachter einen Bezug finden kann und diese durch das vorgetragene Thema begleitet.

Gestaltungselemente

Da man jedoch, aus Urheberrechtlichen Gründen, nicht einfach wahllos Charaktere aus dem Internet benutzen kann, suchte ich nach einer Seite, auf der man solche Animationen erstellen kann. Ich bin auch direkt fündig geworden, trotz alle dem war es erstmal schwer etwas zu finden das man auch Gratis verwenden konnte. Dann bin ich auf die Seite www.Powtoon.com gestoßen. Dort fing ich als nächstes an, unseren sogenannten Klaus in all den Stimmungen, die wir für unsere Präsentation brauchten, zu gestalten.
Wir entschieden im Vorfeld als Gruppe, dass sich unser Männchen so wenig wie möglich bewegen soll, aus dem gleichen Grund wie auch oben schon erwähnt und zwar um den Fokus nicht zu sehr darauf zu richten und das Publikum nicht zu sehr abzulenken. Nachdem das erledigt war, war es an der Zeit, die eben erstellten Videos von unserer Persona einzeln so zu kürzen und zu beschneiden, dass man sie gut auf die gewünschte Position einsetzen konnte. Als dieses erledigt wurde, besprach ich weitere Vorgehen einzeln mit den anderen Gruppenmitgliedern. Ich Informierte mich darüber, über welches Thema jeder einzelne vorträgt und wie sie sich ihren Präsentationsteil vorstellen. Diese Wünsche und Vorstellung setzte ich dann dementsprechend in unserer Präsentation um. Falls sie an dem Ergebnis interessiert sind, haben sie die Möglichkeit sich dies im Git-Repository im Ordner ”Präsentation” anzusehen.

Weiterverwendung für Infoscreens

Am Mittwoch, den 24.10.2018 fand das Info-Treffen mit Frau Fabi statt, welches von Prof.Dr. Matthias Hopf organisiert wurde. Dieses Treffen diente der Besprechung und Erörterung zukünftiger Nutzungsmöglichkeiten der Info-Screens, das sind geplante und teils bereits montierte Bildschirme zur Information-Bereitstellung für die Hochschulangehörigen auf dem Campusgelände. Frau Fabi ist Angehörige der Fakultät efi und in ihrer Funktion für die Dokumentenlenkung und Webredaktion an dem ”Tag”-basiertem Nachrichten-System unserer Webanwendung inter-

essiert, welches zukünftig auch auf den Bildschirmen dargestellt werden könnten. Sie ließ auch das Interesse von Herr Pöhlau, dem Dekan der Fakultät efi bekunden.
Außerdem informierte Frau Fabi uns, die Projektmitglieder und unseren Projektbetreuer, dass die Fakultät efi sicher die restlichen Info-Screens montieren werde und derzeit Besprechungen hinsichtlich weiterer Sicherheitsaspekte zu den Montagebereichen am Laufen sind. Ergänzend würden sie sich eine Softwarelösung wünschen, für die Ansteuerung und ”Befüllung” der Monitore mit den Daten unserer App. Diese wird jedoch nicht mehr im Rahmen unseres Projektes umgesetzt werden. Die Umsetzung einer Softwarelösung wäre als Idee für ein zukünftiges Projekt denkbar, nach Aussage von Prof.Dr. Matthias Hopf.

Soll/Ist Vergleich auf Basis des Projektplanes

Zu Beginn des Semesters hat die Gruppe zusammen ein Pflichtenheft für den ersten Abschnitt des Projekts erstellt. Ziel war es den Datenzugriff zu ermöglichen und eine Schnittstelle zur Verfügung zu stellen. Es besteht eine Datenbankanbindung über den Server der Technischen Hochschule Nürnberg. Darüber hinaus sollten die folgenden Ziele im Bereich der grafischen Benutzerschnittstelle erreicht werden.
Die Oberfläche der PWA sollte über eine Login Möglichkeit, eine Menüleiste, eine Suchoption, ein Profil, eine Filterfunktion, ein Dashboard als Hauptseite, ein Einstellungsmenü, Nachrichtenfelder und ein Formular zum Verfassen einer Nachricht verfügen. Die Login Funktion ist noch nicht implementiert und die Filterfunktion soll im nächsten Semester in das Suchfeld und die Profilanzeige eingebaut werden. Die restlichen Funktionen sind in der PWA grafisch umgesetzt worden. Es sollte zusätzlich ein Abhängigkeitsdiagramm erstellt werden. Allerdings war dies aufgrund fehlender komplexer Zusammenhänge noch nicht notwendig. Für die API-Schnittstelle wurde eine Dokumentation erstellt, um den Überblick zu bewahren. Im Bereich der Anwendungslogik wurde das Routing, die Darstellung der Nachrichten, die Speicherung der Nachrichten in der Datenbank und der LDAP-Login besonders behandelt. Das Routing sowie die Erstellung, Speicherung und Darstellung der Nachrichten funktioniert. Die Login Funktion wurde noch nicht näher betrachtet. Codetests wurden bisher noch keine durchgeführt. Das Rollenkonzept der Nutzer wurde bereits besprochen, allerdings noch nicht umgesetzt.
Darüber hinaus gibt noch einige nachrangige Punkte. Dazu gehört die Festlegung des Anwendungsnamens, die Erstellung eines Logos, das UX-Konzept, das Filtern der Datenbankabrufe,

das Festlegen der Anzeigekriterien und die Funktion einem Kanal zu folgen. Wie oben bereits erwähnt wird das Filtern der Abrufe im nächsten Semester umgesetzt. Auch die Anzeigekriterien und die Möglichkeit einem Kanal zu folgen wird erst zu einem späteren Zeitpunkt möglich sein.

Fazit

Im ersten Projektabschnitt ist viel Zeit ist in der Konzeptionsphase vergangen, da es ein sehr iterativer Prozess ist und deshalb mehrere Durchläufe bis zum Endentwurf nötig waren. Verschiedene Meinungen mussten mit einbezogen werden. Doch ein gutes Konzept erspart am Ende viel Änderungsarbeit.
Für die weitere Erarbeitung der App haben sich die Projektmitglieder auf die Bereiche Frontend, Datenbank, Sever und Service-Worker aufgeteilt. Diese Aufgabenverteilung hat gut funktioniert. Trotzdem ist unser Team sehr oft ins Stocken geraten, da sich die Aufgabenbereiche gerade am Anfang oft überschnitten haben. Hinzukommt, dass wir insgesamt wenig Erfahrung in diesen Bereichen hatten. Wir haben uns deshalb zu Beginn intensiv in die Themen eingearbeitet, was länger dauerte als erwartet. Zusätzlich mussten wir auf fachliche Unterstützung zurückgreifen.

Alles in einem kann man trotzdem von einem erfolgreichen Projektabschnitt sprechen. Ein erster funktionierender Prototyp ist fertig und kann im nächsten Zug erweitert werden. Parallele individuelle Arbeiten sind einfacher möglich, da das Grundgerüst steht. Das Team hat sich in die Materie eingearbeitet und kann im zweiten Projektabschnitt schneller durchstarten. Die Idee und das Konzept für die App sind soweit ausgereift, sodass diese nur noch umgesetzt werden müssen. Abläufe innerhalb der Gruppe sind klarer geworden, allerdings gibt es noch einiges strukturelle Abläufe, die im zweiten Projektabschnitt weiter optimiert werden können.

Quellen- und Literaturverzeichnis