Hochschule Offenburg
offenburg.university

SYSTECS
software engineering

Bachelor Thesis
an der Hochschule Offenburg

Konzeption und Realisierung einer mobilen App zur Arbeitszeit-verwaltung auf Grundlage des PWA-Ansatzes und ASP.NET Core

Wintersemester 20/21
Bearbeitungszeitraum:
01.10.2020 – 31.01.2021

Vorgelegt von:
Olivia Preis

Fakultät:
Medien und Informationswesen

Studiengang:
Medien und Informationswesen

Betreut durch:
SYSTECS Informationssysteme GmbH

Erstbetreuer:
Prof. Dr. rer. nat. Tom Rüdebusch

Zweitbetreuer:
Frederik Pakai

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Literaturverzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Textstellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche gekennzeichnet.

Der Durchführung einer elektronischen Plagiatsprüfung stimme ich hiermit zu. Die eingereichte digitale Fassung der Arbeit entspricht der eingereichten schriftlichen Fassung exakt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben kann.

_____ _____

Datum, Ort Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

App	Application (zu Deutsch Anwendung)
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifikator
JSON	JavaScript Object Notation
LESS	Leaner Style Sheets
MVC	Model-View-Contoller
PWA	Progressive Web App
SASS	Syntactically Awesome Style Sheet
SCSS	Sassy Cascading Style Sheets
SSL	Secure Sockets Layer
UI	User Interface
URL	Uniform Resource Locator
XAML	Extensible Application Markup Language

1. Einleitung

In unserer modernen Arbeitswelt wächst der Wunsch auf Mobilität stetig. Längst haben die meisten Menschen ein Smartphone in der Hosentasche und die Entwicklung mobiler Anwendungen hat zunehmend an Bedeutung gewonnen. Die Möglichkeiten für Entwickler sind inzwischen sehr vielfältig. Ob native App, Hybrid App oder Progressive Web App, es gibt für jede Art eine Menge verschiedener Hilfestellungen zur Entwicklung dieser, durch verschiedene Frameworks.

Das Unternehmen SYSTECS Informationssysteme GmbH möchte seinen Mitarbeitern mehr Mobilität ermöglichen. Dies wollen sie erreichen, indem sie die Möglichkeit bieten, schnell von unterwegs seine Arbeitsstunden und Abwesenheiten einzutragen. Für die Erstellung dieser Anwendung wird eine Progressive Web App auf Basis des Frameworks ASP.NET Core gewählt. Wie es zu dieser Entscheidung kommt und wie diese sich entwickeln lässt, ist Teil dieser Arbeit.

Die Arbeit wurde in elf Kapitel unterteilt. Im nachfolgenden Kapitel wird es zunächst um die genaue Aufgabe dieser Arbeit bzw. der entstehenden PWA gehen.

Im dritten Kapitel werden bereits existierende Apps zur Arbeitszeitverwaltung analysiert. Dies soll zur Einführung in das Thema dienen und hilft bei der Erfassung der Anforderungen für die App dieser Arbeit.

Das vierte Kapitel erläutert den Ist-Zustand bei dem Unternehmen SYSTECS. Es offenbart die vorhandenen Systeme, die für die Entwicklung der PWA zur Verfügung stehen und hilft ebenfalls festzustellen, was die mobile App leisten soll. Außerdem wird hier auch kurz auf das Arbeitsumfeld im Unternehmen Bezug genommen.

Das fünfte Kapitel behandelt die Anforderungen an die PWA, die nach der erfolgreichen Analyse der existierenden Apps und dem Ist-Zustand zusammen mit dem Unternehmen formuliert werden konnten. Dabei wird genauer auf die Zielgruppe sowie die funktionellen und nicht funktionellen Anforderungen eingegangen.

Nachdem die Anforderungen für die App festgelegt sind, wird gezielt auf die verschiedenen Entwicklungsansätze einer mobilen App eingegangen, damit die Entscheidung des PWA-Ansatzes nachvollzogen werden kann. Dafür werden native Apps, Hybrid Apps und PWAs einzeln aufgegriffen, um sie nachfolgend direkt zu vergleichen.

Im nachfolgenden siebten Kapitel wird das Konzept der PWA erläutert. Dabei wird vor allem auf das User Interface der PWA eingegangen.

Das achte Kapitel handelt von den verwendeten Technologien, um das vorher beschriebene Konzept umzusetzen. Hierbei hat vor allem ASP.NET Core hohes Gewicht, welches die Basis dieser Arbeit ist. Zusätzlich werden die weiteren Technologien und Bibliotheken vorgestellt, die ebenfalls zur Realisierung der Anwendung beitragen.

Im neunten Kapitel wird dann auf die Realisierung der App eingegangen, bei dem auch einige Codeausschnitte zur Veranschaulichung eingebunden werden.

Das zehnte Kapitel stellt die entstandene mobile App, durch Einbinden von Screenshots, vor und bewertet sie.

Im elften und letzten Kapitel wird die Arbeit kurz zusammengefasst und abschließend ein Fazit gezogen.

2. Aufgabenbeschreibung

Die Firma SYSTECS hat ein neues eigenes System zur Projektverwaltung in Entwicklung. Für einen Teil dieses neuen Systems wird eine mobile App für Android- und iOS-Smartphones entwickelt, welche Teil dieser Arbeit ist.

Das Unternehmen geht davon aus, dass die meisten Anwendungsfälle des Systems das Eintragen von Stunden und Abwesenheiten ist. Dies stellt somit das Hauptaugenmerk dieser Arbeit da. In der mobilen App soll also zum einen das Projekttracking möglich sein, bei dem es darum geht seine gearbeiteten Stunden auf die einzelnen Projekte einzutragen. Zum anderen soll das Eintragen von Abwesenheiten sowie das genehmigen/ablehnen von Urlaub möglich sein.

Zu Beginn der Bearbeitung muss zunächst evaluiert werden, was für eine Art App am besten passt. Hierfür müssen die Entwicklungsansätze native App, Hybrid App und Progressive Web App genauer beleuchtet werden. Zusammen mit der Firma SYSTECS wird dann die Art der App entschieden. Die App-Form bestimmt somit auch die Art der Implementierung.

Als nächstes muss für eine optimale Darstellung ein Konzept für mobile Endgeräte entwickelt werden und anschließend folgt die Implementierung. Der Entwicklungsfortschritt wird auf Android- und iOS-Geräten überprüft.

Der Fokus bei der Entwicklung der mobilen App wird auf dem Frontend liegen, da die Backend-Komponenten vom bereits vorhandenen System verwendet werden können.

Das Ziel und gleichzeitig die Motivation dieser Arbeit ist vor allem eine neue Technologie kennenzulernen, die ohnehin immer mehr genutzt wird und zum anderen soll es möglich werden mal eben schnell von unterwegs seine Stunden einzutragen und Urlaub zu beantragen oder zu genehmigen. Diese Ortsunabhängigkeit führt zu einer höheren Flexibilität der Mitarbeiter, was somit auch der größte Nutzen der mobilen App ist.

3. Analyse existierender Apps zur Arbeitszeitverwaltung

Zur Ermittlung der Anforderungen und der Erstellung des Konzeptes dieser Arbeit, ist es wichtig vorab bereits existierende mobile Apps mit ähnlichen Funktionen zu analysieren. Dies ist hilfreich, um Anforderungen für diese Arbeit zu ermitteln oder Aspekte zu entdecken, die bei der mobilen App dieser Arbeit in einer besseren Variante umgesetzt werden sollen.

Um einen möglichst hohen Nutzen aus der Analyse zu ziehen, werden Apps gewählt, die sich in ihren Funktionen unterscheiden. Des Weiteren wird beachtet, dass nur ein Android-Smartphone zum Testen zur Verfügung steht und die Apps sich kostenfrei herunterladen lassen. Aus diesen Gründen fiel die Wahl auf folgende Android-Anwendungen: *Gleeo Time Tracker, Stempeluhr II, Zeiterfassung (DynamicG) und WorkingHours.* [1]

Zu Beginn der Arbeit ist bereits klar, dass keiner der genannten Apps für das Unternehmen zur Verwendung in Frage kommt, da die mobile App eine Anbindung an die Daten des Unternehmens benötigt. Trotzdem kann diese Analyse hilfreich sein, um sich anzuschauen, wie die Darstellung und Bedienung in diesen Apps gelöst wird.

3.1 Gleeo Time Tracker

In dieser Anwendung können mehrere Projekte und zugeordnete Tasks hinzugefügt werden. Diese werden in Listenform beim Öffnen der App angezeigt. Durch ein Tippen auf den Play-Button, der jeweils links von einem Task liegt, kann das Aufnehmen der Zeit für diesen Task gestartet und wieder gestoppt werden. Ebenfalls kann man manuell einen Zeiteintrag hinzufügen. [2]

Unter dem Menüpunkt „Zeitachse“ erhält man eine Übersicht über die einzelnen Stundeneinträge und eine Zusammenfassung aller Stunden des Tages, der Woche und des Monats. Ebenfalls kann man unter „Report“ eine einfache Ansicht der Summe der Stunden einsehen, die für die einzelnen Tasks im angegebenen Zeitraum geleistet wurden.

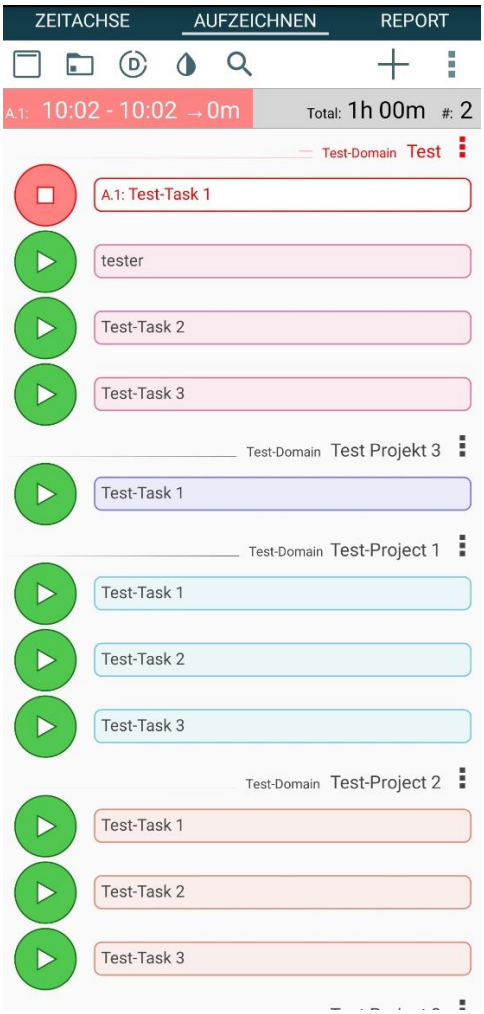

Abbildung 1: Screenshot Gleeo Time Tracker

Für bereits angelegte Tasks ist das Stoppen der Zeit sehr einfach und schnell. Die restliche Benutzerführung wirkt allerdings sehr kompliziert und es nimmt einige Zeit in Anspruch, bis man alle Funktionen verstanden hat und weiß, wo man was findet. Zusätzlich sind einige Links und Buttons sehr klein, was die Bedienung mit dem Finger erschwert.

3.2 Stempeluhr II

Diese App ermöglicht das Eintragen von Arbeitszeiten, wie auch Urlaub oder Krankheitstage und bietet eine einfache Übersicht der Einträge in Listenform auf der Startseite. Über diese Startseite lässt sich das Eintragen der Arbeitsstunden einfach erreichen und das Eintragen der Urlaubs- und Krankheitstage lässt sich über das Menü erreichen. [3]

Abbildung 2: Screenshot Stempeluhr II

Die Bedienung dieser Anwendung ist sehr einfach, allerdings besitzt sie nur einen begrenzten Funktionsumfang, somit lassen sich beispielsweise die Arbeitsstunden nicht kategorisieren.

3.3 Zeiterfassung (DynamicG)

Diese Anwendung ist ein recht mächtiges Tool, wenn man sich erstmal mit der Bedienführung angefreundet hat. Zu Beginn der Verwendung muss über die Einstellungen der App vieles eingerichtet werden, um die App bestmöglich verwenden zu können. So werden hier beispielsweise die unterschiedlichen Projekte angelegt, auf diese die Stunden eingetragen werden können. Außerdem kann man einen Stundensatz eintragen, Sollstunden angeben und sogar das UI individualisieren. [4]

Stundeneintragungen lassen sich über einen automatischen Zeitstempel über „Jetzt Einstempeln“ und anschließend „Jetzt Ausstempeln“ oder manuell über eine Uhrzeitwahl durch „Einstempeln“ und „Ausstempeln“ eintragen. Außer-

dem besteht die Möglichkeit über eine zuvor selbsterstellte oder bereits bestehende Vorlage einen Eintrag vorzunehmen.

Zusätzlich bietet diese App eine ausführliche Berichterstattung, in der z.B. die *Kunden pro Tag* oder die *Arbeitseinheiten* analysiert werden können. Diese Berichte lassen sich zudem personalisieren und in unterschiedlichen Dateiformaten versenden.

Abbildung 3: Screenshot Zeiterfassung (DynamicG)

Diese App bietet dem Nutzer umfangreiche Möglichkeiten, allerdings bedarf es einer gewissen Einarbeitung in die vielen verschiedenen Einstellungen.

3.4 WorkingHours

In dieser Anwendung lassen sich über den Startbildschirm schnell Stunden manuell oder durch einen Zeitstempel eintragen. Die Stunden lassen sich ein-

zelnen Aufgaben zuordnen, die man ebenfalls selbst anlegen kann. Eine Übersicht der eingetragenen Stunden erhält der Nutzer durch eine Listenansicht ebenfalls auf der Startseite. Unter dem Menüpunkt „Analyse“ befindet sich ein Diagramm, das die Einträge zusammenfasst. [5]

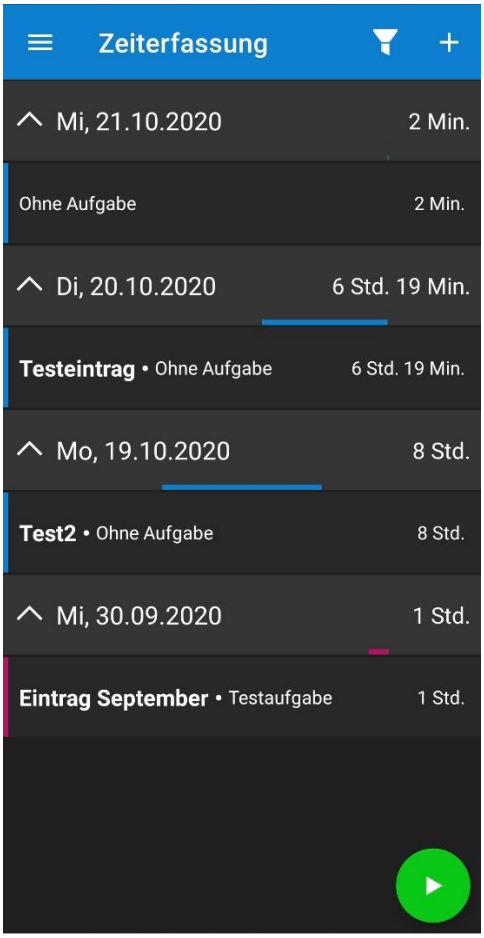

Abbildung 4: Screenshot WorkingHours

Diese Anwendung ist leicht verständlich und einfach zu bedienen. Allerdings besitzt sie auch nur einen begrenzten Funktionsumfang und so lassen sich beispielsweise keine Urlaubstage festhalten.

3.5 Tabellarischer Vergleich und Ergebnisse

Für den direkten Vergleich der Apps werden einige Vergleichsaspekte aufgestellt. Bezüglich der Eingabe wird untersucht ob für **mehrere Jobs/Projekte** Daten eingegeben werden können, **Feiertage** integriert sind und **Abwesenheiten** eingetragen werden können. Bei der Auswertung bzw. Übersicht der Stunden wird beachtet, welche **Art der Wiedergabe** verwendet wird. Zusätzlich wird die **Bedienführung** bewertet. [1] Diese Aspekte sind relevant für diese Arbeit und werden deshalb ausgewählt.

	Gleeo Time Tracker	Stempeluhr II	Zeiterfassung	WorkingHours
Mehrere Jobs/Projekte	✓	✓ (2 Jobs)	✓	✓
Feiertage	×	✓	×	×
Abwesen-heiten	×	✓	×	×
Art der Wiedergabe	grafisch und tabellarisch	tabellarisch	grafisch und tabellarisch	grafisch und tabellarisch
Bedien-führung	-	+	+-	++
✓ vorhanden × nicht vorhanden ++ sehr gut + gut +- passabel - schlecht				

Tabelle 1: Vergleich der bestehenden Apps für Arbeitszeitverwaltung

Zur Erstellung dieser Arbeit wurden aus dieser Analyse einige Erkenntnisse gewonnen. Zum einen wurde bei der Analyse festgestellt, dass die meisten Apps eine komplizierte Benutzerführung besitzen und oft nicht klar ist, wo man was findet. Somit braucht es häufig eine längere Einarbeitungszeit in diese Anwendungen, um sie vollumfänglich nutzen zu können. Bei der Erstellung des Konzeptes für diese Arbeit wird daher darauf geachtet, dass sich die App intuitiv bedienen lässt.

Des Weiteren wurde festgestellt, dass häufig die Bedienung mit dem Finger durch zu kleine Bedienelemente erschwert ist. Bei der Konzeption und der anschließenden Realisierung soll somit darauf geachtet werden, dass alle bedienbaren Elemente eine gute Größe für eine Bedienung mit dem Finger erhalten.

Zusätzlich ist festzuhalten, dass sich eine Liste oder Tabelle als einfache Übersicht über verschiedene Einträge eignet. Dies könnte auch nützlich für das Konzept der neuen mobilen App sein.

Bei funktioneller Betrachtung der Anwendungen ist aufgefallen, dass es jeweils immer eine Stundenübersicht gibt, über die man dann das Stundeneintragen erreicht. Das Eintragen der Stunden erfolgt dann jeweils über Uhrzeiteingaben. Lediglich bei *WorkingHours* gibt es die Möglichkeit, Arbeitsstunden direkt in

Stundenzahlen anzugeben. Zusätzlich lassen sich in allen betrachteten Anwendungen, außer bei *Stempeluhr II,* die Stunden auch über einen Zeitstempel eintragen. Dieser ermöglicht das Ermitteln der Stunden über das Starten und Stoppen der Arbeitszeit. Diese möglichen Arten des Stundeneintragens werden bei der Sammlung der Anforderungen für diese Arbeit berücksichtigt.

4. Ist-Zustand bei SYSTECS

Für ein firmeninternes System soll eine zusätzliche Anwendung für mobile Geräte entwickelt werden. Um feststellen zu können, was die App leisten muss und welche Teile des vorhandenen Systems verwendet werden können, bedarf es zunächst einer Analyse des Ist-Zustandes.

4.1 Arbeitsumfeld

SYSTECS Informationssysteme GmbH ist ein mittelständiges Unternehmen, das unterschiedliche Software in vier Abteilungen entwickelt. Dabei werden Projekte in den Branchen Automotive, Bauindustrie, Industrie & Produktion, Medizintechnik und Verwaltung umfassend begleitet. [6]

Die vorhandene Web-App wird von den Mitarbeitern des Unternehmens selbst entwickelt. Ein sehr großer Teil wird hier auch von Studenten übernommen. Insgesamt besteht das Projekt zusammen mit der zu entwickelten mobilen App aus drei Praktikanten, einem Werkstudenten und zwei Projektleitern.

Viermal in der Woche findet eine Besprechung mit allen Beteiligten des Projektes statt, in dem aktuelle Themen besprochen und Fragen geklärt werden können. Zusätzlich besteht die Möglichkeit, die Teammitglieder über *Microsoft Teams* oder per Mail zu erreichen.

4.2 Systembeschreibung

Die Webanwendung mit dem Arbeitstitel SYPRA (**SY**STECS **P**rocess and **R**esource **A**dministration) soll die Abläufe der täglichen Arbeit vereinfachen. Es geht um die Verwaltung von Mitarbeitern, Abwesenheiten, Aufträgen, Projektdaten, Kunden und Kosten. Die Verwaltung wurde bisher von einer in die Jahre gekommene Windows-Applikation übernommen, die SYPRA nun ersetzen soll.

Das Frontend dieser Anwendung, steht bereits zum Großteil. Die Datenbankstruktur ist bereits vollständig aufgebaut. Im Backend fehlen noch einige Bereiche, die noch implementiert werden müssen. Am System wird während der Erstellung dieser Arbeit weiterentwickelt.

Die einzelnen Bereiche und Funktionen der SYPRA Anwendung sollen durch nachfolgende Abbildung nachvollzogen werden können.

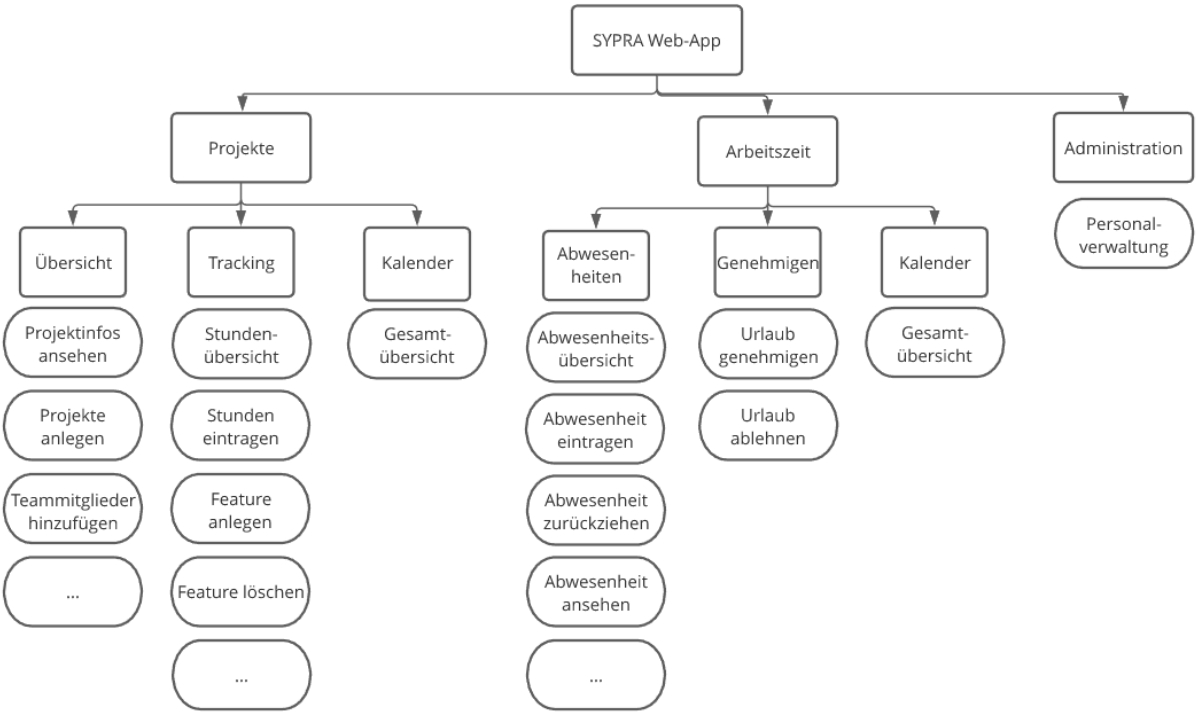

Abbildung 5: Funktionsüberblick bezüglich SYPRA

Der Schwerpunkt für die Erstellung dieser Arbeit sind das Projekttracking, die Erfassung von Abwesenheiten sowie das Genehmigen bzw. Ablehnen von Urlaub. Der Ist-Zustand dieser Teilbereiche ist für die Erstellung dieser Arbeit somit das Wichtigste, daher wird in diesem Kapitel ausschließlich auf diese Bereiche des vorhandenen Systems genauer eingegangen.

Gelangt der Nutzer zum Projekttracking erhält er zunächst eine Übersicht seiner aktiven Projekte und geleisteten Stunden je Tag in Tabellenform. Klickt der Nutzer auf ein Projekt erscheint darunter eine weitere Tabelle in dem die Arbeitsbereiche (auch Feature oder Items genannt) des gewählten Projektes angelegt sind bzw. angelegt werden können. Beim Anlegen eines Items muss ein Name und die zugehörige Feature ID angegeben werden. Die Items können ineinander geschachtelt sein, bearbeitet werden und sofern kein anderes Item untergeordnet ist und hierfür noch keine Stunden eingetragen wurden, kann es wieder gelöscht werden. Die einzutragenden Arbeitsstunden werden einem untergeordneten Item zugeordnet und können als Dezimalzahl für entsprechende Tage in die Tabelle eingetragen werden. Bereits eingetragene Stunden können ersetzt oder gelöscht werden. Außerdem ist das Springen in vorherige oder nachfolgende Daten sowie eine Auswahl des anzuzeigenden Bereichs möglich. Die Verwaltung der Projekte findet in anderen Bereichen der Anwendung statt und spielt für diese Arbeit keine Rolle.

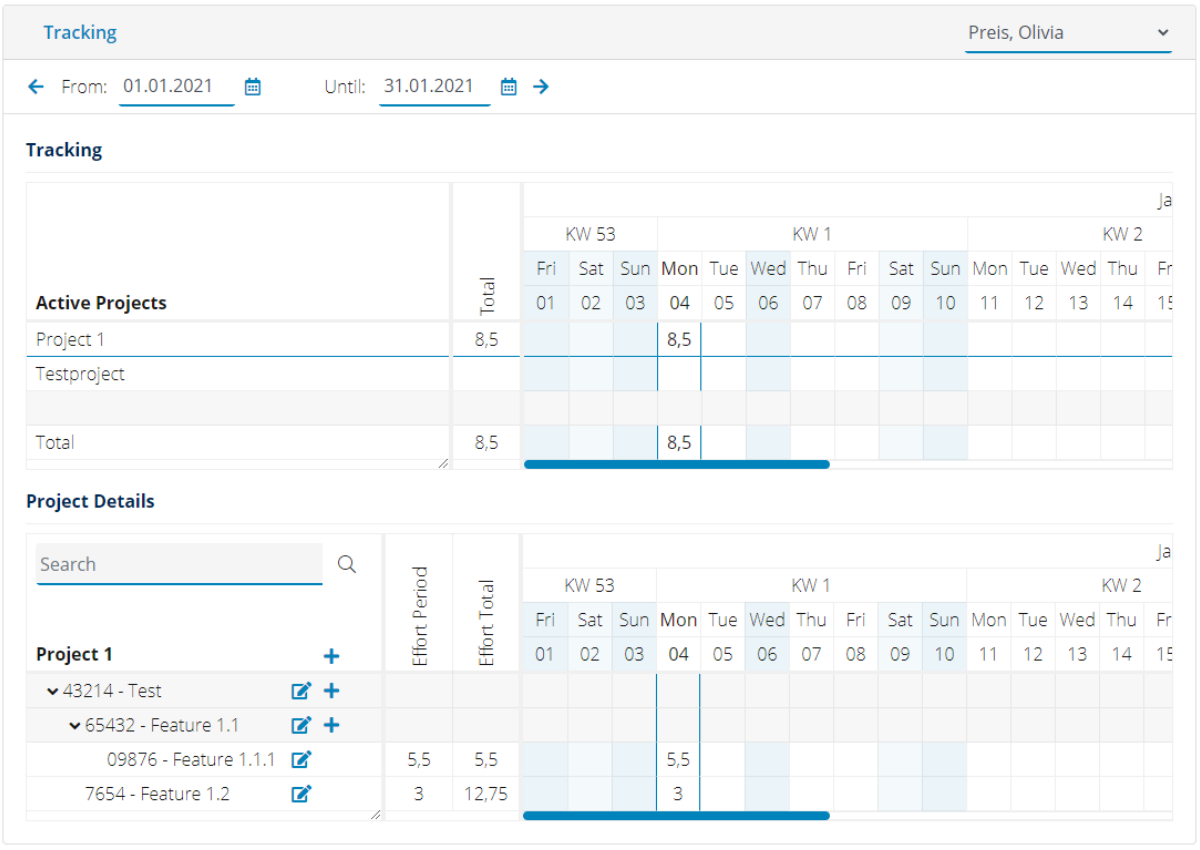

Abbildung 6: Projekttracking in SYPRA

Geht der Nutzer zum Bereich seiner Abwesenheiten erhält er eine Übersicht seiner Urlaubsanfragen und eine Kalenderansicht aller seiner Abwesenheiten, in der er zwischen den Monaten wechseln kann. Klickt der Nutzer auf eine der Eintragungen erscheint eine Detailansicht des Eintrags, in der dieser durch optionales Angeben eines Kommentars zurückgezogen werden kann. Es besteht außerdem die Möglichkeit eine neue Abwesenheit hinzuzufügen. Beim Hinzufügen wird zunächst eine Abwesenheitsart angegeben. Folgende Typen sind hier möglich: Urlaub, Sonderurlaub, Krankheit, Kind krank, Kurzarbeit, Elternzeit und Mutterschutz. Welche der Typen auswählbar sind, ist von den Rechten des Benutzers abhängig. Zusätzlich muss der Zeitraum der Abwesenheit und der Anteil der Abwesenheit am Start- und Endtag angegeben werden. Hier kann man zwischen halben und ganzen Tagen entscheiden. Optional lässt sich noch ein Kommentar angeben.

Benutzer mit bestimmten Rechten (Geschäftsführer, Bereichsleiter und Personal) können in der Anwendung Urlaubsanfragen genehmigen oder ablehnen. Hierfür steht dem Nutzer eine Übersicht aller aktuellen Anfragen zur Verfügung. Klickt er auf eine der Anfragen, werden die Details gezeigt und es kann angenommen oder abgelehnt werden. Beim Ablehnen kann optional noch ein Kommentar angegeben werden.

Diese vorgestellten Funktionalitäten haben hohen Einfluss auf die Anforderungen der mobilen App, denn der Nutzer soll einen möglichst hohen Wiedererkennungswert beim Nutzen der mobilen App haben.

4.3 Technologien

Das System ist rein webbasiert und besteht somit aus den Techniken HTML, CSS und JavaScript. Das Backend läuft hierbei in einer Cloudumgebung. Folglich ist die Anwendung von überall erreichbar.

Als Basis wird das ASP.NET Core Framework verwendet. Es vereinfacht die Entwicklung durch die Razor-Syntax und vieler schon vorhandener Bestandteile. Auf ASP.NET Core wird in einem späteren Kapitel nochmal genauer eingegangen.

Außerdem werden die Frontend-Frameworks Bootstrap, Font Awesome und FullCalendar eingesetzt, auf die ebenso später genauer eingegangen wird.

5. Anforderungen an die neue mobile App zur Arbeitszeitverwaltung

In diesem Kapitel geht es um die Anforderungen, die an die App gestellt werden.

Eine Anforderung beschreibt die Erwartungen an das System wie Bedingungen, Attribute, Ziele und den Nutzen. Es ist wichtig diese Anforderungen vor der Implementierung festzuhalten, da nur so sichergestellt werden kann, dass die Anwendung die Erwartungen erfüllt. [7]

Die Anforderungen sind durch die Analyse der bestehenden Apps sowie des Ist-Zustandes bei SYSTECS und aber auch in erster Linie durch Absprache mit dem Unternehmen entstanden.

5.1 Zielgruppe

Es wurde zuvor schon erwähnt, dass dieses Projekt firmenintern ist. Die Zielgruppe ist hiermit schon festgelegt und beinhaltet alle Mitarbeiter der Firma SYSTECS Informationssysteme GmbH, die ein Android- oder iOS-Smartphone besitzen. Hiermit ist auch davon auszugehen, dass die Zielgruppe technikaffin ist. Außerdem erlang die Entwicklung hierdurch einen hohen Qualitätsanspruch.

5.2 Funktionelle Anforderungen

Funktionelle Anforderungen beschreiben, was das System tun soll. Es handelt sich hier also um die Funktionen, Abläufe und Szenarien, welche beschreiben, wie die Anwendung auf bestimmte Eingaben reagieren soll. [7]

Die Anwendung wird zwei Hauptfunktionen beinhalten: das Projekttracking und die Abwesenheitsverwaltung. Zum Projekttracking gehört das Eintragen der Stunden auf die jeweiligen Projekte, das Anlegen von Aufgabenbereichen zu einem Projekt sowie eine Übersicht der eingetragenen Stunden. Die Abwesenheitsverwaltung inkludiert eine Jahresübersicht, das Eintragen sowie Zurückziehen einer Abwesenheit, das Genehmigen bzw. Ablehnen von Abwesenheitsanfragen sowie ebenfalls eine Übersicht. Der Fokus liegt allerdings zunächst auf dem Projekttracking, da dieser Teil dem Unternehmen am wichtigsten ist.

Die funktionellen Anforderungen werden, aufgrund des begrenzten Projektzeitraums, in Prioritätsstufen von 1-3 kategorisiert. Priorität 1 beschreibt hierbei die Funktionen, die das System auf jeden Fall aufweisen muss und die Funktionen mit Priorität 3 sind eher optional.

Stundenübersicht (Priorität 1)

Der Nutzer soll eine projektübergreifende Übersicht seiner eingetragenen Stunden erhalten. Dies soll durch eine Kalenderansicht für Monat, Woche und Tag möglich sein. In der Monatsansicht werden lediglich die eingetragenen projektübergreifenden Gesamtstunden je Tag angezeigt. In der Wochenansicht bekommt der Nutzer eine Übersicht über die Stunden die pro Projekt und Tag eingetragen wurden. Gelangt der Nutzer zur Tagesansicht wird jedes Detail eines Stundeneintrags angezeigt: Projekt, Item und Stundenzahl. Zwischen Monats- und Wochen-, Tagesansicht soll der Nutzer selbst umschalten können. Außerdem gelangt der Nutzer zur Tagesansicht, wenn er in der Monatsansicht auf einen Tag oder in der Wochenansicht auf einen Stundeneintrag tippt.

Stunden eintragen (Priorität 1)

Das Eintragen der Stunden wird über die Übersicht erreicht. Zunächst wird ein Item eines Projektes ausgewählt auf das die Stunden eingetragen werden sollen. Der Nutzer soll dann die Möglichkeit bekommen die Stunden für mehrere Tage auf einmal einzutragen. Bereits eingetragene Stunden sollen hierbei angezeigt werden.

Item zu Projekt hinzufügen und löschen (Priorität 1)

Der Nutzer soll über die Anwendung neue Items zu einem Projekt oder einem anderen Feature anlegen können. Beim Anlegen eines neuen Items muss ein Name und die zugehörige Feature ID angegeben werden. Hat ein Feature kein untergeordnetes Item und noch keine Stundeneintragungen, soll es auch gelöscht werden können.

Abwesenheitsübersicht (Priorität 2)

Die Anwendung soll eine Jahresübersicht der Abwesenheiten des Nutzers anzeigen können. Dies erfolgt mit Hilfe einer Liste aller Eintragungen. Ebenfalls wird über die Liste eine Detailansicht der einzelnen Abwesenheiten erreicht.

Abwesenheit eintragen (Priorität 2)

Das Eintragen einer Abwesenheit wird über die Jahresübersicht erreicht. Der Systemnutzer gibt einen Zeitraum, Art der Abwesenheit, Tagesanteil des Start- und Endtages sowie optional einen Kommentar an. Wenn es beim Absenden der Daten zu einer Übereinstimmung mit einer Abwesenheit gleichen Typs gibt, wird der Nutzer hierüber informiert und gebeten seinen Eintrag zu überprüfen.

Abwesenheit zurückziehen (Priorität 2)

Eine Abwesenheit soll in diesem System auch zurückgezogen werden können. Hierfür soll es einen Button bei den Details der einzelnen Abwesenheit geben. Beim Zurückziehen kann zusätzlich optional ein Kommentar eingegeben werden.

Abwesenheit genehmigen/ablehnen (Priorität 2)

Berechtigte Systemnutzer sollen über die App Abwesenheiten ablehnen oder genehmigen können. Diese Funktion wird nach dem Auswählen einer Abwesenheitsanfrage erreicht. Lehnt der Nutzer die Anfrage ab, kann er optional einen Kommentar angeben.

Zur Veranschaulichung einer Abwesenheit, soll es zusätzlich eine Kalenderansicht geben (Priorität 3).

Durch das Annehmen bzw. Ablehnen erhält der Mitarbeiter, der die Anfrage gestellt hat, eine entsprechende Mitteilung in Form einer Push-Benachrichtigung auf sein Smartphone (Priorität 3).

5.3 Nicht funktionelle Anforderungen

Nicht funktionelle Anforderungen (auch Qualitätsanforderungen genannt) ergänzen die funktionellen Anforderungen. Sie beschreiben qualitative Eigenschaften des Systems wie Verfügbarkeit, Sicherheit und Technologie [7]

Zuverlässigkeit und Verfügbarkeit

Die Anwendung soll zuverlässig funktionieren und immer verfügbar sein. Die Zuverlässigkeit und Verfügbarkeit des Systems soll bei 99% liegen.

Performance

Innerhalb fünf Sekunden soll die App geöffnet werden können und auch alle Benutzerinteraktionen sollen innerhalb dieser fünf Sekunden beantwortet werden.

Kompatibilität

Die Anwendung soll mindestens kompatibel mit Android Lollipop 5.0 (erreichen von 89,3 % der Android-Smartphones [8]) und iOS 12.2 (erreichen von 94% der iOS-Smartphones [9]) sein.

Benutzerführung und -oberfläche

Die Daten werden über eine mobil optimierte grafische Benutzeroberfläche ein- und ausgegeben. Hierbei erfolgt die Benutzerführung ausschließlich in Englisch. Zusätzlich ist zu beachten, dass die Anwendung zur Corporate Identity der Firma

SYSTECS passt und die Funktionen sich dem bereits bestehenden System ähneln, so dass der Wiedererkennungseffekt groß ist.

Implementierung

Die Entwicklungsumgebung ist Visual Studio Professional 2019 und als Basis wird *.NET* mit C# verwendet. Die Anwendung wird während der Entwicklungsphase und im Anschluss auf einem Android- und iOS-Gerät getestet. Obendrein wird der Code zur Verständlichkeit kommentiert.

Sicherheit und Authentifizierung

Die Authentifizierung der Nutzer soll über den Dienst *Active Directory* von *Microsoft* über die Domäne *systecs.com* erfolgen. Des Weiteren soll eine sichere Datenübertragung gewährleistet werden.

Verbreitung

Die App soll sich zum Startbildschirm hinzufügen lassen und im *standalone*-Modus öffnen. Zur Verbreitung lässt sich zusätzlich festhalten, dass das Unternehmen selbst einen möglichst geringen Aufwand bezüglich der Verbreitung wünscht.

Wartbarkeit

Die Anwendung soll einfach gewartet und erweitert werden können, damit dem Unternehmen ein möglichst geringer Aufwand entsteht.

6. Entwicklungsansätze mobiler Apps

In diesem Kapitel werden verschiedene Arten von mobilen Apps vorgestellt, so dass am Ende des Kapitels die Entscheidung des Entwicklungsansatzes für dieses Projekt nachvollzogen werden kann.

6.1 Native App

Eine native App wird speziell für ein Betriebssystem entwickelt und angepasst. Die Technologien sind dementsprechend, je nach Betriebssystem, unterschiedlich. Meist wird in den Programmiersprachen Java, C++ oder Swift entwickelt. Will man eine App für alle Betriebssysteme entwickeln, ist also aufgrund der plattformeigenen Entwicklung der Realisierungsaufwand ziemlich groß. [10]

Ein entscheidender Vorteil von nativen Apps ist, dass sie jegliche Hardware und Funktionen des Handys einbinden können. Hierzu gehört beispielsweise der Zugriff auf den Kalender, die Kamera oder das Verwenden des Fingerabdrucksensors.

Native Apps werden über einen entsprechenden App-Store (*Google Play* für Android und *App Store* für iOS) heruntergeladen, woraufhin alle Dateien der App direkt auf dem Smartphone gespeichert werden.

Durch bestimmte Frameworks ist eine plattformübergreifende Entwicklung möglich. Ein Beispiel hierfür ist *Xamarin.* Hierbei handelt es sich um ein Framework der .NET-Familie von Microsoft. Bei einer Entwicklung mit diesem Framework wird, .NET typisch, in C# programmiert und das UI wird mit XAML definiert. Aus dem entstandenen Quellcode wird dann jeweils für Android und iOS eine eigene native App kompiliert. Die Verwendung so eines Frameworks führt allerdings häufig zu verminderter Performance der Anwendung. [11]

Das Beispiel einer Hotel-Such-Plattform als native Android-App zeigt folgende Abbildung:

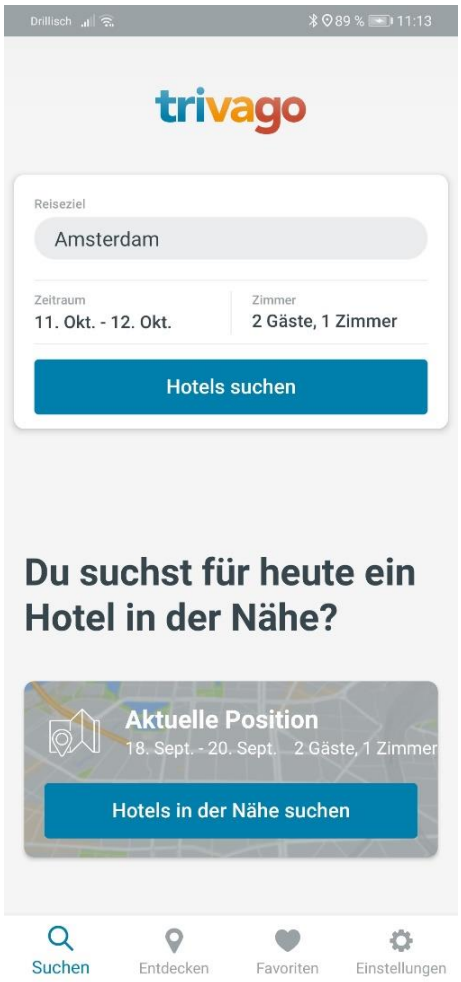

Abbildung 7: Screenshot native App Trivago

Dass es sich bei der Trivago-App [12] um eine native App handelt, kann man an den verwendeten Technologien erkennen, die Trivago in den App-Einstellungen unter „Lizenzen Dritter“ bekannt geben. Hier ist zu erkennen, dass diese App in der Programmiersprache *Java* mit Hilfe der Entwicklungsumgebung *Visual Studio App Center SDK for Android* entwickelt wurde.

6.2 Progressive Web App

Eine PWA ist nichts anderes als eine modifizierte Webseite und stellt somit die Weiterentwicklung einer klassischen Web-App da. Sie passt sich dem Ausgabegerät bzw. dem verwendeten Browser an, ist vollständig für mobile Geräte optimiert und sieht nach dem Hinzufügen auf den Startbildschirm des Smartphones aus wie eine native App. Die Technologien sind entsprechend webbasiert: HTML, CSS und JavaScript. Demzufolge sind PWAs vollkommen plattformunabhängig. [13]

Anders als bei nativen Apps lassen sich bei einer PWA nicht alle Funktionen und Hardware des Smartphones integrieren. Bluetooth, Fingerabdrucksensor, NFC

sowie Zugriff auf Kontakte und Kalender können nicht eingebaut werden. Bei iOS-Geräten kommt hinzu, dass keine Push-Benachrichtigungen und keine Hintergrundsynchronisierung unterstützt wird. Allerdings passen sich die Funktionen der PWA den Rahmenbedingungen an. Wenn also beispielsweise eine PWA entwickelt wird, die Push-Benachrichtigungen enthält, funktioniert diese App auch auf iOS-Geräten und lediglich die Push-Benachrichtigungen finden hier keine Anwendung. Außerdem muss ein Gerät nicht einmal PWA-fähig sein, um sie im Browser öffnen zu können. Auf einem alten Smartphone ist die PWA vielleicht nur eine einfache Webseite, funktioniert aber trotzdem. Je moderner dann der Browser, in der die App läuft, desto mehr Funktionen stehen zur Verfügung. Das nennt man *Progressive Enhancement* (progressive Verbesserung), daher auch der Begriff *Progressive Web App*.

Eine PWA lässt sich im Gegensatz zu normalen Webseiten auch offline nutzen. Der sogenannte Service Worker sorgt dafür, dass die einzelnen Seiten über den Cache des Browsers offline verfügbar sind. Dies führt auch dazu, dass die App nach dem ersten Öffnen ohne Verzögerung startet. Weitere Offlinefunktionalitäten lassen sich ebenfalls implementieren.

Es ist bei dieser Art App keine Installation notwendig. Sie ist über eine URL auf jedem Gerät bzw. Browser verfügbar. Die App kann mit wenig Aufwand über das Kontextmenü des Browsers auf den Startbildschirm hinzugefügt werden. Hierfür können bei der Entwicklung ein eigenes Icon sowie Name der App ausgewählt werden.

In der nachfolgenden Abbildung lässt sich das Hinzufügen einer PWA auf einem Android-Smartphone nachvollziehen:

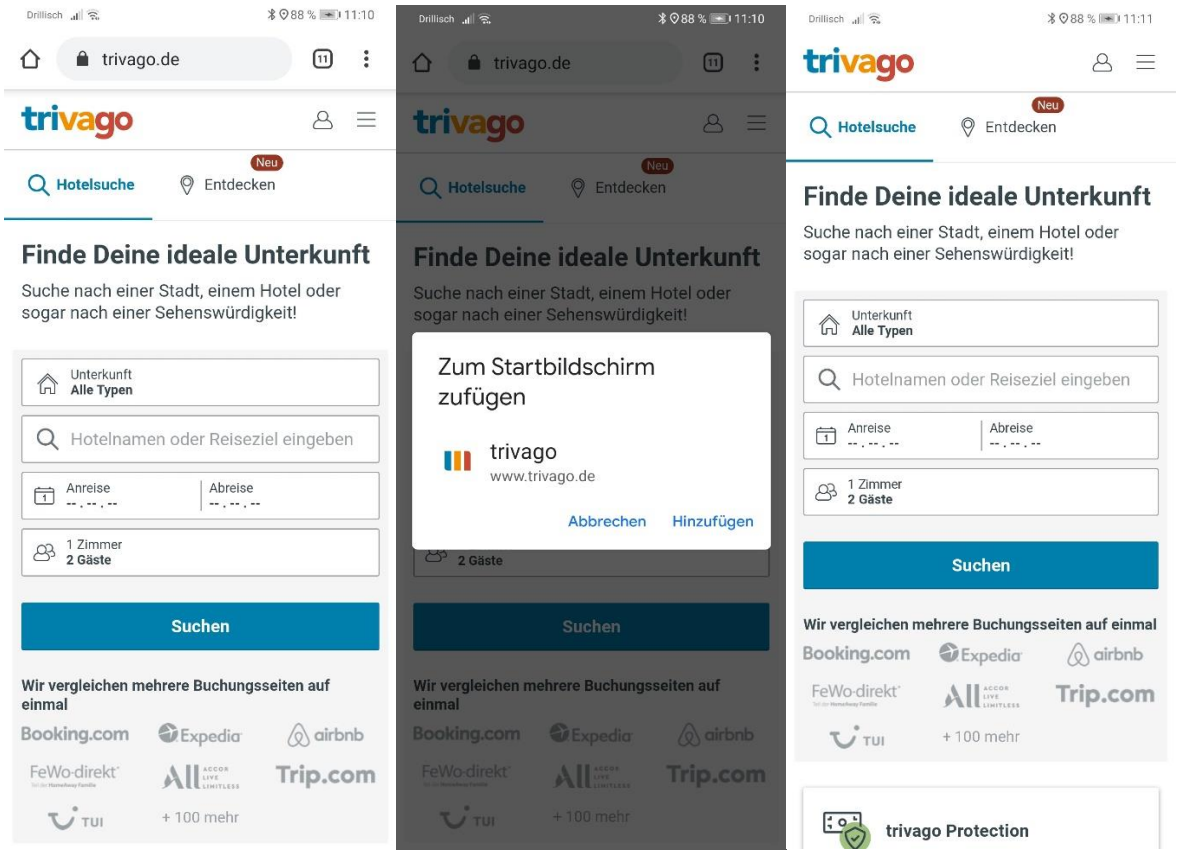

Abbildung 8: Screenshots Trivago als PWA [14]

Auf dem Startbildschirm des Smartphones, sieht man keinen Unterschied mehr, ob es sich um eine native App oder eine PWA handelt. Lediglich die Icons der unterschiedlichen Apps unterscheiden sich in diesem Beispiel:

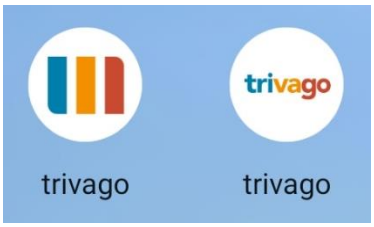

Abbildung 9: Icons Trivago-Apps. Links PWA, Rechts native App

Viele einfache Webseiten verhalten sich ebenfalls beim Hinzufügen wie eine PWA. Sie besitzen ein eigenes Icon und starten beim Öffnen im *standalone*-Modus (also ohne Browserbedienfelder). Dass es sich bei Trivago um eine PWA handelt, kann man dadurch sicher erkennen, weil die Webseite einen Service Worker enthält, den man über die Entwicklertools von Google Chrome über „Application“ aufrufen kann. Außerdem erscheint bei PWAs, die in Chrome auf einem Windows-PC geöffnet werden, ein Plus-Icon in der URL-Leiste, mit dem man die PWA zu seinem Desktop hinzufügen kann:

Abbildung 10: Screenshot von URL-Leiste der Trivago-PWA

6.3 Hybrid App

Eine Hybrid App ist eine Kombination aus nativer App und Web-App. Sie besteht aus einem nativen Rahmen, auf dem die Webseite über einen Web-View eingebunden wird. Solch ein Web-View-System wird genutzt, um in einer nativen App Webinhalte ohne eine URL-Leiste oder sonstigen Browserelementen darstellen zu können. [15]

Dieser Aufbau einer Hybrid App lässt sich in nachfolgender Grafik nachvollziehen:

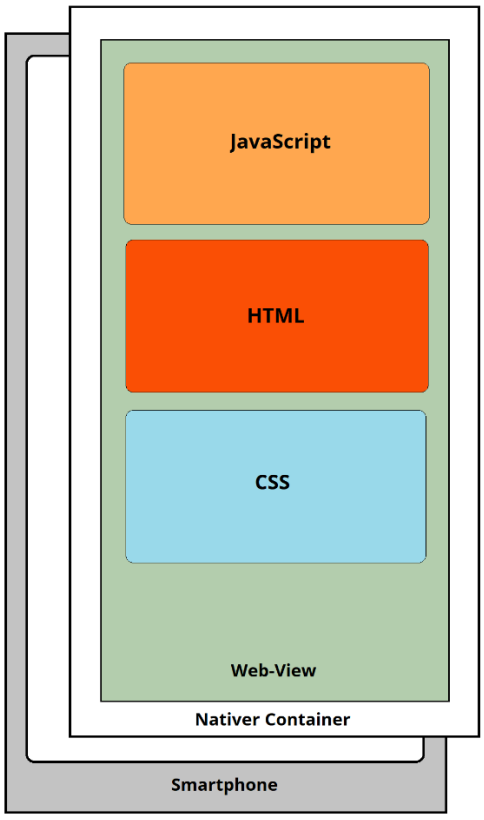

Abbildung 11: Aufbau einer Hybrid App

Durch die verwendeten Webtechnologien kann die Hybrid App plattformunabhängig entwickelt werden. Lediglich der native Rahmen ist je nach Betriebssystem unterschiedlich. Dadurch entstehen selbstverständlich auch zwei unterschiedliche App-Versionen, die gepflegt werden müssen. Diese entstandenen Apps lassen sich wie native Apps über den jeweiligen App-Store verbreiten.

Die Entwicklung von Hybrid Apps ist durch verschiedene Frameworks möglich. Beispiele hierfür sind z.B. Ionic und PhoneGap. Adobe hat jedoch angekündigt sein

Produkt PhoneGap einzustellen. Grund hierfür ist vor allem die Weiterentwicklung der moderneren PWA-Technologie. [16]

6.4 Native App vs. PWA

In diesem Unterkapitel wird eine PWA einer nativen App gegenübergestellt, um nachfolgend zu begründen, wie die Entscheidung bezüglich des Entwicklungsansatzes der App ausgefallen ist.

Da PWAs die modernere Technologie gegenüber Hybrid Apps darstellt, wurde zusammen mit dem Unternehmen SYSTECS eine Hybrid App für dieses Projekt ausgeschlossen.

6.4.1 Vergleichsaspekte

Um eine Entscheidung zwischen PWA oder nativer App einfacher zu gestalten, wurden anhand der Anforderungen Vergleichsaspekte aufgestellt, die hier genauer beschrieben werden.

Ein wichtiger Aspekt sind die **Funktionalitäten**, die mit der App möglich sind. Es muss sichergestellt werden, dass alle geforderten Funktionen mit der gewählten Art realisierbar sind.

Zum Zweiten wird auf die **Plattformabhängigkeit** nochmal gezielt eingegangen, die sich vor allem auch auf den **Realisierungsaufwand** auswirkt, da die App schließlich für Android und iOS gleichermaßen zur Verfügung stehen soll.

Bei der Realisierung ist es wichtig auch zu beachten, wie die **Debug- und Testmöglichkeiten** der App sind. Während der Entwicklung ist es immer wieder erforderlich, den aktuellen Stand zu testen und Fehler zu finden.

Da selbstverständlich die **Performance** der App laut Anforderungen auch eine Rolle spielt, ist es wichtig auch hierauf noch einmal gezielt einzugehen. Bei der Performance ist dann auch relevant, wie sehr der **Speicherplatz** des Smartphones belastet wird.

Bei der entstandenen Entwicklung wird wichtig sein, wie es um die **Datensicherheit** der Anwendung steht, da die App aus vielen Serveranfragen und –antworten bestehen wird.

Zudem ist auch ausschlaggebend, wie die **Verbreitungsmöglichkeiten** der App sind und wie es sich mit der **Wartbarkeit** verhält. Das Unternehmen möchte hier einen möglichst geringen Aufwand.

Abschließend wird ein Blick auf die **Zukunft der Technologie** geworfen. Die App soll schließlich lange Zeit in Verwendung sein, deshalb ist es wichtig, dass die gewählte Technologie dies ermöglicht.

6.4.2 Tabellarische Gegenüberstellung

	Native App	**PWA**
Funktionalität	Alle Funktionen und Hardware eines Smartphones lassen sich integrieren	Bluetooth, Fingerabdrucksensoren, NFC, Zugriff auf Kontakte und Kalender kann nicht integriert werden, Funktionalitäten der App passen sich jeweils den Rahmenbedingungen an, Startseite lädt offline durch so-genannte Service-Workers, weitere Offlinefunktionalität möglich, durch Apple unterstützt seit IOS 12.2 [17], aber keine Push-Benachrichtigung und Hintergrundsynchronisierung
Plattform-abhängigkeit	Spezielle Entwicklung für ein Betriebssystem, Frameworks machen plattform-unabhängige Entwicklung möglich (z.B. Xamarin)	Durch Webtechnologie vollkom-men plattformunabhängig
Realisierungs-aufwand	Weil zwei Betriebssysteme, auch mit Xamarin mehr Aufwand als PWA	Geringer Arbeitsaufwand
Debug- und Testmöglich-keiten	Android Emulatoren auf Windows, [18] iOS Emulatoren auf Mac, [19] Anschließen eines entsprech-enden Smartphones	Testen und Debuggen ohne Emulator im Browser möglich [13]

Performance	Evtl. Laden der App mit Verzögerung, verminderte Performance durch Xamarin oder andere Frameworks	Startseite lädt nach dem ersten Öffnen sofort, ansonsten voraus-sichtlich ähnlich wie native App
Speicherplatz	Alles direkt auf dem Gerät ge-speichert, können Daten bis zum Speicherlimit des Gerätes sichern	Sehr geringer Speicherbedarf (je nach Größe des Offline-Caches)
Daten-sicherheit	Möglichkeit des Einbaus verschiedener Sicherheitsmaßnahmen z.B. Multifaktor-Authentifizierung [10]	SSL-Verschlüsselt (Verschlüsselte Kommunikation über HTTPS [20]) [13]
Verbreitungs-möglichkeiten	Verbreitung über jeweiligen App-Store möglich, nach Installation automatische Auflistung des App-Icon auf Startseite	Keine Installation notwendig, über URL auf jedem Gerät aufrufbar, App kann mit einem Icon auf dem Bildschirm des Smartphones gespeichert werden, bei Android Veröffentlichung im Play Store möglich [21]
Wartbarkeit	Veröffentlichung und Pflege mehrerer App-Versionen der unterschiedlichen Betriebssysteme im jeweiligen App-Store	Keine Aktualisierung über App Stores nötig, greift stets über URL auf den neusten Stand zu
Zukunft der Technologie	Neue Android - und iOS - Versionen kommen auch mit alten Apps zurecht,	App Anbieter und Nutzer setzen große Hoffnungen in PWAs,

in ferner Zukunft ungewiss wie lange Verwendung möglich ist	Android wird immer mehr für PWAs optimiert, [22] Apple ist mit PWAs nicht sehr zufrieden, akzeptiert sie aber seit IOS 12.2 [17], Webseite an sich ist immer und überall verwendbar

Tabelle 2: Vergleich native App und PWA

6.4.3 Warum PWA?

Anhand der Anforderungen und der Auseinandersetzung mit den verschiedenen App-Formen, ließ sich eine Wahl treffen. Zusammen mit dem Unternehmen wurde entschieden, dass die App als PWA realisiert wird.

Zum einen lässt sich feststellen, dass sich alle Funktionen der vereinbarten Anforderungen durch eine PWA realisieren lassen. Es wird keine Hardware integriert, es bedarf keinen Zugriff auf den Kalender oder Kontakte und es ist keine Hintergrundsynchronisierung erforderlich. Die einzige Anforderung, die mit einer PWA nicht vollendend erreicht werden kann, sind die Push-Benachrichtigungen. Diese sind bei iOS mit aktuellem Stand der Technologie nicht möglich.

Der Entwicklungsaufwand ist durch diese Wahl erheblich geringer, außerdem muss die Firma sich nicht mit den App-Stores auseinandersetzen, denn die PWA lässt sich unter den Mitarbeitern ganz einfach über die URL verbreiten. Hierzu kommt, dass die Technologien zum bereits vorhandenen System gleichbleiben, was die Wartung und spätere Weiterentwicklung für das Unternehmen einfacher macht.

6.5 Erweiterte Anforderungen aufgrund PWA

Durch die Wahl des Entwicklungsansatzes haben sich die Anforderungen bezüglich der App erweitert.

Zum einen ist klar, dass außer der *.NET*-Basis vorher noch keine Technologien bestimmt wurden. Hierbei ist zunächst festzustellen, dass zur Erstellung der Web-App ASP.NET Core verwendet wird. Hinzu kommen die Webtechnologien HTML, CSS und JavaScript. Für ein einfacheres Entwickeln des Frontend wird der CSS-Präprozessor *SASS* und das CSS-Framework *Bootstrap* verwendet. Außerdem werden Icons über *Font Awesome* eingebunden.

Laut ursprünglichen Anforderungen soll die Anwendung Push-Nachrichten senden, wenn es eine Antwort auf eine Abwesenheitsanfragen gibt. Dies ist bei einer

PWA auf iOS nicht möglich, weshalb diese Anforderung nur noch für Android gilt. Da diese Anforderung ohnehin nur Priorität 3 erhielt, führt diese Änderung zu keinem allzu großen Verlust.

Die Offlinefunktionalitäten einer PWA sollen nach Möglichkeit genutzt werden (Priorität 3). Dabei ist lediglich angedacht, dass die Startseite sich offline starten lässt. Sollte ansonsten keine Internetverbindung bestehen wird eine eigens erstellte Offlineseite angezeigt.

Weitere Offline-Funktionen für die PWA wären als spätere Ergänzung ebenfalls möglich. Es wäre beispielsweise denkbar, dass Daten nach dem Absenden zwischengespeichert werden könnten, falls keine Internetverbindung besteht. Dies ist allerdings nicht Teil dieser Arbeit.

Die PWA muss mindestens kompatibel mit den Browsern Google Chrome, Safari und Samsung Internet sein. Dies ist dadurch bedingt, dass sie die meistverwendeten Browser auf Smartphones sind. Alle anderen Browser liegen unter 1,6% Nutzung und sind somit zu vernachlässigen. [23]

Für Google Chrome ist es möglich eine UI-Komponente zu entwickeln, die den Nutzer fragt, ob er die PWA zu seinem Startbildschirm hinzufügen möchte. Chrome-Nutzer sollen diese Meldung erhalten, sofern sie die PWA noch nicht hinzugefügt haben. (Priorität 3)

Schlussendlich ist noch festzuhalten, dass die App über *Microsoft Azure* gehostet wird und SSL verschlüsselt sein muss.

7. Konzeption der Progressive Web App zur Arbeitszeitverwaltung

In diesem Kapitel wird die Konzeption der mobilen App behandelt und anhand von Grafiken veranschaulicht. Es beinhaltet eine Sitemap und das User Interface. Die Abbildungen dieses Kapitels wurden mit Hilfe der Anwendung *Balsamiq Cloud* [24] erstellt. Außerdem sind alle Begriffe der Grafiken in Englisch, da die Anwendung ausschließlich in englischer Sprache zur Verfügung stehen wird.

7.1 Sitemap

Eine Sitemap stellt eine hierarchische Seitenübersicht einer Webseite dar. [25] Diese gibt es in verschiedenen Formen, für diese Arbeit jedoch reicht eine einfache graphische Darstellung der Sitemap aus. Sie soll einen Überblick der Seiten schaffen, die im nachfolgenden Unterkapitel genauer beschrieben werden.

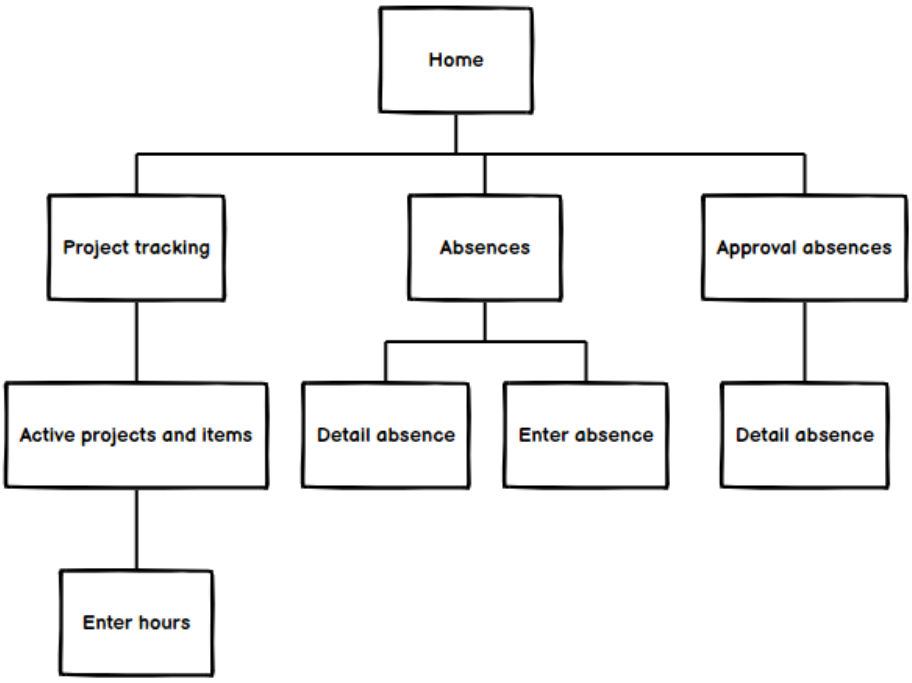

Abbildung 12: Balsamiq Screenshot der Sitemap

Nach dem Öffnen der App und erfolgreicher Anmeldung wird die Startseite angezeigt. Von dieser aus gelangt man zu den drei Bereichen der Anwendung: Projekttracking, Abwesenheiten eintragen und Abwesenheiten genehmigen. Dem Nutzer wird hier jeweils zunächst eine Übersicht angezeigt.

Von der Übersicht des Projekttrackings gelangt der Nutzer zum Eintragen der Stunden, wo er zunächst ein Projekt und Item wählt und dann auf der nächsten Seite die Stunden eintragen kann.

Bei der Abwesenheitsübersicht gibt es zwei Möglichkeiten für den Nutzer. Zum einen kann er die Details einer seiner Abwesenheiten einsehen oder eine neue eintragen.

Im Bereich zum Genehmigen von Abwesenheiten gelangt der Nutzer von der Übersicht aus ebenfalls zu einer Detailansicht, bei der hier dann das Genehmigen bzw. Ablehnen der Abwesenheit erfolgt. Dieser Bereich ist nur für Nutzer mit bestimmten Rechten sichtbar (Bereichsleiter, Geschäftsführer und Personal).

7.2 User Interface

Das User Interface beschreibt die Schnittstelle, durch die der Nutzer mit einer Maschine oder einem Server kommuniziert. Das User Interface stellt somit den Vermittler zwischen dem Menschen und der Maschine dar. Bei einer Software und auch in dieser Arbeit spricht man in der Regel von der grafischen Oberfläche des Systems. Somit kann der Nutzer die Anwendung einfach bedienen, ohne die Prozesse im Hintergrund verstehen zu müssen. [26]

Die Erstellung des UI-Konzeptes stellt einen großen Teil dieser Arbeit dar. Es wurde dem Unternehmen mehrfach vorgestellt, bis alle Beteiligten zu einer endgültigen Entscheidung gekommen sind.

Das UI der App ist vorwiegend für ein Smartphone im Hochformat konzipiert, lässt sich allerdings selbstverständlich auch im Querformat bedienen. Bei späterer Erweiterung der Anwendung wäre denkbar, das UI für das Querformat noch speziell anzupassen.

7.2.1 Grundlayout und Startseite

Die Startseite der Anwendung beinhaltet vier Buttons. Drei davon führen jeweils zu einer der Bereiche der Anwendung und über den vierten kann man sich abmelden.

Die Kopfzeile der Anwendung enthält den Titel der geöffneten Seite und ein Menü, das sich über das Menü-Icon öffnen und schließen lässt. Die Links im Menü sind das entsprechende Äquivalent zu den Buttons der Startseite. Die Kopfzeile wird immer zu sehen sein, so dass der Nutzer zu jederzeit zwischen den Bereichen springen kann. Außerdem befindet sich auf allen weiteren Seiten links in der Kopfzeile zusätzlich ein Pfeil, mit dem der Nutzer jeweils zur vorherigen Seite zurückkehren kann.

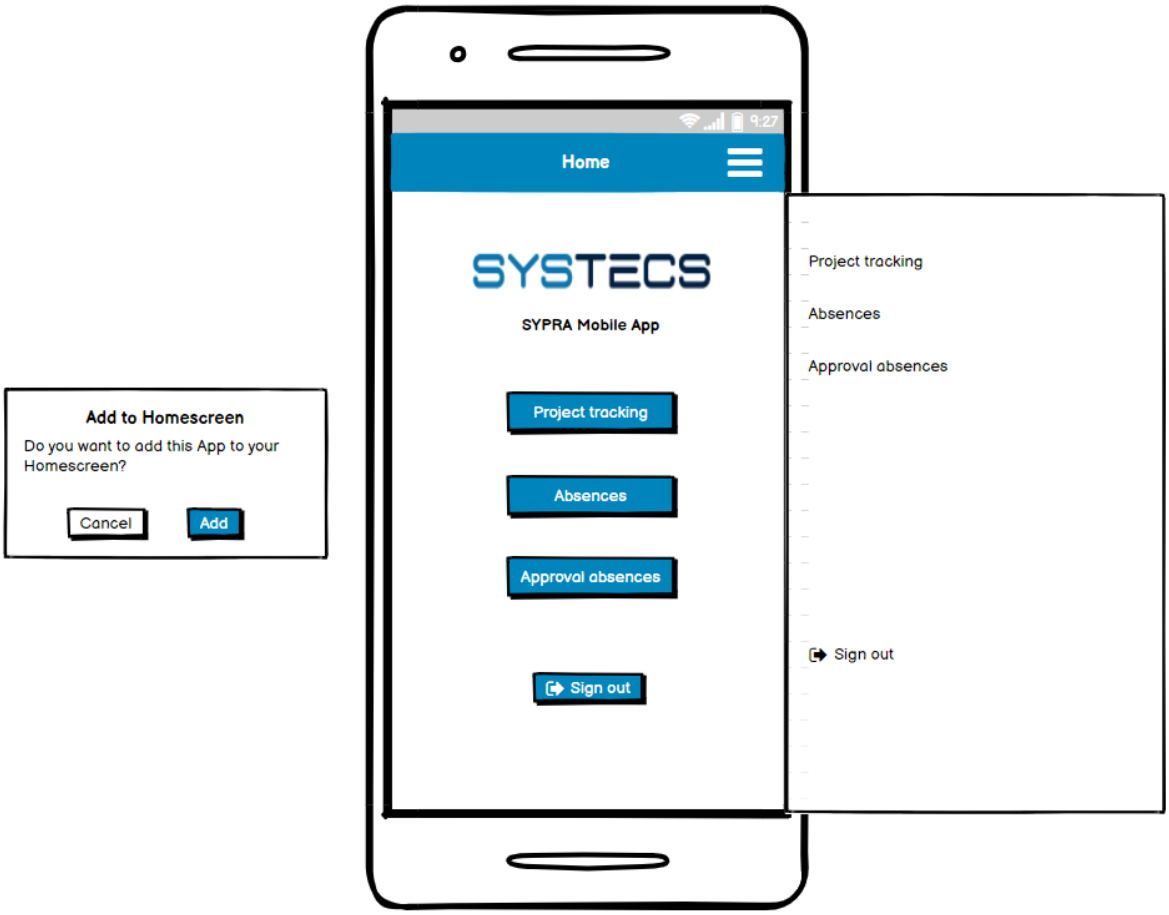

Abbildung 13: Balsamiq Screenshot der Startseite

Auf der linken Seite der Abbildung sieht man das Pop-Up, dass sich beim Starten der App öffnen soll, wenn man sie noch nicht zu seinem Startbildschirm hinzugefügt hat. Alle weiteren Pop-Ups der Anwendung werden im gleichen Stil dargestellt.

7.2.2 Projekttracking

Die Übersicht des Projekttrackings besteht aus drei Teilen, zwischen denen der Nutzer über ein Drop-Down-Menü wechseln kann.

In der Monatsübersicht ist ein Kalender zu sehen, der tageweise die Summe der Stundeneinträge enthält. Tage, bei denen keine Stunden eingetragen wurden, bleiben leer. Es werden die Kalenderwoche und der Wochentag angezeigt und das aktuelle Datum wird farblich markiert. Zusätzlich werden auch das Wochenende und die Feiertage markiert.

In der Wochenübersicht erhält der Nutzer einen Überblick seiner geleisteten Stunden in den verschiedenen Projekten. Tageweise werden hier die Projekte und die zugehörige Stundenzahl sowie die Tagessumme angezeigt.

Die einzelne Auflistung aller Features und der dazugehörigen eingetragenen Stunden erhält der Nutzer in der Tagesansicht, die in der Abbildung rechts zu sehen ist. Zusätzlich wird auch hier die Tagessumme angezeigt.

Zwischen verschiedenen Monaten, Wochen und Tagen lässt sich oberhalb der Übersicht jeweils über Rechts- und Links-Pfeile wechseln. Ausgangspunkt für die verschiedenen Ansichten ist jeweils das aktuelle Datum.

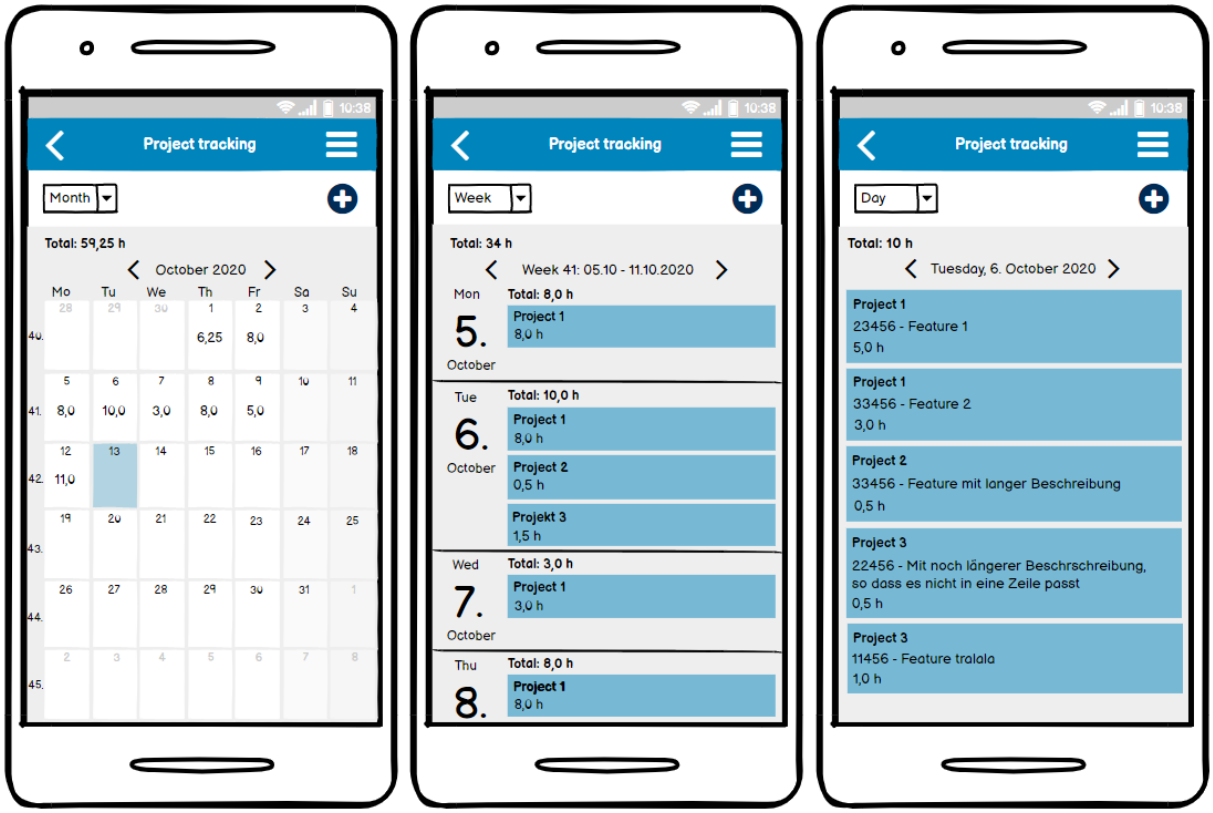

Abbildung 14: Balsamiq Screenshot der Stundenübersicht

Über den Plus-Button in der Übersicht gelangt der Nutzer zunächst zum Auswählen eines Features, auf dem im nächsten Schritt die Stunden eingetragen werden sollen.

Es werden zunächst alle Projekte aufgelistet. Über den Pfeil auf der rechten Seite der Projektnamen lassen sich die Features anzeigen. Items die wiederrum untergeordnete Elemente besitzen, lassen sich ebenso wie die Projekte aufklappen. Zu den Projekten, sowie zu anderen Featuren, sofern sie noch keine eingetragenen Stunden haben, lassen sich neue untergeordnete Features hinzufügen. Beim Hinzufügen erscheint ein Pop-Up, in das man den Featurenamen und die ID einfügt. Die ID und der Name sind Pflichtfelder, ohne die ein Item nicht gespeichert werden kann.

Features lassen sich auch löschen, sofern sie keine untergeordneten Items besitzen und noch keine Stunden eingetragen wurden. Hierfür wird ebenfalls ein Pop-Up geöffnet, bei dem der Nutzer nochmal bestätigen muss, dass er dieses

Feature löschen will. Dadurch wird ein versehentliches Löschen vermieden. Der entsprechende Button, der zum Löschen führt, wir nur angezeigt, wenn das entsprechende Feature gelöscht werden kann.

Über eine Suchfunktion oben rechts, ist es möglich direkt nach einem Feature zu suchen, ohne sich durchklicken zu müssen.

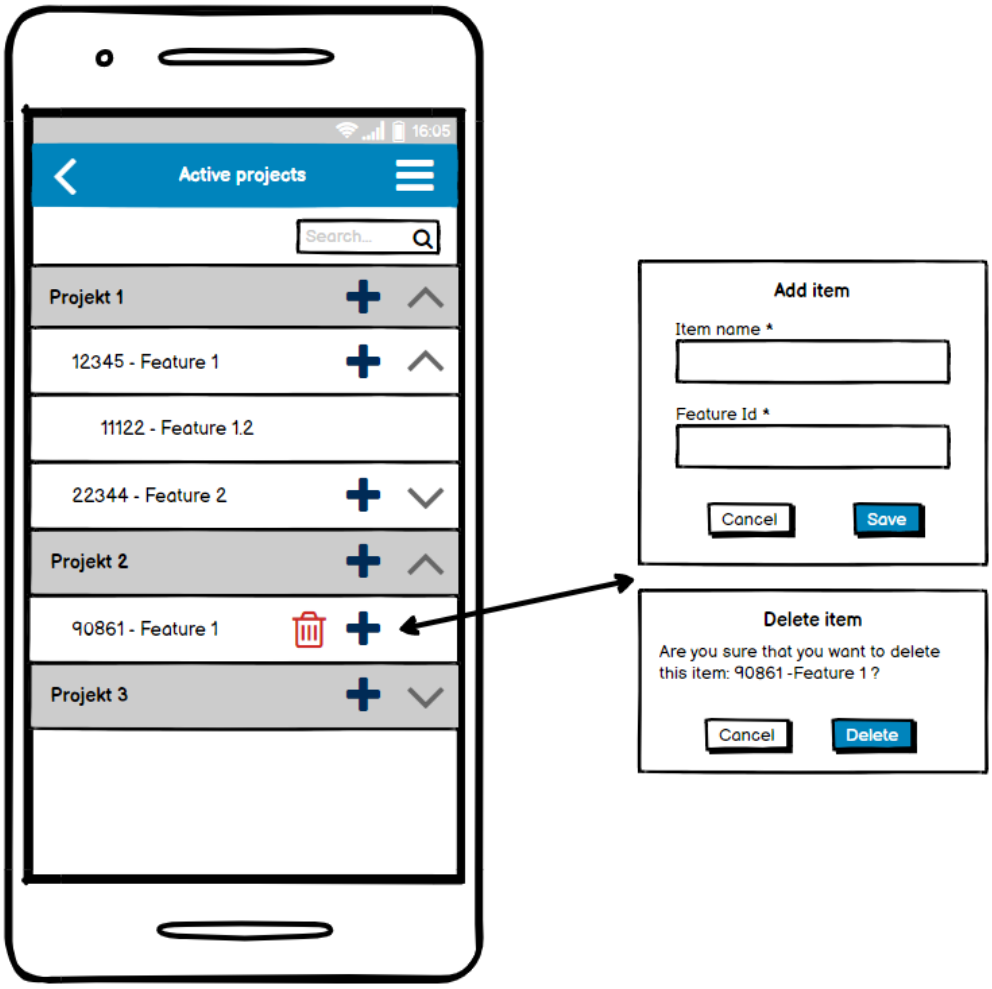

Abbildung 15: Balsamiq Screenshot der Feature-Auswahl

Wählt der Nutzer durch Tippen auf ein Feature dieses aus, gelangt er zur Stundeneintragung dieses Features.

Das gewählte Item sowie zugehöriges Projekt werden dem Nutzer auf der Seite für das Stundeneintragen zusätzlich nochmal angezeigt. Darunter befindet sich der Bereich in dem die Stunden eingetragen werden können. Es wird jeweils eine Woche, ausgehend vom aktuellen Datum, angezeigt. Zwischen den Wochen lässt sich über Rechts- und Linkspfeile wechseln.

An den Tagen, an denen bereits Stunden für das Feature eingetragen wurden, steht bereits die entsprechende Stundenzahl. Der Nutzer kann für jeden Tag seine Stundenzahl eintragen oder ändern.

Sowohl das Wochenende, die Feiertage und das aktuelle Datum werden hier, wie auch bei der Monatsansicht, farblich hervorgehoben.

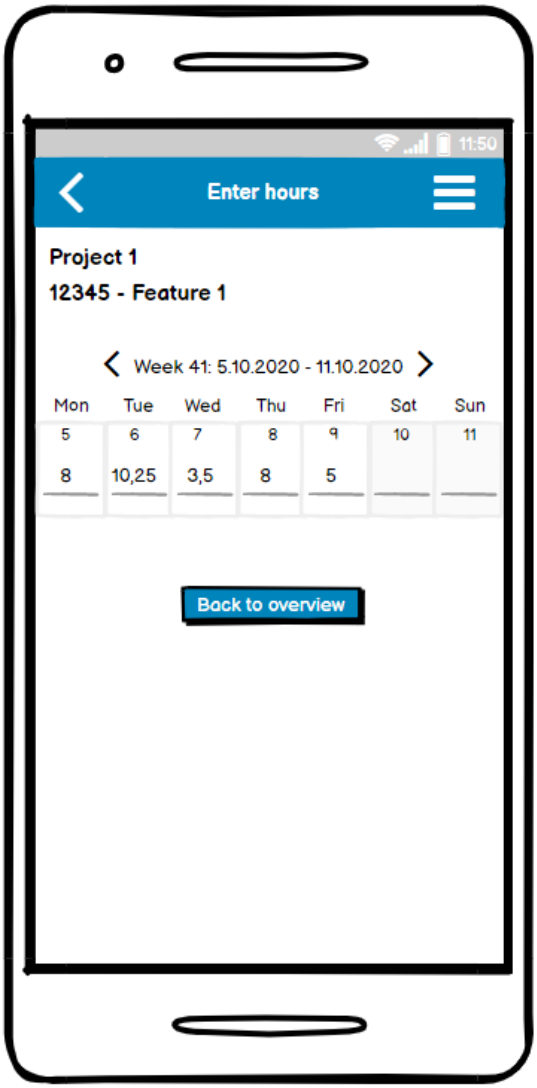

Abbildung 16: Balsamiq Screenshot der Stundeneintragung

Diese Art des Stundeneintragens ist dem bereits vorhandenen System sehr ähnlich. Es ist lediglich auf Smartphone-Größe angepasst worden.

7.2.3 Abwesenheiten eintragen

Gelangt der Nutzer zum Bereich der Abwesenheiten wird ihm eine Übersicht seiner bereits eingereichten Abwesenheiten in Listenform angezeigt. Zu sehen ist hierbei nur der Typ der Abwesenheit und der entsprechende Zeitraum.

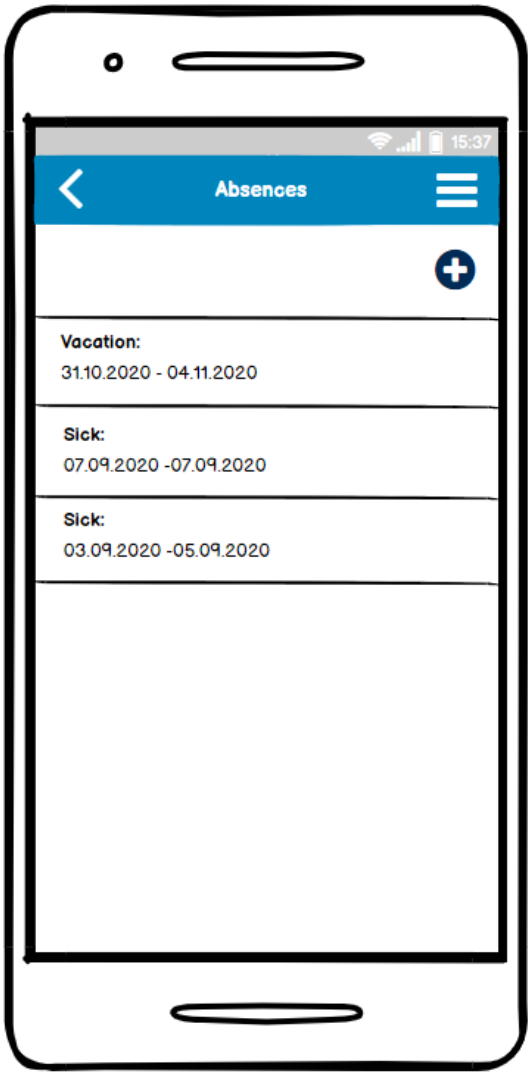

Abbildung 17: Balsamiq Screenshot der Abwesenheitsübersicht

Tippt der Nutzer auf eine Abwesenheit der Liste, gelangt er zu einer Detailansicht, die alle Informationen der Abwesenheit anzeigt.

Rechts oben in der Detailansicht befindet sich ein Button, über den die Abwesenheit zurückgezogen werden kann. Beim Zurückziehen kann im erschienenen Pop-Up optional ein Kommentar angegeben werden.

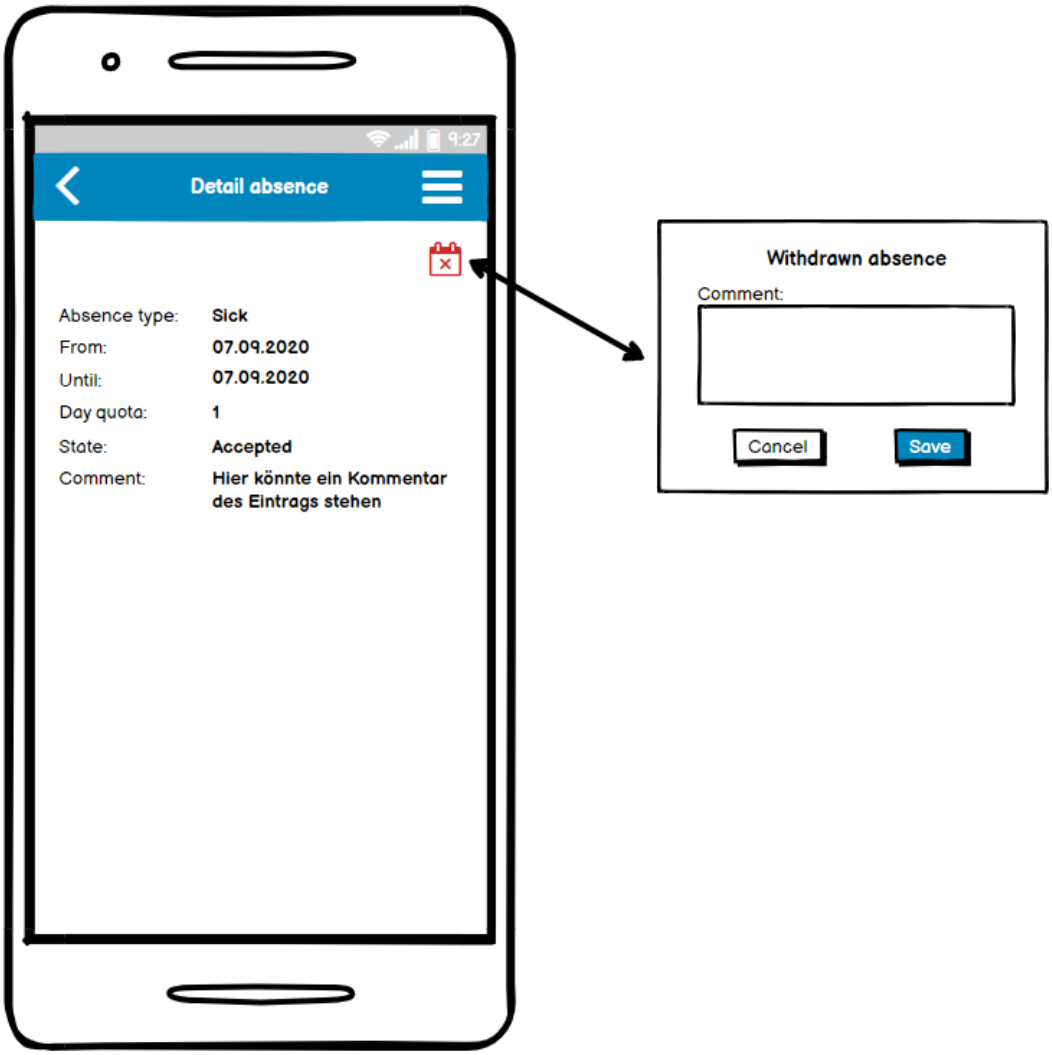

Abbildung 18: Balsamiq Screenshot der Detailansicht einer Abwesenheit

Tippt man in der Abwesenheitsübersicht auf den Plus-Button, der sich oben rechts befindet, kann der Nutzer eine neue Abwesenheit eintragen. Hier müssen alle Felder, die mit einem „*“ gekennzeichnet sind, ausgefüllt werden. Diese werden bereits in den Anforderungen als Pflichtfelder definiert. Optional kann hier zusätzlich ein Kommentar angegeben werden. Das Start- und Endedatum kann über einen Kalender ausgewählt werden. Der Abwesenheitstyp und die Quote des Start- und Endetages lassen sich über ein Drop-Down-Menü auswählen. Die Quote beschreibt hierbei, ob es sich um einen ganzen (1) oder halben Tag handelt (0,5).

Wird beim Speichern der Abwesenheit festgestellt, dass es für diesen Zeitraum bereits eine Abwesenheit dieses Typs gibt, wird dies dem Nutzer mitgeteilt und er kann seinen Eintrag korrigieren.

Wird ein Eintrag gespeichert, wird die Anwesenheitsübersicht wieder angezeigt. Durch ein Tippen auf den „Cancel“- Button gelangt der Nutzer ebenfalls zurück zur Übersicht, jedoch ohne dass der Eintrag gespeichert wird.

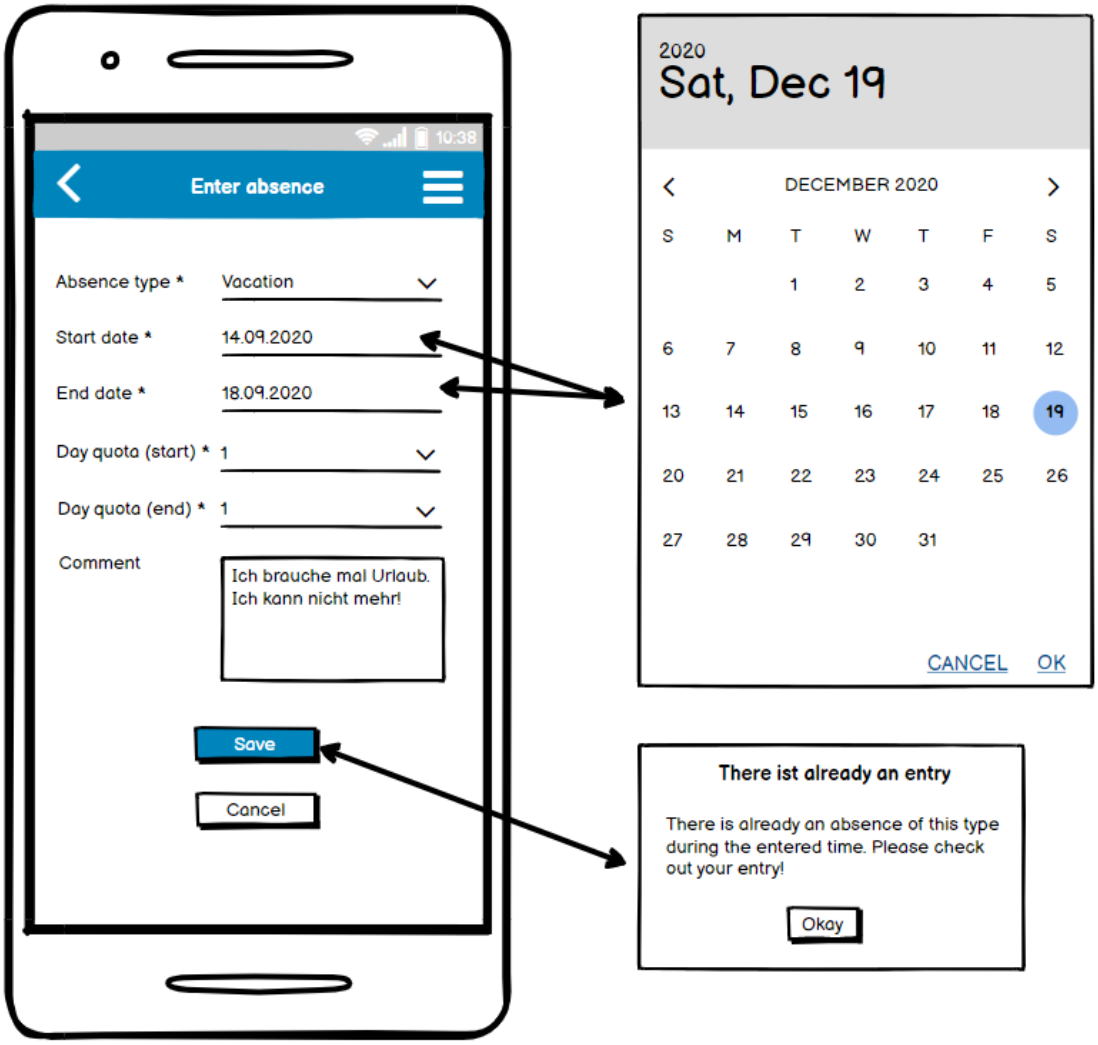

Abbildung 19: Balsamiq Screenshot des Eintragens einer Abwesenheit

7.2.4 Abwesenheitsanfragen

Im Bereich der zu genehmigenden Abwesenheiten wird zunächst eine Liste, ähnlich die zur Abwesenheitsübersicht, angezeigt. Zusätzlich zum Abwesenheitstyp und dem Zeitraum, wird hier der Mitarbeiter angezeigt, von dem die Anfrage stammt.

Abbildung 20: Balsamiq Screenshot der Übersicht der Abwesenheitsanfragen

Tippt der Nutzer auf einen dieser Listeneinträge, gelangt er zur Detailansicht der Anfrage. Zusätzlich wird hier der Zeitraum der Anfrage in einem Kalender angezeigt.

Über zwei Buttons kann die Abwesenheit abgelehnt oder genehmigt werden. Beim Ablehnen kann der Nutzer zusätzlich einen Kommentar angeben, der über ein Pop-Up eingegeben wird.

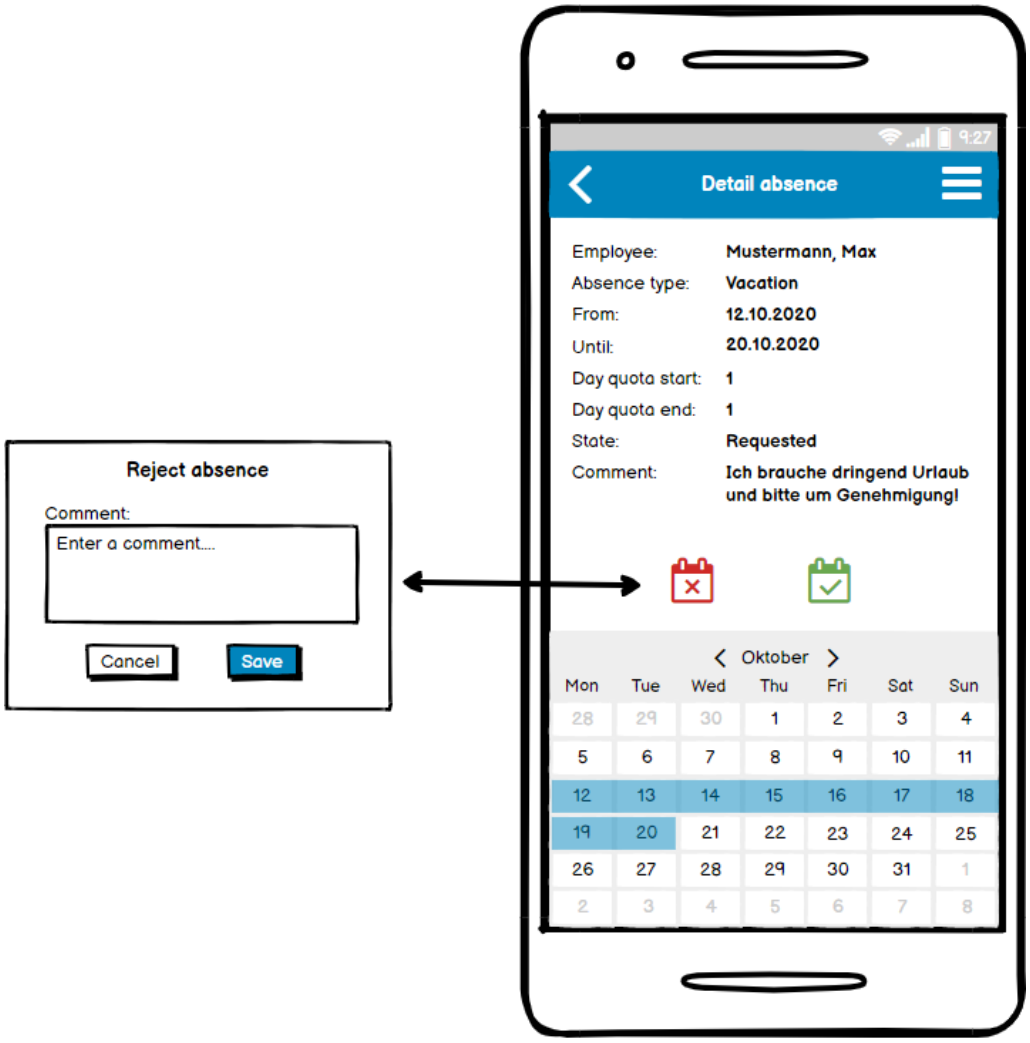

Abbildung 21: Balsamiq Screenshot der Detailansicht der Abwesenheitsanfrage

Nach dem Ablehnen oder Genehmigen gelangt der Nutzer jeweils wieder zur Listenübersicht zurück.

Das in diesem Kapitel vorgestellte UI beinhaltet alle Komponenten der Prioritäten 1-3, unabhängig davon welche Komponenten im Laufe dieser Arbeit umgesetzt werden können.

8. Technologien zur Realisierung der PWA

In diesem Kapitel werden die Technologien vorgestellt, die verwendet werden, um das Konzept umzusetzen.

8.1 ASP.NET Core

Das Microsoft .NET Framework dient als Grundprogramm für andere Programme und Software. Entwickler haben dadurch eine Plattform, mit der sie Anwendungen entwickeln können. [27]

Das ASP.NET Core basiert auf dem .NET Core Framework und wird mit der Programmiersprache C# (gesprochen „C Sharp“) verwendet. Es handelt sich dabei um ein Open-Source-Framework und dient unter anderem zur Erstellung von cloudfähigen Anwendungen, wie Web-Apps. [28]

Im Unternehmen SYSTECS wird viel mit dem .NET Core Framework entwickelt, auch das System, auf das die PWA aufbaut, ist eine ASP.NET Core Webapplikation. Somit war schnell klar, dass die mobile App ebenso eine .NET-Basis erhält. Außerdem bietet sich dies an, da sich dadurch viele Möglichkeiten für Entwickler ergeben und sich so recht einfach Webapplikationen erstellen lassen.

8.1.1 Model-View-Controller (MVC-Pattern)

Innerhalb des ASP.NET Core wird das MVC-Entwurfsmuster verwendet, welches den Code nach Verantwortlichkeit trennt. Die Anwendung ist hierbei in die folgenden drei Bereiche unterteilt: Modelle (Models), Ansichten (Views) und Controller (Controllers). [28]

Modelle beschreiben die Daten der Anwendung. Bei Web-Applikationen spielen vor allem die View-Models eine große Rolle. Diese werden vom Controller befüllt und an die Ansicht weitergegeben, die dann mit den Daten arbeiten kann.

Die Ansicht ist das UI der Anwendung. Hierbei wird die Razor-Syntax verwendet, um .NET-Code in die Webseite einzubetten. Die Ansicht wird dabei einfach gehalten. Logik sollte sich lediglich auf die Ansicht beziehen, alles weitere soll dem Controller überlassen werden.

Der Controller verarbeitet und beantwortet Benutzerinteraktionen. Er steuert die Anwendung und bestimmt, wie sie auf bestimmte Interaktionen reagieren soll.

Nachfolgende Abbildung veranschaulicht das Zusammenspiel von Modell, Ansicht und Controller:

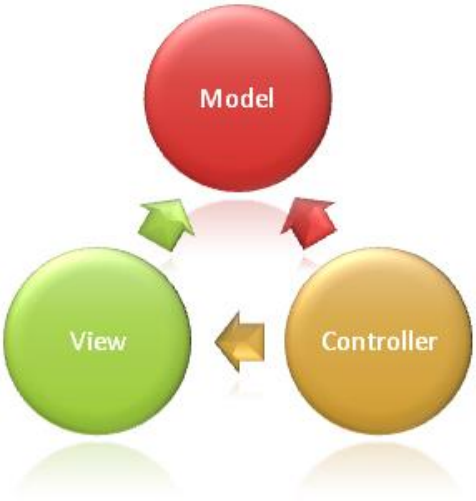

Abbildung 22: Beziehung zwischen Model, View und Controller [29]

8.1.2 ASP.NET Web Pages und Razor-Syntax

Razor wird verwendet, um serverseitigen Code in eine Webseite einzubinden. Dateien mit Razor erkennt man an der Dateiendung *cshtml*. Solch eine Datei kann das Razor Markup, C#, HTML, CSS und JavaScript enthalten. Durch das Einbinden von Razor in HTML, ist der Code sehr sauber und lässt sich daher auch einfacher warten. [28]

Razor-Code wird durch ein @-Symbol hinzugefügt. Anschließend folgt eine in sich geschlossene Anweisung (z.B. *foreach*) oder Anweisungen, die von geschweiften Klammern umschlossen sind:

```
@foreach (var item :string in Model)
{
}
@{
    if (1 == foo)
    {
    }
}
```

Quellcode 1: Razor in HTML einbetten

Codeblöcke könne auch mit HTML vermischt werden. Beginnt eine Zeile mit HTML-Markup, wird diese automatisch als HTML erkannt:

```
@foreach (var item :string in Model)
{
    var x = 0;
    <tr></tr>
}
```

Quellcode 2: Zeile mit HTML in Razor-Block

Es ist ebenfalls auch möglich eine Zeile mit Razor und HTML zu vermischen, beispielsweise um eine Variable auszugeben:

```
<h1>Hello @username</h1>
```

Quellcode 3: Razor und HTML in einer Zeile

Beim Rendern der Datei werden die Razor-Anweisungen ausgeführt und eine HTML-Seite erzeugt. Dabei wird HTML-Markup unverändert übernommen, wie beim Rendern einer gängigen HTML-Datei. [30]

8.1.3 Die Startup-Klasse

Die *Startup.cs*-Datei wird beim Einrichten eines ASP.NET Core Visual Studio Projektes automatisch erzeugt. In dieser Klasse wird konfiguriert, wie die Anwendung grundlegend HTTP-Anfragen und -Antworten verarbeiten soll und richtet alle erforderlichen Dienste der App ein. [28]

In der *ConfigureServices*-Methode werden die Dienste der App registriert, die dann als Komponenten für die App verfügbar sind und genutzt werden können. In der *Configure*-Methode hingegen findet die Konfigurierung der Reaktionen auf die HTTP-Anfragen und -Antworten statt. Diese beiden Methoden werden dann aufgerufen, wenn die Anwendung gestartet wird. [31]

8.1.4 Die appsettings.json

Die *appsettings.json* enthält, wie der Name schon sagt, die Einstellungen der Anwendung, die zur Konfigurierung der App verwendet werden. Es handelt sich um eine JSON-Datei, die aus mehreren Schlüsseln und dem jeweiligen Inhalt besteht (*Key-Value-Paare*). In dieser Datei lassen sich einige Werte festhalten, die in der Startup-Klasse verwendet werden. Hierzu zählen Daten bezüglich der Authentifizierung oder eine URL, mit der man einen Dienst erreichen kann. [32]

Einige vordefinierte Methoden, die in der Startup-Klasse verwendet werden können, greifen auf Daten der *appsettings.json* zu. Ein Beispiel hierfür ist das Hinzufügen von Service Worker und Manifest, auf das in Kapitel 8.1.6 genauer eingegangen wird.

8.1.5 Die Layout-Datei

Die Datei *_Layout.cshtml* bestimmt das Grundgerüst der Anwendung. Sie gibt also an, wo sich die Kopf- und Fußzeile, sowie das Menü und der Hauptinhalt befinden. Durch diese Datei müssen die unterschiedlichen Ansichten der Anwendung lediglich den Hauptinhalt enthalten, der dann in das Layout eingebettet wird. Ohne diese Datei müsste man das Layout bei jeder Ansicht neu definieren. Würde man dann einen Menüpunkt hinzufügen wollen, müsste man das bei jeder einzelnen Ansicht gesondert vornehmen, was sehr mühsam, zeitaufwendig und fehleranfällig wäre. Außerdem macht es auch Sinn in dieser Layout-Datei die Skripte und Stylesheets einzubinden die ansichtsübergreifend verwendet werden. [33]

8.1.6 Von ASP.NET Core Website zur PWA

Mit Hilfe des NuGet-Paketes *WebEssentials.AspNetCore.PWA* kann aus einer ASP.NET Core Webapplikation recht einfach eine PWA entstehen.

NuGet ist ein Mechanismus, über den Entwickler Code für .NET erstellen, freigeben und verwenden können. Dieser Code wird in Pakete gebündelt. [34]

Durch das benannte NuGet-Paket ist ein Registrieren eines Service Workers und Hinzufügen der Manifest-Datei sehr einfach. Hierfür stellt das Paket einige Dienste zur Verfügung, die der *ConfigureServices*-Methode der *Startup*-Klasse hinzugefügt werden können. [35]

Die Methode *AddProgressiveWebApp* fügt der Anwendung einen Service Worker und die Referenz auf die Manifest-Datei hinzu:

```
public void ConfigureServices(IServiceCollection services)
{
    // Your other services
    services.AddProgressiveWebApp();
}
```

Quellcode 4: Beispiel AddProgressiveWebApp-Methode [35]

Service Worker und Manifest können ebenfalls unabhängig voneinander über die Methoden *AddServiceWorker* und *AddWebManifest* hinzugefügt werden.

Die Einstellungen bezüglich der PWA können in der *appsettings.json* beeinflusst werden. Hier wird beispielsweise die Strategie des Service Workers gewählt und die Seiten angegeben, die dem Browser-Cache hinzugefügt werden sollen:

```
{
        //Other settings,
        "pwa": {
                "cacheId": "Worker 1.1",
                "strategy": "cacheFirstSafe",
                "routesToPreCache": "/Home/Contact, /Home/About",
                "offlineRoute": "fallBack.html",
                "registerServiceWorker": true,
                "registerWebmanifest": true
        }
}
```

Quellcode 5: Beispiel PWA-Einstellungen in appsettings.json [35]

Die *cacheId* definiert die aktuelle Version der Web-App. Sie wird im Service Worker verwendet, um zu erkennen, ob der Nutzer auf dem neusten Stand ist.

Über *strategy* wird die Strategie gewählt, auf welche Art der Service Worker mit dem Cache umgehen soll. Als Standard ist *cacheFirstSafe* definiert. Bei dieser Strategie wird jede Ressource zwischengespeichert. Beim Laden einer Ressource wird immer zunächst im Netzwerk geschaut, ob es eine neue Version gibt, andernfalls wird auf den Cache zugegriffen. Wird *networkFirst* gewählt, wird immer zuerst versucht die Ressourcen aus dem Netzwerk zu laden und dann zwischenzuspeichern. Nur wenn der Zugriff auf das Netzwerk nicht möglich ist, wird die entsprechende Ressource aus dem Cache geladen. Wenn allerdings keine Daten zwischengespeichert werden sollen, wird *minimal* gewählt.

Mit *routesToPreCache* werden die Pfade angegeben, die sofort nach dem Registrieren des Service Worker im Cache zwischengespeichert werden sollen.

Die *offlineRoute* gibt jene Ressource an, die geöffnet werden soll, wenn eine Seite aufgrund fehlender Netzwerkverbindung und fehlender Ressource im Cache angezeigt werden soll.

Über *registerServiceWorker* und *registerWebmanifest* lassen sich Service Worker und Manifest aktivieren bzw. deaktivieren.

Es ist auch möglich die Einstellungen über *PwaOptions* in der Startup-Klasse direkt vorzunehmen. Somit kann man beispielsweise für die App-Version eine Variable (*version* im folgenden Beispiel) nutzen, die aus einer Datenbank stammen könnte.

```
public void ConfigureServices(IServiceCollection services)
{
        //Your other services
        services.AddProgressiveWebApp(new PwaOptions
        {
                CacheId = "Worker " + version,
                Strategy = ServiceWorkerStrategy.CacheFirst,
                RoutesToPreCache = "/Home/Contact, /Home/About"
                OfflineRoute = "fallBack.html",
        });
}
```

Quellcode 6: Beispiel PwaOptions in Startup-Klasse [35]

Um zu testen, ob der Service Worker erfolgreich registriert wurde, hilft das Entwicklertool von Chrome. Dieses lässt sich über das Menü oder durch drücken der F12-Taste öffnen. Unter dem Reiter „Application“ kann eingesehen werden, ob die Registrierung des Service Workers gelungen ist.

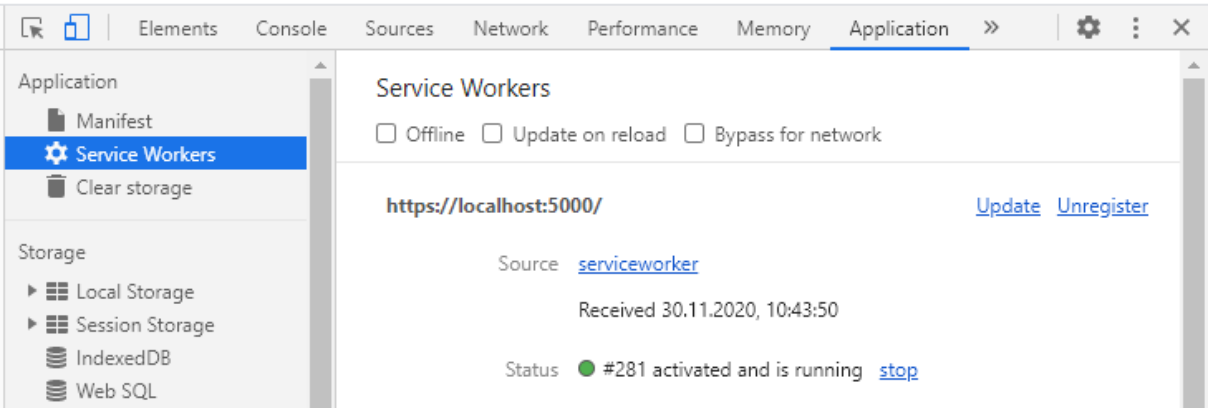

Abbildung 23: Screenshot des Entwicklertools zum Testen des Service Worker

Die Manifest der PWA ist eine JSON-Datei (*manifest.json*). In dieser Datei werden Informationen zur App eingetragen, wie den Namen, Icons und die Start-URL, die für das Herunterladen der PWA benötigt wird. Diese Datei wird dem *wwwroot*-Ordner der Applikation hinzugefügt, damit sie von außen einsehbar ist. Die Manifest hat folgendes Schema:

```
{
        "name": "My Progressive Web App",
        "short_name": "My PWA",
        "orientation": "portrait",
        "display": "standalone",
        "icons": [
                {
                        "src": "images/logox192.png",
                        "sizes": "192x192",
                        "type": "image/png"
                },
                {
                        "src": "images/logox512.png",
                        "sizes": "512x512",
                        "type": "image/png"
                }
        ],
        "start_url": "/",
        "color": "#ffffff",
        "background_color": "#20c4f4",
        "theme_color": "#20c4f4"
}
```

Quellcode 7: Beispiel Manifest-Datei [35]

Bei *orientation* wird angegeben, ob die Ansicht der App in horizontaler (*landscape*) oder vertikaler (*portrait*) Ausrichtung angezeigt werden soll. Wenn beides erlaubt ist, gibt man *any* an.

Mit *display* wird die Ansicht der App nach dem Herunterladen ausgewählt. Der Standard Wert hierfür ist *browser.* Dabei wird die Web-App wie jede gängige Webseite direkt im Browser mit allen dazugehörigen Bedienelementen geöffnet. Gibt man *standalone* oder *fullscreen* an, werden die Bedienelemente ausgeblendet. Bei *standalone* wird allerdings die Statusleiste des Smartphones zusätzlich angezeigt, welche bei *fullscreen* überdeckt wird. Wählt man *minimal-ui* kommen Browser-Kontrollelemente zur Ansicht hinzu, wie die URL-Leiste, Neuladen- und Zurück-Button. Die Elemente können je nach Browser unterschiedlich sein. [36]

Die Schlüssel *color* und *background_color* geben Schrift- und Hintergrundfarbe des Splash Screen (Startbildschirm beim Laden) an. Mit *theme_color* wird die Themenfarbe der PWA angegeben. Diese wird ausschließlich in Google Chrome verwendet, um die Farbe des Browserfenster zu bestimmen. [35]

Es gibt noch einige Einstellungen, die man in der Manifest vornehmen kann. Die hier aufgezeigten, sind allerdings die gängigsten.

8.2 CSS-Präprozessor SASS (SCSS)

Das *Cascading Style Sheet* (CSS) ist für die Gestaltung von Webseiten verantwortlich. Um die Gestaltung für Entwickler etwas komfortabler zu machen, kommen CSS-Präprozessoren zum Einsatz. *Syntactically Awesome Style Sheet* (SASS) ist einer davon. Damit die Styles vom Browser erkannt werden, müssen die SASS-Dateien zunächst in CSS umgewandelt werden, daher nennt es sich Präprozessor. [37]

SassScript nennt sich die verwendete Sprache. Diese ermöglicht einige zusätzliche Möglichkeiten, die CSS nicht bietet. Zum einen ist es möglich Variablen festzulegen, was sich vor allem für das Erfassen von Farbwerten anbietet. Dadurch muss eine Farbe nur einmal definiert werden und kann dann mehrfach verwendet werden. Mit *SassScript* sind zudem mathematische Funktionen mit den Operationen +, -, *, / oder % möglich, wodurch sich zum Beispiel Größenangaben aus anderen Werten errechnen lassen. Auch andere integrierte Funktionen erleichtern die Arbeit am Design, wie das Errechnen von Farbwerten aus anderen Farben. So lassen sich beispielsweise Farbwerte aufhellen oder abdunkeln. Auch lassen sich Styleregeln durch *while*- und *for*-Schleifen und durch eine Fallunterscheidung mit *if*- und *else*-Anweisungen erzeugen.

Weitere hilfreiche Optionen bietet SASS durch Verschachtelung, Vererbung und Teildateien (sogenannte *Partials*), die eine Menge Schreibaufwand ersparen. Durch Verschachtelungen lassen sich Zusammenhänge auch visuell im Code darstellen. Möchte man beispielsweise *paragraph*-Elemente in der Klasse *.foo* ansprechen, kann man die Regeln des *p*-Elementes einfach verschachtelt in der *.foo* Klasse darstellen. Durch die Vererbung hingegen lassen sich Regeln anderer Elemente übernehmen, ohne diese erneut aufzulisten und durch *Partials* lassen sich einzelne SASS-Dateien in eine andere SASS-Datei integrieren.

Die ursprüngliche SASS-Syntax arbeitet ohne Klammern und Semikolon. Die neuere Variante *Sassy CSS* (SCSS) hingegen ist eher so etwas wie CSS nach Art von SASS. Seit Version drei von SASS wurde SCSS als offizielle Syntax festgelegt, die mit Klammern und Semikolon arbeitet. Alle SASS-Funktionen stehen allerdings auch für SCSS zur Verfügung.

Das Beispiel in Anhang 1 verdeutlicht, dass nicht nur das Erzeugen von Styleregeln durch SCSS vereinfacht wird, sondern auch die Lesbarkeit durch die Verschachtelungsmöglichkeiten wesentlich höher ist.

Außer SASS gibt es auch noch den Präprozessor LESS, der dem SCSS sehr ähnelt. Hier sind allerdings beispielsweise keine Schleifen und Fallunterscheidungen möglich. Außerdem ist SCSS inzwischen sehr viel beliebter, mitunter weil das populäre Frontend-Framework *Bootstrap* mit Version 4 inzwischen auch auf SCSS umgestiegen ist.

8.3 CSS- und JavaScript-Bibliotheken

8.3.1 Bootstrap

Bootstrap ist ein kostenloses und quelloffenes Frontend-Framework für die Webentwicklung. Es enthält HTML und CSS-basierte Vorlagen für das Design der Typografie, Formulare, Navigation und andere Elemente. Hinzu kommen noch einige JavaScript-Erweiterungen. Entwickler können diese Komponenten für ihre Webseite nutzen und sparen somit einiges an Entwicklungsaufwand. [38]

Die Komponenten, die durch Bootstrap zur Verfügung stehen, bestehen aus einer HTML-Struktur sowie CSS-Styleregeln und fallweise aus begleitendem JavaScript-Code. Für die Styleregeln nutzt Bootstrap seit Version 4 den CSS-Präprozessor SASS. Der Entwickler kann die definierten CSS-Klassen und HTML-Struktur von Bootstrap nutzen, um die Ansicht seiner Webseite anzupassen.

Die Layoutkomponenten sind das Herzstück des Frameworks. Die Grundlegende Komponente ist hier der Container. Der Entwickler kann hier zwischen fester (*.container*) oder dynamischer (*.container -fluid*) Breite wählen. Bei der festen Breite sind vier feste Größen des Containers definiert, die jeweils von der Größe des Bildschirmes abhängig sind. Bei einer dynamischen Breite hingegen füllt der Container immer die Breite des Bildschirmes aus. Im Inneren des Containers definieren andere Layout-Komponenten ein Rasterlayout mit Zeilen und Spalten. Dieses Layout basiert auf 12 Spalten. Der Entwickler kann hierbei wählen wie viele Spalten eine Komponente einnehmen soll. Dies kann man für unterschiedliche Bildschirmgrößen gesondert angeben. Dadurch entsteht schnell und einfach eine responsive Webseite, die auf unterschiedliche Bildschirmgrößen angepasst ist.

Um einem Element in einem Container eine zugewiesene Breite zu geben, werden die *.col*-Klassen verwendet. Folgende Abbildung soll dies veranschaulichen:

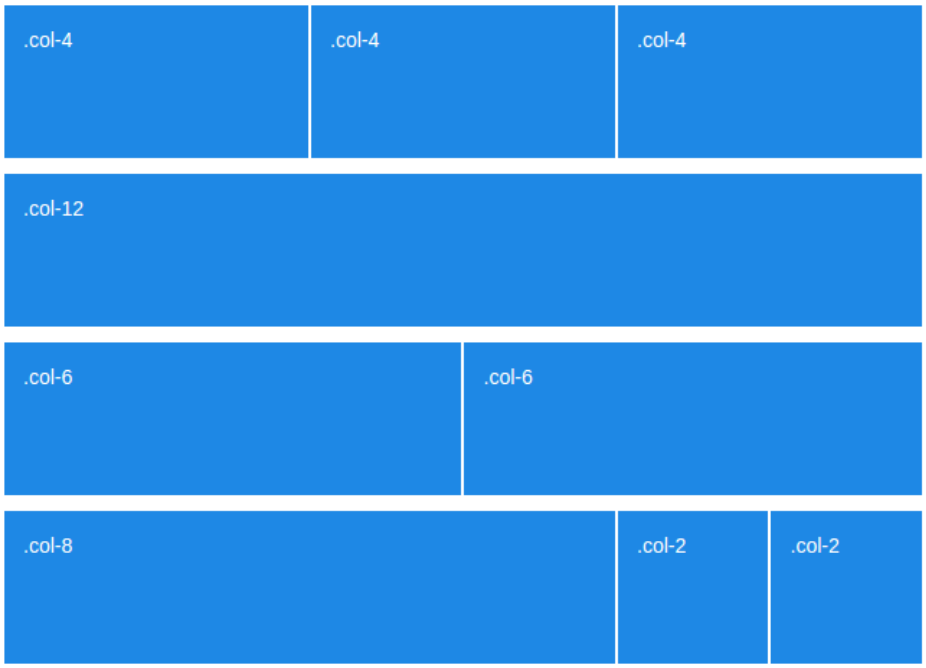

Abbildung 24: Beispiel der .col-Klassen des Bootstrap-Layout [39]

Eine weitere nützliche Komponente von Bootstrap sind die Modals. Hierbei handelt es sich um Dialogfester, die man über einer Ansicht anzeigen kann. Diese Komponente besteht aus HTML, CSS und JavaScript. Ein Modal hat eine Kopfzeile, einen Hauptinhalt und eine Fußzeile. [40]

```
<div class="modal" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">Modal title</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <p>Modal body text goes here.</p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>
```

Quellcode 8: Aufbau eines Bootstrap-Modal [40]

Das durch den vorangehenden Code erstellte Modal würde folgendermaßen aussehen:

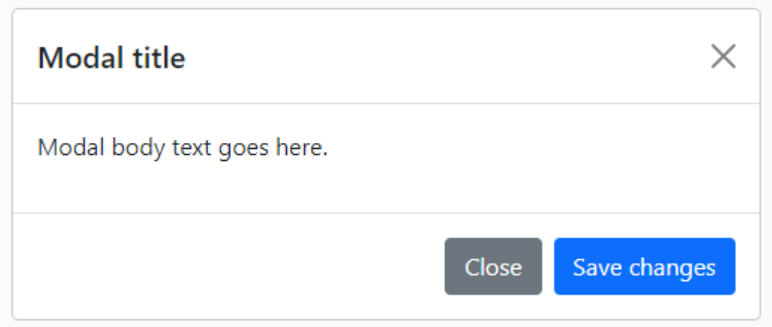

Abbildung 25: Beispiel eines Bootstrap-Modal [40]

Über die Funktionen *.modal("show")* und *.modal("hide")*, lässt sich ein Modal anzeigen und wieder schließen.

Es sind im Konzept einige Dialoge vorgesehen, die sich durch Modals realisieren lassen.

8.3.2 Font Awesome

Font Awesome ist eine der größten Open-Source-Bibliotheken für Iconfonts. Bei diesen handelt es sich um Webfonts, die Piktogramme (Symbole) statt Buchstaben enthalten und auf Webseiten verwendet werden können. Die Piktogramme basieren auf Vektorgrafiken und können daher beliebig verlustfrei skaliert werden. Font Awesome bietet dabei mehr als 600 Icons. Diese Icons lassen sich einfach über CSS-Klassen in die Webseite einbauen und können bezüglich der Größe, Farbe und Deckkraft durch eigenes CSS angepasst werden. [41]

Die Icon-Bibliothek kann auf der Webseite von Font Awesome durchsucht werden. Wählt man ein Icon aus, bekommt man eine Hilfestellung zum Verwenden dieses Icons. Beispielsweise gibt es die Möglichkeit das entsprechende HTML-Element durch einen Klick zu kopieren, um dieses dann auf seiner Seite einzufügen. [42]

8.3.3 FullCalendar

Bei FullCalendar handelt es sich um ein Open-Source-JavaScript-Framework zur Erstellung von Eventkalendern. Hierbei ist das Darstellen des Kalenders in voller Größe möglich und Events können per Drag-n-Drop verschoben werden. Die Drag-n-Drop-Funktionalität wird allerdings für diese Arbeit nicht benötigt. [43]

Über JavaScript lässt sich solch ein Kalender konfigurieren. Hier wird beispielsweise angegeben von welchem initialen Datum (*initialDate*) und welcher initialen Ansicht (*initialView*) der Kalender ausgehen soll und mit welchen Kontrollelementen (unter *headerToolbar*) der Kalender ausgestattet werden soll. Zusätzlich kann dem Kalender direkt beim Konfigurieren Veranstaltungen (*events*) hinzugefügt werden.

```
var calendarEl = document.getElementById('calendar');

var calendar = new window.FullCalendar.Calendar(calendarEl, {
    initialView: 'dayGridMonth',
    initialDate: '2020-11-07',
    headerToolbar: {
        left: 'prev,next today',
        center: 'title',
        right: 'dayGridMonth,timeGridWeek,timeGridDay'
    },
    events: [
        {
            title: 'All Day Event',
            start: '2020-11-01'
        },
        {
            title: 'Long Event',
            start: '2020-11-07',
            end: '2020-11-10'
        },
        ...,
        ...,
        ...,
        ...,
        ...,
        ...,
        ...,
        ...
    ]
});
calendar.render();
```

Quellcode 9: Beispiel einer Konfiguration eines Kalenders mit FullCalendar [44]

Der Kalender, der aus vorangehendem Code-Beispiel erstellt wird, sieht dann folgendermaßen aus:

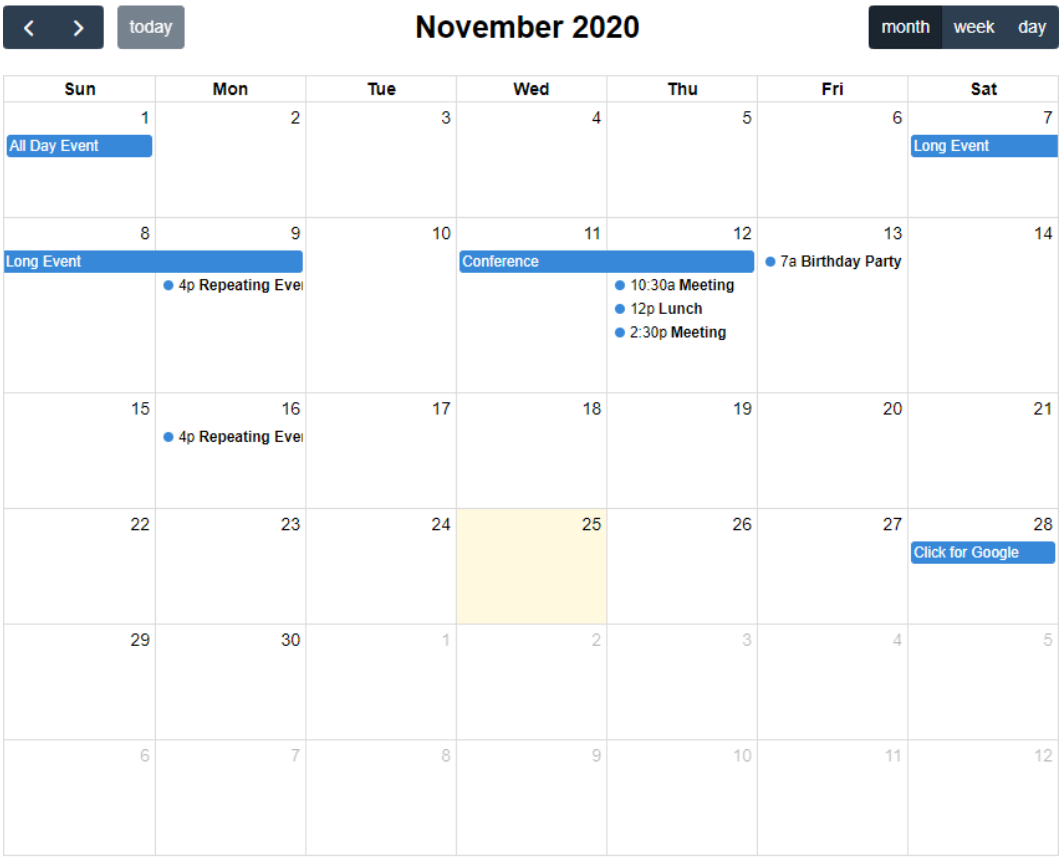

Abbildung 26: FullCalendar Beispiel [44]

Veranstaltungen können über eine *addEvent*-Methode auch nachträglich hinzugefügt werden.

Für Webentwickler ist es durch dieses Framework ohne viel Aufwand möglich, einen Kalender zu erstellen und mit Veranstaltungen zu versehen. Dies wird für die Monatsübersicht der Stundeneinträge nützlich sein. Außerdem wird dieses Framework auch bereits in der SYPRA-App verwendet, auf welche die mobile Version aufbaut.

9. Realisierung der Progressive Web App zur Arbeitszeitverwaltung

Nachdem nun alle Grundlagen für die Implementierung aufgearbeitet sind, geht es nun in diesem Kapitel um die Realisierung der Progressive Web App. Zur Veranschaulichung werden Codeausschnitte eingebunden.

9.1 Aufbau und Layout der Web-App

Die App ist für die Implementierung ebenso in die drei Bereiche Projekttracking, Abwesenheiten eintragen und Abwesenheiten akzeptieren aufgeteilt. Die entsprechenden Ansichten für die Bereiche sind jeweils in eigene Dateiordner aufgeteilt und jeder Bereich hat einen eigenen Controller. Außerdem werden für die Daten, die in den Ansichten verwendet werden, unterschiedliche Modelle eingesetzt. Teilweise werden diese bereichsübergreifend genutzt.

Zusätzlich zu den Controllern für jeden Bereich gibt es einen *BaseController* und einen *HomeController*. Alle Controller erben vom *BaseController*, der somit die Grundlage für alle Controller darstellt. Er initialisiert alle elementaren Dienste und Funktionen, die in der App benötigt werden. Somit regelt er auch die Rechte der Nutzer und die Handhabung beim Auftreten von Fehlern, die beim Aufrufen von *Views* und *PartialViews* auftreten können. Der *HomeController* ist zum einen für die Ausgabe der Startseite zuständig und leitet bei fehlenden Rechten oder anderen Fehlern zu entsprechenden Fehleransichten weiter.

Die *_Layout.cshtml* definiert den Aufbau der Seite und integriert die verwendeten Frameworks sowie eigene Styleregeln und Skripte. Im *head* der Datei werden diese eingebunden. Außerdem sind hier auch der Titel und die Icons für die Browser definiert. Im *body* befindet sich zunächst die Kopfzeile der PWA, die einen Zurück-Button, den Titel der angezeigten Ansicht und das Menü enthält. Der Zurück-Button wird auf allen Seiten außer der Startseite angezeigt und führt zur jeweils vorherigen Seite, die jeweils für jede Ansicht gesondert definiert ist. Unter der Kopfzeile befindet sich der Hauptinhalt, in den die jeweilige Ansicht geladen wird. Zusätzlich enthält das Layout noch eine Ladeanimation von Bootstrap, die so von überall erreichbar ist und so jederzeit angezeigt oder ausgeblendet werden kann. Das Grundgerüst der App kann in Anhang 2 nachvollzogen werden.

Die Hauptansichten werden direkt in das Layout als Hauptinhalt eingebunden. In den Hauptansichten werden außerdem einige Teilansichten verwendet. Diese besitzen eigene Models und können in die Hauptansichten geladen werden. Dies hat sich beispielsweise für die unterschiedlichen Ansichten der Stundenübersicht bewährt. Dadurch kann einfach über die Auswahl im Drop-Down-Menü die Anzeige gewechselt werden. Außerdem ist es mit Hilfe der Teilansichten einfach einen

jeweiligen neuen Zeitraum zu laden, beispielsweise beim Wechseln in die nächste oder vorherige Woche oder Monat.

9.2 Umsetzung der einzelnen Bereiche

Das Projekttracking besteht aus drei Hauptansichten: Übersicht, Projekte mit Features und das Stundeneintragen. Hinzu kommen verschiedene Teilansichten z.B. für die Tages-, Wochen- und Monatsübersicht, sowie die Tabelle zum Stundeneintragen und Teilansichten für die hierarchische Darstellung der Features. Die unterschiedlichen Ansichten und Teilansichten bekommen ihre erforderlichen Daten über den *ProjectTrackingController* mit Hilfe verschiedener Modelle wie den *EffortCalenderViewModel* für die Monatsansicht oder des *WorkItemModel* für die Features der Projekte.

Das *EffortCalenderViewModel* wird hier beispielhaft für alle anderen Modelle des Projektes genauer erläutert. Dieses Modell enthält die Mitarbeiter-ID, ein Start- und Enddatum (Anfangs- und Enddatum des angezeigten Monats), eine Liste der Feiertage sowie die zusammengefassten Eintragungen der Stunden (*Efforts*).

```
using System;
using System.Collections.Generic;
using Systecs.Sypra.DomainModel.ProjectManagement;
using Systecs.Sypra.DomainModel.WorkingTime;

namespace SypraMobileApp.Models
{
    4 Verweise | Olivia Preis, vor 7 Tagen | 1 Autor, 3 Änderungen
    public class EffortCalendarViewModel
    {
        2 Verweise | Olivia Preis, vor 49 Tagen | 1 Autor, 1 Änderung
        public string EmployeeId { get; set; }
        2 Verweise | Olivia Preis, vor 49 Tagen | 1 Autor, 1 Änderung
        public DateTime StartDate { get; set; }
        2 Verweise | Olivia Preis, vor 49 Tagen | 1 Autor, 1 Änderung
        public DateTime EndDate { get; set; }
        2 Verweise | Olivia Preis, vor 7 Tagen | 1 Autor, 2 Änderungen
        public IList<PublicHoliday> PublicHolidays { get; set; }
        3 Verweise | Olivia Preis, vor 43 Tagen | 1 Autor, 2 Änderungen
        public IList<Effort> Efforts { get; set; }
    }
}
```

Quellcode 10: EffortCalendarViewModel für die Monatsübersicht

PublicHoliday und *Effort* sind bereits definierte Klassen des sogenannten *DomainModel*, welches im SYPRA-Service sowie in der bestehenden Webapplikation bereits genutzt werden. In *PublicHoliday* sind jeweils Datum und Name des Feiertags sowie eine ID vorhanden. Bei den *Efforts* hingegen sind mehr Informatio-

nen enthalten wie die ID des Mitarbeiters, Projektes sowie Feature und eine Liste von *WorkUnits*, die jeweils ein Datum und die dazugehörige Stundenzahl enthalten. Zusätzlich können *Efforts* die Summe des im Service angeforderten Zeitraums und die Gesamtsumme der eingetragenen Stunden enthalten.

Die Daten des Models werden im *ProjectTrackingController* mit Hilfe des im Hintergrund laufenden Webservice befüllt:

```
var publicHolidays :ConfiguredTaskAwaitable<IList<PublicHoliday>> = WorkingTimeService
    .GetPublicHolidaysAsync(startDate.Year, state: "BW") // Task<IList<PublicHoliday>>
    .ConfigureAwait( continueOnCapturedContext: false);

var efforts :ConfiguredTaskAwaitable<IList<Effort>> =
    ProjectService.GetEffortForProjectSummariesAndEmployeeAsync(
            new Guid(employeeId), startDate, endDate) // Task<IList<Effort>>
        .ConfigureAwait( continueOnCapturedContext: false);

var model = new EffortCalendarViewModel
{
    EmployeeId = employeeId,
    StartDate = startDate,
    EndDate = endDate,
    PublicHolidays = await publicHolidays,
    Efforts = await efforts
};
```

Quellcode 11: Befüllen des EffortCalendarViewModel

Für die Monatsansicht wird das FullCalendar-Framework verwendet. Beim Konfigurieren des Kalenders wird das initiale Datum auf den aktuellen Tag gesetzt und in die Kopfzeile des Kalenders wird in die Mitte der Monatstitel gesetzt sowie links und rechts davon jeweils die Buttons für den nächsten bzw. vorherigen Monat. Zusätzlich wird festgelegt, dass die Wochennummern angezeigt werden und eine Woche mit dem Montag beginnt. Außerdem wird beim Initialisieren des Kalenders festgehalten, dass durch ein Auswählen eines Tages, die Tagesansicht geöffnet wird. Die entsprechende JavaScript-Funktion *openDayOverview* wird beim Tippen auf ein Event des Kalenders wie auch beim Tippen auf einen Tag ausgelöst. Dies ist erforderlich damit die Stelle, auf die der Finger des Nutzers trifft, keine Relevanz für auslösen bzw. nicht auslösen der Funktion hat, da beim Tippen auf das Event-Objekt kein Event für das Tippen auf den Tag ausgelöst wird.

```
ProjectTrackingCalendar = new window.FullCalendar.Calendar(trackingCalendarElem,
    {
        initialView: 'dayGridMonth',
        initialDate: new Date(),
        headerToolbar: {
            left: '',
            center: 'prev title next',
            right: ''
        },
        locale: 'en',
        weekNumbers: true,
        firstDay: 1,
        dateClick: (info) => {
            openDayOverview(info.dateStr);
        },
        eventClick: (info) => {
            openDayOverview(info.event.start);
        }
    });
```

Quellcode 12: Initialisierung des Kalenders für die Monatsansicht

Die Summen der Stundeneinträge werden, je nach angezeigtem Monat, aus dem Service geladen und als Events in den Kalender eingefügt. Die Styles für die Events des Kalenders wurden hierfür durch wenige eigene Styleregeln so überschrieben, dass sich die Ansicht dem Konzept gleicht.

Die Wochenansicht wird durch eine zweispaltige Tabelle realisiert, die in der ersten Spalte jeweils das Datum und in der zweiten die Stundeneintrage enthält. Die Daten werden auch hier über den *ProjektTrackingController* vom Service geladen und mit Hilfe des *EffortWeekViewModel* an die Ansicht übergeben.

Für die Tagesansicht reihen sich lediglich *div*-Elemente aneinander, die die verschiedenen Stundeneinträge, die aus dem *EffortDayViewModel* stammen, enthalten. Diese besitzen die Klasse *hour-entry* und bekommen somit die entsprechenden hierfür definierten Styleregeln.

```
foreach (var effort in Model.DayEfforts)
{
    <div class="hour-entry">
        <p>@effort.ProjectName</p>
        <p>@effort.FeatureId - @effort.FeatureName</p>
        <p>@effort.HoursWorked&#104;</p>
    </div>
}
```

Quellcode 13: Ansicht der Stundeneinträge des Tages

Über den jeweiligen Teilansichten mit der ausgewählten Übersicht befindet sich der Kontrollbereich mit einem runden Button, in dem sich ein Plus-Icon von Font

Awesome befindet. Dieser öffnet eine andere Ansicht, bei der die Projekte und Features angezeigt werden. Die Hauptansicht *ActiveProjects* enthält zunächst nur Listen mit jeweils einem Listenelement, welches den Projektnamen enthält. Für jede dieser Listen wird mit Hilfe des Controllers und Teilansichten die Features hierarchisch in die jeweiligen Projektlisten geladen. Das Auf- und Zuklappen der Features des Projektes und untergeordneten Features von Items wird über JavaScript-Code gelöst, der entsprechende Listenelemente aus- oder einblendet und die Icons anpasst. Die Ansicht enthält zusätzlich zu den Projektlisten zwei Bootstrap-Modals, die durch ein Tippen auf den Plus- oder Lösch-Button geöffnet werden. Das Modal zum Erstellen eines neuen Features, löst nach dem Ausfüllen der Input-Felder und das Tippen auf den Speichern-Button eine Funktion zum Erstellen eines neuen Features im Controller aus, der die Ansicht erneut zurückgibt, die dann das neue Feature enthält. Beim Löschen eines Features wird der Nutzer durch das Modal noch einmal gefragt, ob er das Feature wirklich löschen möchte. Nach Bestätigung wird hier ebenfalls eine entsprechende Funktion im Controller ausgelöst und erneut die Ansicht zurückgegeben.

Über den Projektlisten befindet sich noch ein Suchfeld, welches bei jedem Eintippen eines Buchstaben eine JavaScript-Funktion ausführt, die entsprechend passende Features anzeigt. Hierfür wird aus der Eingabe ein *Regex* erzeugt, mit dem man Zeichenketten nach Übereinstimmung überprüfen kann. In diesem Fall muss lediglich überprüft werden welche Features eine Übereinstimmung mit der Eingabe im Inputfeld haben.

```
$(".work-item").each((i, elem) => {
    if (!regex.test($(elem).find(".work-item-title").text())) {
        $(elem).hide();
    } else {
        $(elem).show();
        filterResults.push($(elem).attr("id"));
    }
});
```

Quellcode 14: Überprüfung eines Textes mit einem Regex

Die IDs der mit dem *Regex* übereinstimmenden Elementen werden in dem Array *filterResults* gespeichert, um nachfolgend die Eltern- und Kinderelemente des Features ebenfalls anzeigen zu können. Hierfür haben die Kinderelemente ein Attribut namens *data-parentid.*

Das Tippen auf eines der Features führt zur Ansicht für das Eintragen der Stunden für das gewählte Feature. Hierfür wird eine Teilansicht verwendet, welche eine einzeilige Wochentabelle mit Input-Elementen im Inneren erzeugt und schon eingetragene Stunden in die input-Elemente einträgt. Die Daten hierfür stammen aus dem *EffortForWorkItemModel*, welches im *ProjectTrackingController* befüllt wird.

Das Speichern von Stunden erfolgt über das Eintragen von Werten in die Input-Felder bzw. über das Ändern der schon eingetragenen Werte.

Der Bereich für das Eintragen von Abwesenheiten besteht ausschließlich aus drei Hauptansichten für jeweils die Listenübersicht, Detailansicht und die Ansicht zum Eintragen einer neuen Abwesenheit. Die Abwesenheitsübersicht beinhaltet eine Liste, welche die Abwesenheiten des *AbsenceOverviewModel* auflistet. Der Inhalt der Listenelemente besteht jeweils aus einem Link, der zur entsprechenden Detailansicht führt. Ein Listenelement beinhaltet zusätzlich den Abwesenheitstyp und den Zeitraum. Während der Implementierung ist außerdem aufgefallen, dass es auch hilfreich ist, in der Übersicht bereits den Status der Abwesenheit anzuzeigen. Dieser wurde somit dem Listenelement hinzugefügt und wird am rechten Rand in Höhe des Abwesenheitstyp angezeigt.

```
foreach (var absence in Model.Absences)
{
    <li class="list-group-item" id="@absence.Id"
        data-type="@Model.AbsenceType.FirstOrDefault(at =>
            at.Id == absence.AbsenceTypeId)?.Name">
        <a class="list-group-item-action" asp-controller="Absences"
           asp-action="GetAbsenceDetail" asp-route-absenceId="@absence.Id">
            <p class="mb-0 font-weight-bold">
                @Model.AbsenceType.FirstOrDefault(at =>
                    at.Id == absence.AbsenceTypeId)?.Name
                <span class="float-right font-weight-normal">
                    @absence.AbsenceRequestStates.OrderByDescending(ars =>
                        ars.Timestamp).FirstOrDefault()?.State
                </span>
            </p>
            <p class="mb-0">
                @absence.FirstDay.ToShortDateString() -
                @absence.LastDay.ToShortDateString()
            </p>
        </a>
    </li>
}
```

Quellcode 15: Auflistung der Abwesenheiten

Über der Abwesenheitsliste befindet sich auch hier wieder der Kontrollbereich. Hier befindet sich zum einen der Plus-Button, der zur Ansicht des Abwesenheiten eintragens führt und zusätzlich, abweichend zum Konzept, ein Drop-Down-Menü, mit dem man nach den verschiedenen Abwesenheitstypen filtern kann. Dieser Zusatz kommt hinzu, um eine noch bessere Übersicht seiner Abwesenheiten zu erlangen. Das entsprechende *select*-Element löst beim Ändern eine JavaScript-Funktion aus, welche die entsprechenden Listenelemente anzeigt bzw. ausblen-

det. Hierfür wird das Attribut *data-type* der Listenelemente mit der Auswahl des *select*-Elementes verglichen.

```
function filterAbsences(currentSelection) {
    if (currentSelection === "all") {
        $('#AbsencesList .list-group-item').show();
    } else {
        $('#AbsencesList .list-group-item').hide();
        $('#AbsencesList .list-group-item').each((i, elem) => {
            if ($(elem).data('type') === currentSelection)
                $(elem).show();
        });
    }
}
```

Quellcode 16: Funktion zum Filtern der Abwesenheiten

Die Detail-Ansicht ist lediglich eine Auflistung aller Informationen der gewählten Abwesenheit aus dem *AbsenceDetailModel*. Für diese Ansicht wird das Grid-Layout von Bootstrap verwendet, um zwei Spalten zu erhalten. Hierbei nimmt der Titel der Information jeweils fünf Spalten und die Information sieben Spalten ein und füllen somit die 12 Spalten des Grid-Layouts aus.

```
<div class="container-fluid pt-1">
    <div class="row mb-2">
        <div class="col-5 pr-0">Absence type: </div>
        <div class="col-7 font-weight-bold pl-0">@absenceType</div>
    </div>
    <div class="row mb-2">
        <div class="col-5 pr-0">From:</div>
        <div class="col-7 font-weight-bold pl-0">
            @Model.Absence.FirstDay.ToShortDateString()
        </div>
    </div>
    <div class="row mb-2">...</div>
    @if (Model.Absence.LastDayQuota > 0)
    {
        <div class="row mb-2">...</div>
        <div class="row mb-2">...</div>
    }
    else
    {
        <div class="row mb-2">...</div>
    }
    <div class="row mb-2">...</div>
    <div class="row mb-2">...</div>
</div>
```

Quellcode 17: Grid-Layout der Detail-Ansicht einer Abwesenheit

Die Ansicht zum Eintragen einer Abwesenheit hat das gleiche Grid-Layout wie die Detail-Ansicht. Die linke Spalte enthält dabei die entsprechenden *input*-Elemente, damit der Nutzer die Eingaben tätigen kann. Beim Speichern der Abwesenheit werden die Daten an den *AbsenceController gesendet*, der dann die Daten an die entsprechende Service-Funktion sendet. Der Controller gibt im Anschluss die Übersichtsansicht zurück, die dann auch die neue Abwesenheit enthält.

Bei den Abwesenheitsanfragen gibt es ebenfalls jeweils die Hauptansichten für die Übersicht und die Details. Diese sind sehr ähnlich aufgebaut wie die Übersicht- und Detailseite, der eingetragenen Abwesenheiten. Die Listenelemente bei den Abwesenheitsanfragen haben zusätzlich in der Übersicht den Namen des Anfragestellers. Außerdem werden, abweichend zum Konzept bereits für jedes Listenelement Buttons eingefügt, um ein schnelles Genehmigen bzw. Ablehnen einer Abwesenheit zu gewährleisten.

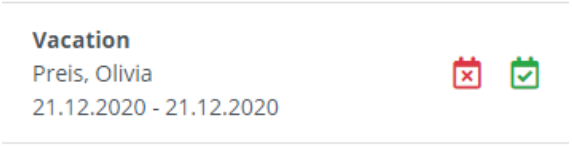

Abbildung 27: Listenelement der Abwesenheitsanfragen mit hinzugefügten Buttons

Die Detailseite der Abwesenheitsanfrage besitzt allerdings, wie im Konzept vorgesehen, ebenfalls diese Buttons. Mit Hilfe des FullCalendar-Frameworks wird obendrein ein Kalender eingefügt, der ein Hintergrundevent für die Tage des Abwesenheitszeitraumes besitzt. Dadurch wird der Hintergrund der entsprechenden Tage farblich markiert.

```
const initDate = new Date('@Model.Absence.FirstDay.ToString(
                                format: "yyyy-MM-ddTHH:mm:ss")');
const endDate = new Date('@Model.Absence.LastDay.ToString(
                                format: "yyyy-MM-ddTHH:mm:ss")');
AbsenceCalendar = new window.FullCalendar.Calendar(absenceCalendarElem,
    {
        initialView: 'dayGridMonth',
        initialDate: initDate,
        headerToolbar: {
            left: '',
            center: 'prev title next',
            right: ''
        },
        locale: 'en',
        weekNumbers: true,
        firstDay: 1,
        contentHeight: 180,
        events: [
            {
                start: initDate,
                end: endDate,
                overlap: false,
                display: 'background',
                color: '#0084bc',
                allDay:true
            }
        ]
    });
AbsenceCalendar.render();
```

Quellcode 18: Konfigurierung des Abwesenheitskalenders

Zum Ablehnen von Abwesenheiten wurde ein Bootstrap-Modal erstellt, welches sich in der Teilansicht *_RejectAbsenceModal* befindet. Dieses Modal wird geladen und angezeigt, wenn der Nutzer in der Übersicht oder in der Detailansicht auf Ablehnen tippt. Das Modal beinhaltet ein *textarea*-Element zur optionalen Eingabe eines Kommentars. Tippt der User im Modal auf den Reject-Button, wird im *ApprovalAbsencesController*, die entsprechende Funktion zum Ablehnen der Abwesenheit ausgeführt und die Übersichtsansicht wird zurückgegeben.

Der *ApprovalAbsencesController* führt des Weiteren Interaktionen nur aus, wenn der Nutzer hierfür entsprechende Rechte besitzt. Hierfür wird folgendes Attribut vor die Funktionen gesetzt:

```
[PermissionAuthorize(ClaimType.Role, value: "MANAGER, HUMAN_RESOURCE, DIRECTOR")]
```

Quellcode 19: Rechte-Attribut für Funktionen des ApprovalAbsencesController

9.3 UI-Komponente für Chrome zur Installation

Um eine UI-Komponente zu entwickeln, die zur Installation der PWA auffordert, wird ein Bootstrap-Modal und zusätzlich JavaScript-Code verwendet. Dies wird der *Index.cshtml*, also der Startseite, hinzugefügt. Das *AddPwaModal* beinhaltet lediglich Text und einen *Cancel*- sowie *Add*-Button.

Zunächst wird nach erfolgreichem Laden der Startseite das Cookie *appInstall* ausgelesen, sofern es vorhanden ist.

```
if (document.cookie) {
    const cookie = document.cookie;
    const name = cookie.substring(0, cookie.indexOf('='));
    let value;

    if (cookie.indexOf(';') !== -1)
    { value = cookie.substring(cookie.indexOf('=') + 1, cookie.indexOf(';'));}
    else { value = cookie.substr(cookie.indexOf('=') + 1, cookie.length); }

    if (name === "appInstall")
        appInstall = value;
}
```

Quellcode 20: Auslesen des Cookies appInstall

Darauffolgend wird überprüft, ob die App im *standalone*-Modus geöffnet ist, was bedeuten würde, dass sie als PWA geöffnet ist. Ist dies der Fall wird das Cookie *appInstall* entsprechend auf *true* gesetzt. Mit Hilfe des Events *beforeinstallprompt*, welches ausgelöst wird, bevor die PWA installiert wird, kann in den Installationsprozess eingegriffen werden. Das Eventobjekt des Events wird in einer Variablen gespeichert, um es später verwenden zu können. Dann wird das *AddPwaModal* angezeigt, solange *appInstall* nicht *true* ist.

```
window.addEventListener('beforeinstallprompt', (e) => {
    e.preventDefault();
    // Stash the event so it can be triggered later.
    deferredPrompt = e;

    if (!appInstall)
        $("#AddPwaModal").modal("show");
});
```

Quellcode 21: Listener für beforeinstallpromt-Event

Beim Tippen auf den *AddPwaButton* wird das Modal wieder ausgeblendet und der Installationsdialog des Browsers wird geöffnet. Hierfür wird die *prompt*-Methode des zuvor gespeicherten Events verwendet. Über die *userChoice*-Methode kann die Wahl, die der Nutzer trifft, abgefangen werden und bei Akzeptieren der Installa-

tion kann somit das Cookie *appInstall* auf *true* gesetzt werden, wodurch das Modal nicht mehr geöffnet wird.

```
$("#AddPwaButton").click(() => {
    $("#AddPwaModal").modal("hide");
    // Show the install prompt
    deferredPrompt.prompt();
    // Wait for the user to respond to the prompt
    deferredPrompt.userChoice.then((choiceResult) => {
        if (choiceResult.outcome === 'accepted') {
            appInstall = true;
            document.cookie = "appInstall=true; expires="
                            + expireDate.toGMTString() + ";";
        }
    });
});
```

Quellcode 22: Click-Event für AddPwaButton

Zusätzlich wird mit Hilfe des Events *appinstalled*, welches beim Installieren der PWA ausgelöst wird, ebenfalls das Cookie auf *true* gesetzt.

9.4 Anpassungen für PWA und Offline Support

Um aus der reinen Webapplikation eine PWA zu erzeugen, wird zunächst der *ConfigureServices*-Methode der *Startup.cs* die Funktion *AddProgressiveWebApp* hinzugefügt. Hier kommen die Angaben in der *appsettings.json* zum Einsatz. Zwischengespeichert werden beim Start der PWA lediglich die Startseite sowie die *fallback.html*, die angezeigt wird, wenn beim Laden einer Seite keine Internetverbindung besteht. Als Strategie für den Service Worker wird *networkFirst* gewählt, somit wird immer erst versucht die Daten vom Server zu laden, da der Nutzer immer aktuelle Daten zu seinen Stunden und Abwesenheit erwartet.

```
"pwa": {
  "cacheId": "Worker 1.1",
  "strategy": "networkFirst",
  "routesToPreCache": "/",
  "offlineRoute": "fallback.html",
  "registerServiceWorker": true,
  "registerWebmanifest": false
},
```

Quellcode 23: PWA-Einstellungen in appsettings.json

Wie in der appsettings.json zu sehen ist, wird zwar der Service Worker, aber nicht die Manifest hinzugefügt. Diese wird manuell im *head* der Layout-Datei eingebunden.

```
@* Add Manifest file *@
<link rel="manifest" href="~/manifest.json"/>
```

Quellcode 24: Hinzufügen der Manifest-Datei in _Layout.cshtml

Der Grund für das manuelle Hinzufügen der Manifest-Datei ist ein Problem der nicht erlaubten Weiterleitung von der Manifest-URL der App auf die URL des Active Directory. Cross-Origin Resource Sharing (CORS) nennt sich der Mechanismus, der den Zugriff auf eine andere Domain erlaubt.

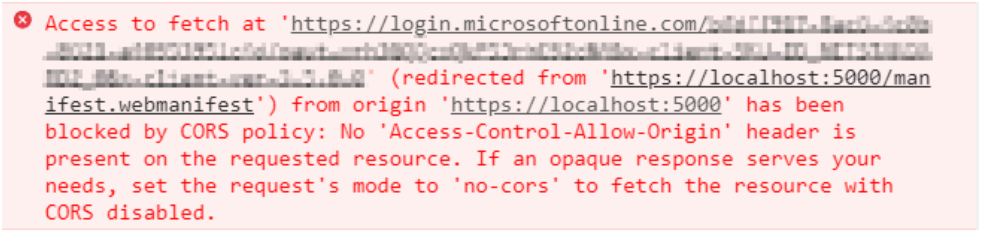

Abbildung 28: Konsolen-Fehler bei Zugriff auf manifest.webmanifest

Dieser Fehler konnte durch Änderungen in der App selbst, wie auch in der Konfigurierung der Authentifizierung mit dem Active Directory nicht gelöst werden. Um dem Fehler aus dem Weg zu gehen, wird die *manifest.json* direkt in die *_Layout.cshtml* eingebunden.

In der Manifest-Datei werden die wichtigen Daten für die Installation der PWA festgehalten. Hierfür wurden zwei unterschiedlich große Icons für verschiedene Browser bzw. Geräte hinzugefügt. Obendrein wird der Display-Modus *standalone* festgelegt, wodurch der Eindruck einer nativen App entsteht. Die Farben für den Ladebildschirm der App sowie die Themenfarbe sind angepasst an die Corporate Identity des Unternehmens.

```
{
  "name": "SYPRA Mobile App",
  "short_name": "SYPRA Mobile",
  "orientation": "portrait",
  "description": "PWA for SYPRA App",
  "icons": [
    {
      "src": "img/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "img/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "/",
  "display": "standalone",
  "color": "#ffffff",
  "background_color": "#0084bc",
  "theme_color": "#0084bc"
}
```

Quellcode 25: Manifest-Datei

9.5 Tests und Änderungen

Um die Kompatibilität als PWA zu testen, wird die App mit Hilfe des Lighthouse-Tools von Chrome analysiert. Mit Lighthouse kann ein Bericht erzeugt werden, um festzustellen in welchem Maße die Seite als PWA funktioniert. Dieses Tool lässt sich über die Entwicklertools von Chrome erreichen. [45]

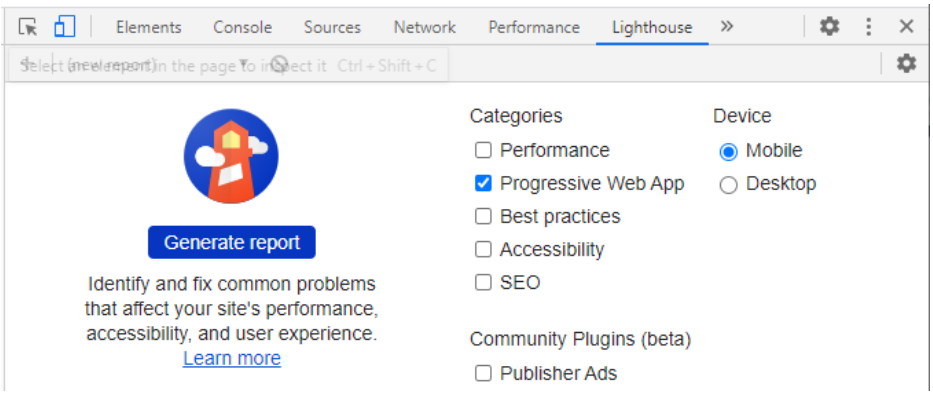

Abbildung 29: Lighthouse-Tool von Chrome

Durch ein Klicken auf den Button „Generate report“ wird der Bericht innerhalb kurzer Zeit erzeugt.

Durch die bereits vorgestellten Anpassungen für eine PWA erlangt der Bericht von Lighthouse einen sehr guten Stand. In den Bereichen *Fast and reliable* und *Installable* werden alle Kriterien erreicht.

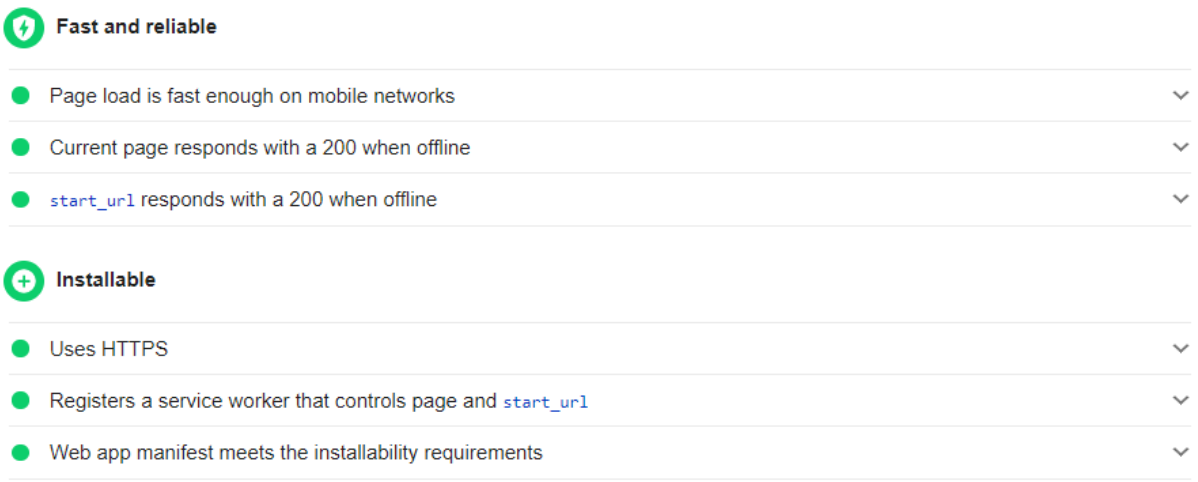

Abbildung 30: Kriterien Fast and reliable und Installable des Lighthouse Berichts

Im Bereich *PWA Optimized* wird lediglich ein Kriterium nicht erreicht. Dies betrifft die Icons der App. Es wird bemängelt, dass kein *maskable Icon* vorhanden ist. Dies ist ein neues Iconformat, welches ermöglicht, dass das Icon auf jedem Android-Gerät gut aussieht. Icons, die diesem Format nicht entsprechen, bekommen einen weißen Hintergrund. *Maskable icons* hingegen füllen den kompletten auf dem Gerät vorgesehenen Bereich aus. [46]

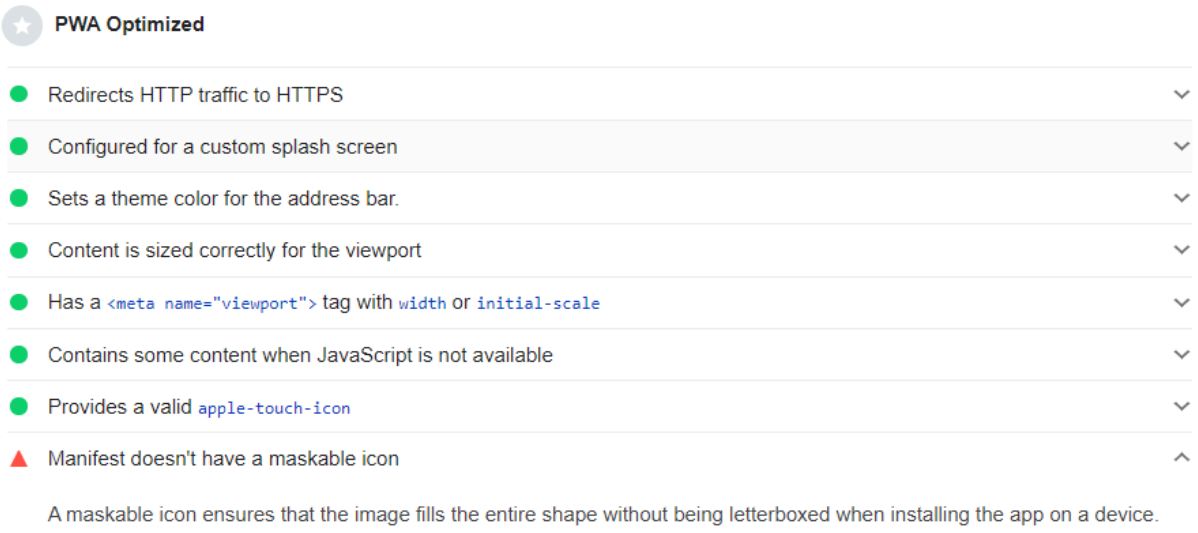

Abbildung 31: PWA Optimized Kriterium des Lighthouse Berichts

Während des Ladens der PWA auf einem Android-Smartphone wird eine Ladebildschirm gezeigt, der unter anderem ein Icon enthält.

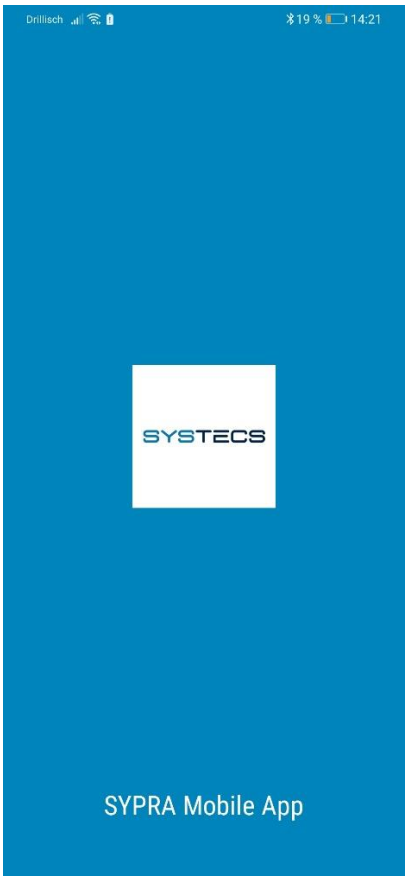

Abbildung 32: Screenshot des Ladebildschirms der PWA auf Android

Bei einem entsprechenden *maskable icon* würde das Logo der Firma keinen weißen Hintergrund bekommen. In diesem Fall wird das Kriterium vernachlässigt, da so das Logo der Firma ohnehin besser zu erkennen ist. Daher werden hierfür keine Änderungen vorgenommen.

Zusätzlich zum Testdurchlauf mit Lighthouse, wird die App auf einem Android-Smartphone und einem iOS-Gerät getestet. Dies führt zum einen zu einer Anpassung einiger Schriftgrößen, damit Links mit dem Finger besser anwählbar sind. Zusätzlich ist beim Testen aufgefallen, dass das Tippen auf den Bildschirm oft zum Markieren des Textes führt. Dies wurde mit Hilfe von CSS deaktiviert.

Nach der Vorstellung der entstandenen PWA beim Unternehmen, mussten lediglich kleine Verbesserungen vorgenommen werden wie der automatische Fokus auf den aktuellen Tag beim Eintragen der Stunden oder der Vorauswahl von „Vacation“ beim Eintragen von Abwesenheiten.

Weitere Anpassungen mussten nicht vorgenommen werden.

10. Vorstellung und Bewertung der entstandenen PWA zur Arbeitszeitverwaltung

In diesem Kapitel wird die erstellte PWA vorgestellt und zugleich mit den Anforderungen sowie dem Konzept verglichen und somit bewertet. Hierfür werden Screenshots der mobilen App eingebunden.

Zunächst werden die Startseite und das Grundlayout vorgestellt. Wie im Konzept vorgesehen enthält die App eine Kopfzeile, in der der Name der geöffneten Seite und ein Menü enthalten ist. Die Startseite enthält die vorgesehenen vier Buttons für das Erreichen der verschiedenen Seiten und zum Ausloggen des Nutzers. Diese Funktionalitäten werden auch über das obige Menü erreicht.

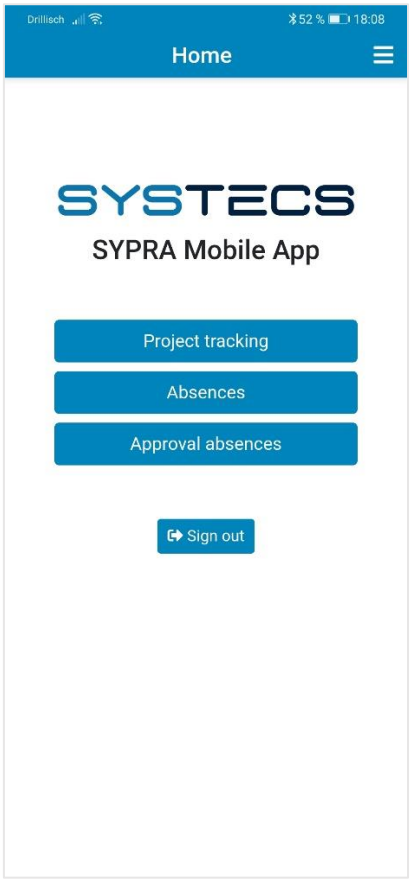

Abbildung 33: Startseite der entstandenen PWA

Nach dem Öffnen des Projekttrackings, wird die Stundenübersicht erreicht. Diese enthält eine Tages-, Wochen- und Monatsübersicht der eingetragenen Stunden, zwischen denen der Nutzer wechseln kann. Die verschiedenen Übersichten liefern dabei die geforderten Informationen. Außerdem gelangt man durch Tippen auf einen Tag der Monatsansicht oder auf einen Eintrag in der Wochenansicht zur Tagesübersicht. Somit sind die Anforderungen für die Stundenansicht vollständig abgedeckt.

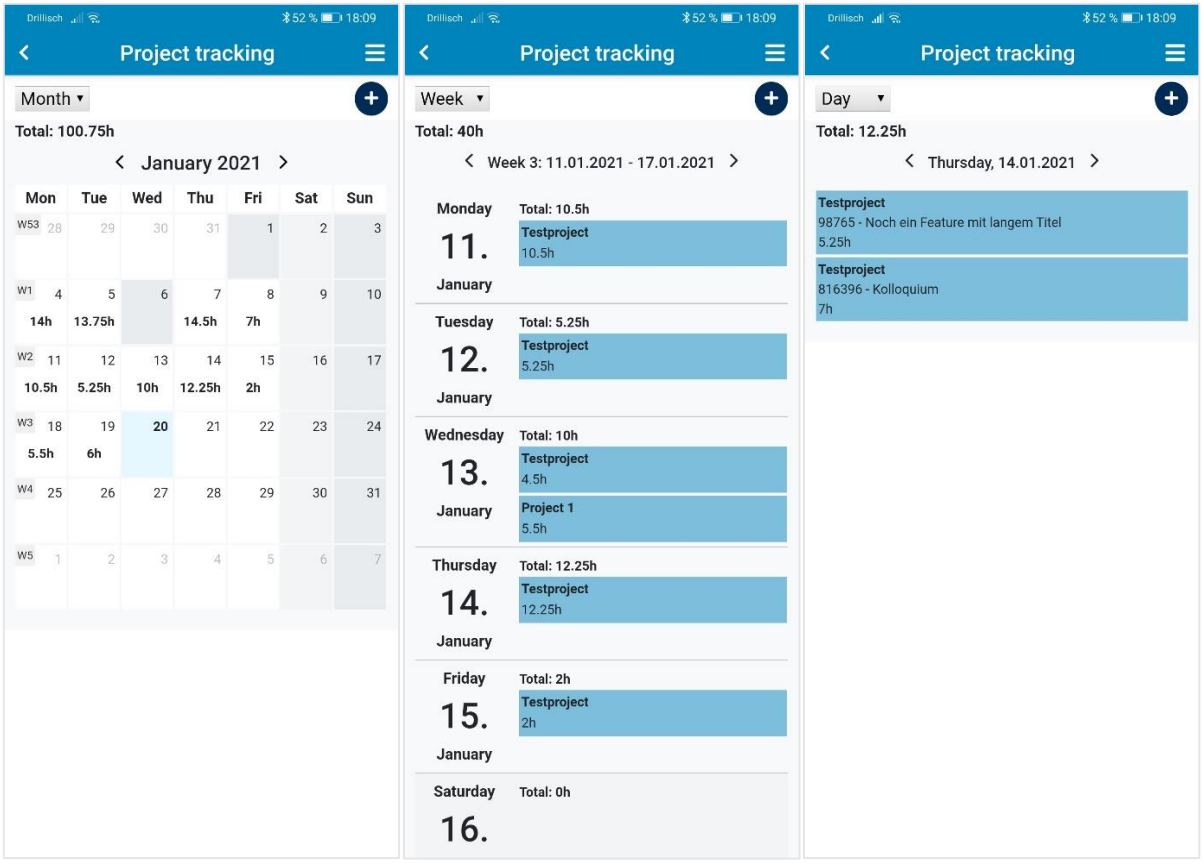

Abbildung 34: Stundenübersicht der entstandenen PWA

Über den Plus-Button gelangt der Nutzer zum Eintragen der Stunden. Dabei wird wie vorgesehen zunächst ein Feature ausgewählt, auf das dann die Stunden eingetragen werden. Laut Anforderung soll dem Nutzer ermöglicht werden, seine Stunden für mehrere Tage einzutragen. Dies wird durch die Anzeige einer ganzen Woche und dem Wechsel zwischen den Wochen ermöglicht. Auch werden vorhandene Stundeneinträge in dieser Wochendarstellung angezeigt. Das Stundeneintragen der realisierten App erfüllt folglich alle diesbezüglichen Anforderungen.

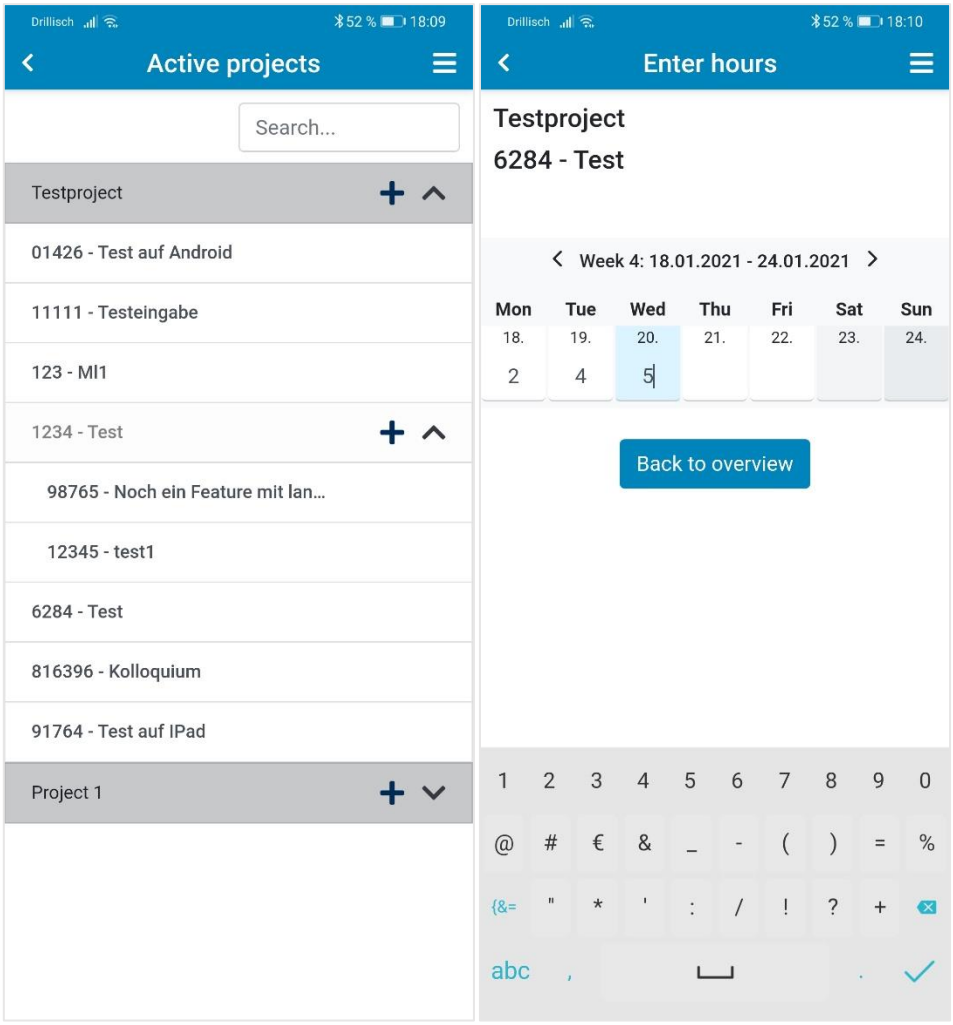

Abbildung 35: Stundeneintragen der entstandenen PWA

Im Bereich der aktiven Projekte, bei dem ein Feature ausgewählt wird, kann außerdem nach einem bestimmten Feature gesucht, ein neues hinzugefügt und wenn möglich gelöscht werden. Somit erfüllt auch dieser Bereich alle Anforderungen.

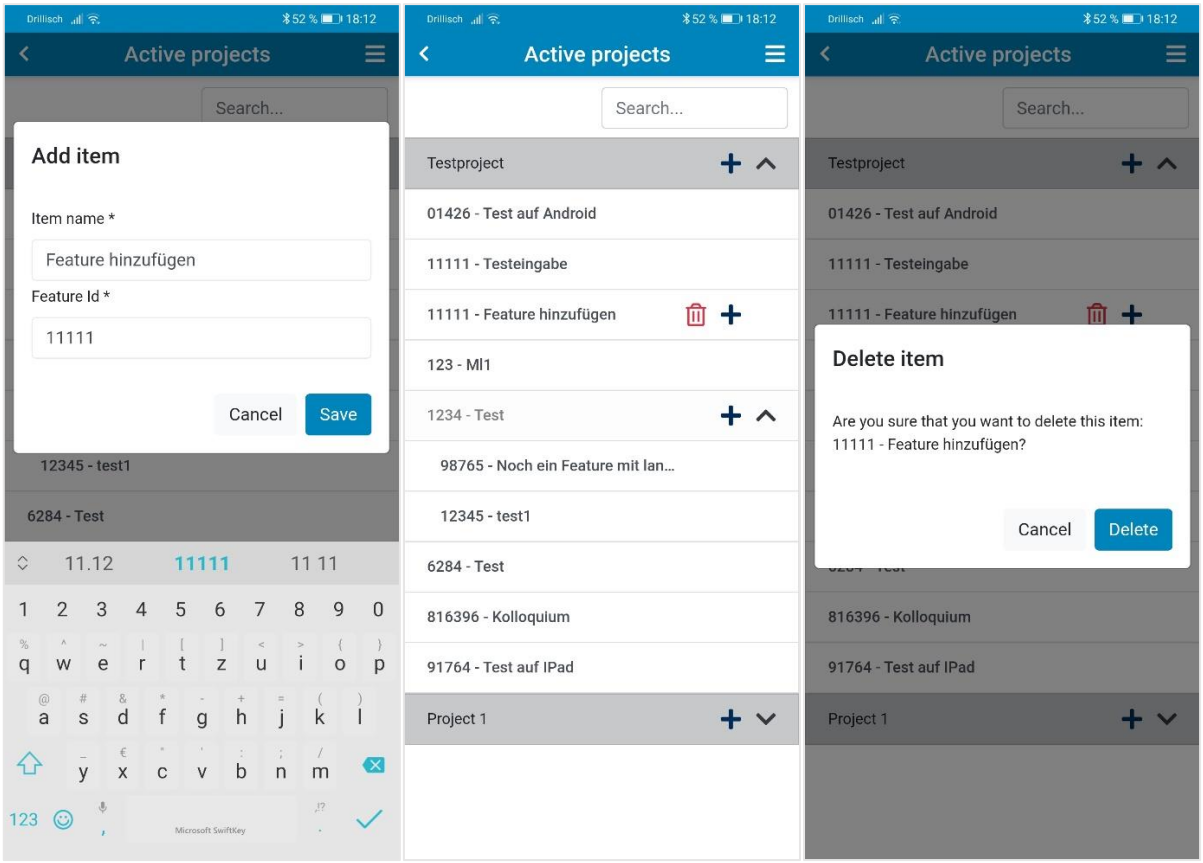

Abbildung 36: Feature in der entstandenen PWA

Bis hierhin lässt sich schon festhalten, dass die PWA bereits alle Anforderungen der Priorität 1 erreicht hat.

Die Abwesenheitsübersicht ist, wie in den Anforderungen und im Konzept vorgesehen, durch eine Liste realisiert. Das Tippen auf eine der Abwesenheiten führt zu der Detailansicht, über die eine Abwesenheit zurückgezogen werden kann.

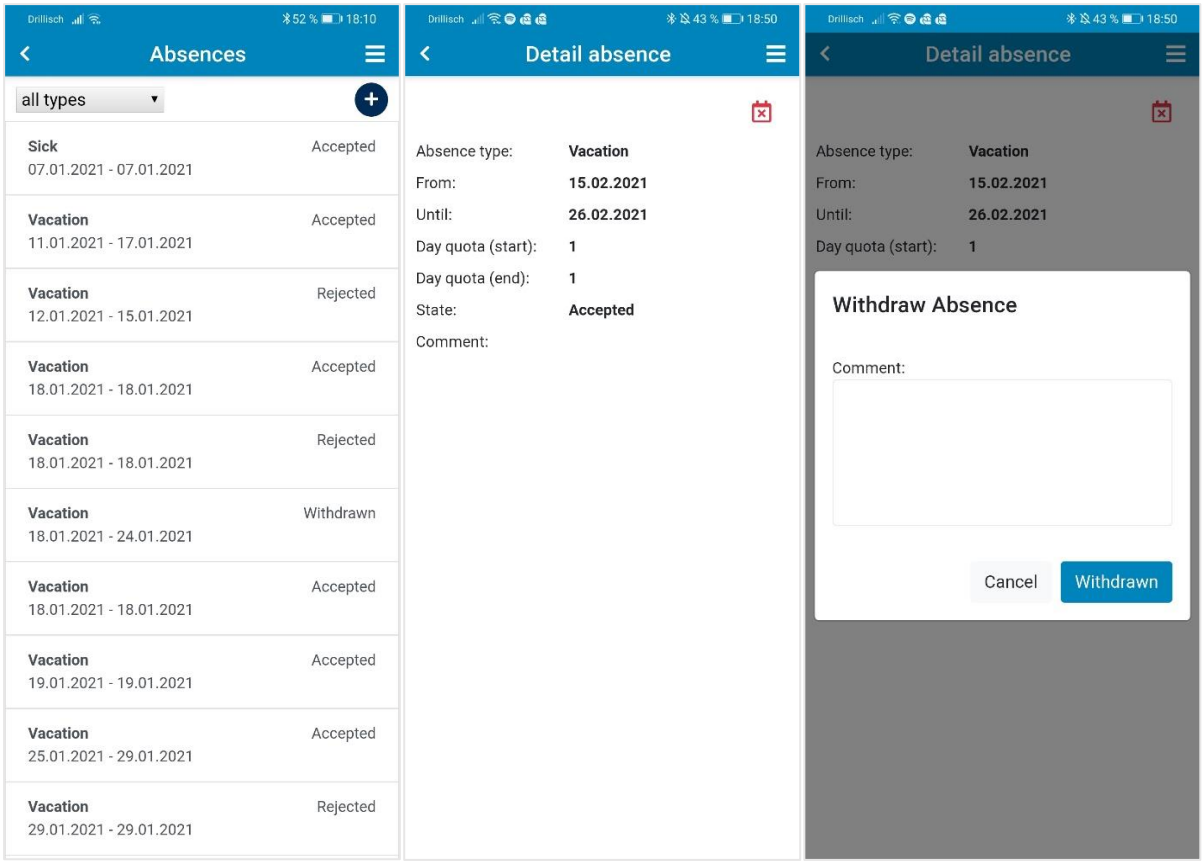

Abbildung 37: Abwesenheitsübersicht und Detailansicht der entstandenen PWA

Das Eintragen der Abwesenheiten, wird wie festgesetzt über den Plus-Button der Übersicht erreicht und die vorgesehenen Daten werden vom Nutzer angegeben. Beim Absenden der Daten wird nach aktuellem Stand allerdings nicht nach Übereinstimmung mit anderen Abwesenheiten geprüft, wodurch die Anforderungen in diesem Bereich nicht vollends erreicht sind. Diese Ergänzung erfordert eine Erweiterung im Service und schließlich in der App selbst. Hierfür hat die begrenzte Zeit nicht mehr ausgereicht, lässt sich aber zu einem späteren Zeitpunkt ohne viel Aufwand ergänzen.

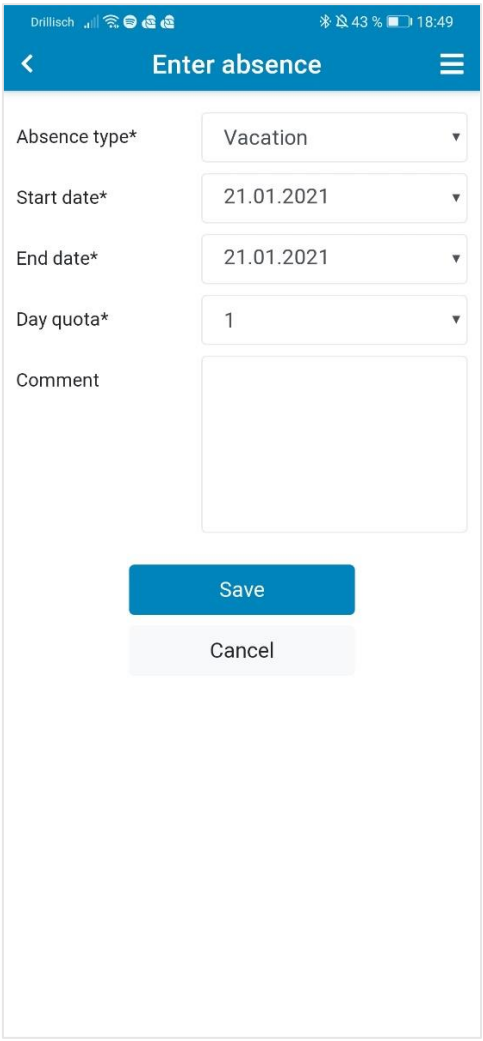

Abbildung 38: Eintragen einer Abwesenheit in der entstandenen PWA

Das Genehmigen bzw. Ablehnen von Abwesenheiten erfüllt durch die Übersicht und Detailseite die Anforderungen. Zusätzlich zum ursprünglichen Konzept wurden die Buttons zum Genehmigen und Ablehnen in der Listenansicht aufgenommen, um noch schneller eine Anfrage bearbeiten zu können. Die geforderten Push-Benachrichtigungen, die durch die PWA ohnehin nur für Android realisierbar sind, wurden aufgrund der begrenzten Zeit nicht umgesetzt. Diese Ergänzung würde nochmals etwas Recherche- und Entwicklungsaufwand mit sich bringen.

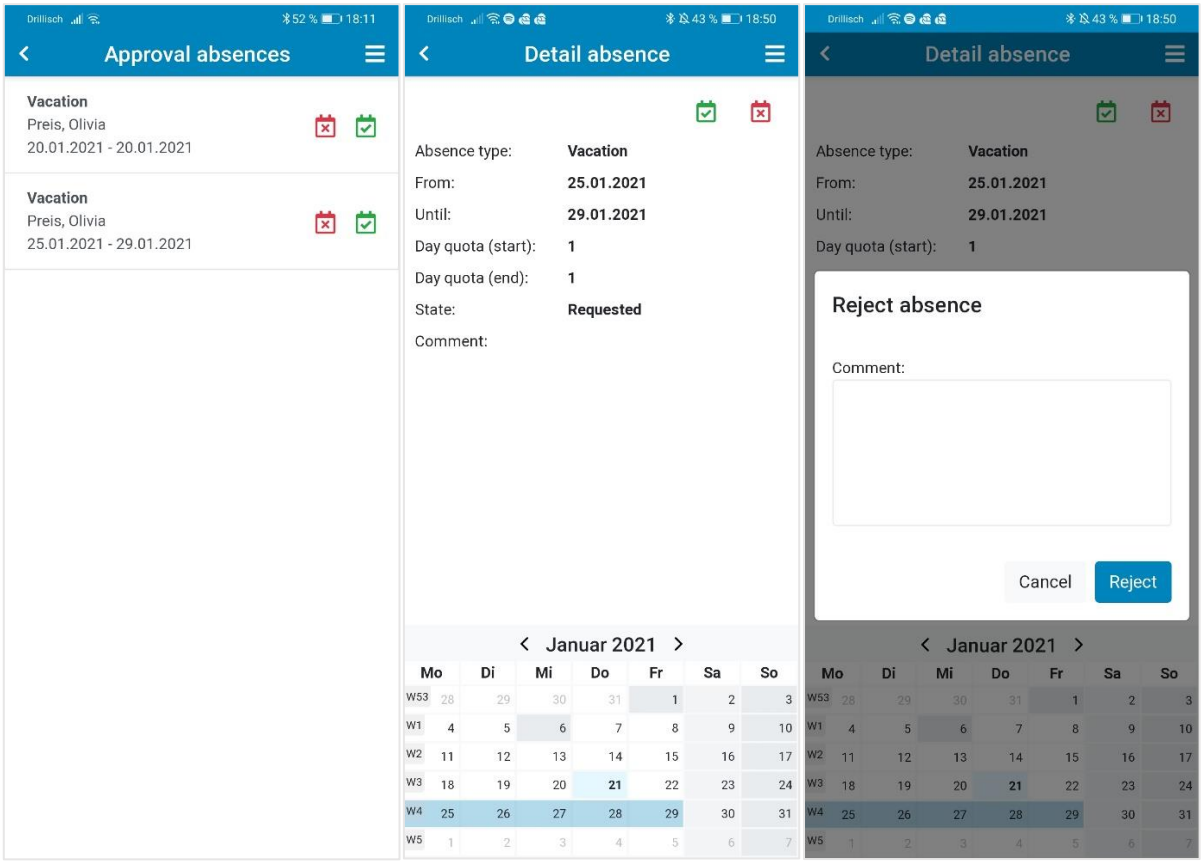

Abbildung 39: Abwesenheitsanfragen der entstandenen PWA

Der geforderte Dialog zur Installations-Aufforderung konnte wie im Konzept umgesetzt werden. Durch das Speichern eines Cookies nach erfolgter Installation wird gewährleistet, dass der Nutzer in der installierten App nicht erneut zur Installation aufgefordert wird.

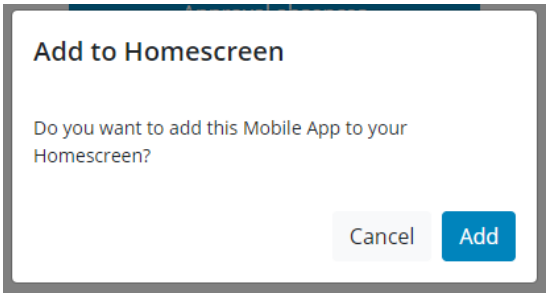

Abbildung 40: Installationsaufforderung der entstandenen PWA

Die App kann bei den Tests innerhalb von fünf Sekunden gestartet werden und Benutzerinteraktionen werden auch immer innerhalb von fünf Sekunden bearbeitet. Die PWA besitzt also eine für die Anforderungen der App ausreichende Performance.

Die Kompatibilität für Android- wie auch iOS-Geräten ist durch die Webbasis gegeben. Außerdem ist die PWA kompatibel mit den Browsern Google Chrome,

Samsung Internet und Safari und deckt hier somit auch die erweiterten Anforderungen aufgrund der Wahl des Entwicklungsansatzes ab.

Die Benutzerführung erfolgt wie gefordert ausschließlich in Englisch und das UI ist für mobile Geräte optimiert. Außerdem wurde bei Erstellung des Konzeptes wie auch bei der Realisierung auf die Corporate Identity des Unternehmens geachtet, was vor allem das Logo, die Farben und Schriftart betrifft. Zusätzlich wurde im Konzept beachtet viele Komponenten ähnlich aufzubauen, wie die des Systems auf das die mobile App aufbaut. Durch die Treue zum Konzept konnte der Wiedererkennungseffekt zum vorhandenen System beibehalten werden.

Die Implementierung wurde wie festgesetzt mit Hilfe des ASP.NET Core Frameworks durchgeführt. Zusätzlich durch die Wahl der PWA wurden die Webtechnologien HTML, CSS und JavaScript verwendet. Hinzu kamen, ebenfalls wie vorgesehen. die Bibliotheken Bootstrap und Font Awesome zum Einsatz. Für die einfachere Erstellung der Styleregeln wurde außerdem planmäßig *SASS* eingesetzt. Obendrein wurde der Code zur Verständlichkeit kommentiert.

Für die Authentifizierung der Nutzer wurde der Dienst *Azure Active Directory Login* von *Microsoft* mit der Domäne *systecs.com* eingesetzt. Zusätzlich wurde eine sichere Datenübertragung durch die SSL-Verschlüsselung gewährleistet. Im Bereich Sicherheit und Authentifizierung werden so alle Anforderungen erreicht.

Die PWA lässt sich zum Startbildschirm hinzufügen und wird von dort im *standalone*-Modus geöffnet. Die Verbreitung der App ist durch die URL außerdem sehr einfach, wodurch die Anforderungen bezüglich der Verbreitung erfüllt werden. Zusätzlich kam durch die Wahl der PWA hinzu, dass die App über *Microsoft Azure* gehostet werden soll. Dies hat das Unternehmen selbst übernommen.

Die Offlinefunktionalitäten der PWA sind beschränkt. Nach erfolgreichem zwischenspeichern der Startseite, kann diese ohne bestehende Internetverbindung geöffnet werden. Besteht allerdings keine Internetverbindung und die angefragte Seite ist nicht im Cache vorhanden, wird eine eigens erstellte Offline-Seite angezeigt. Für die Cache-Strategie wurde *networkFirst* gewählt. Die Daten werden also vorrangig immer aus dem Netzwerk geladen, um einen aktuellen Stand der Daten zu gewährleisten. Weitere Offlinefunktionen sind nur beschränkt vorhanden. Wenn der Nutzer beispielsweise seine Abwesenheiten des aktuellen Jahres schon einmal angefragt hat und dies im Cache noch vorhanden ist, wenn der Nutzer offline ist, dann können diese trotzdem angezeigt werden. Das Eintragen neuer Abwesenheiten wäre aber nicht möglich. Die PWA lässt sich also nur mit bestehender Internetverbindung im vollen Umfang nutzen. Im Anschluss dieser Arbeit wäre es möglich sich mit dem Thema Offlinefunktionalitäten der PWA nochmal genauer zu beschäftigen. Es wäre zum Beispiel denkbar, Daten nach dem Absenden zwischenzuspeichern und später bei bestehender Internetverbindung abzuschicken.

Durch die Wahl der gleichen Technologien wie der des vorhandenen Systems, besteht für das Unternehmen bei der Wartung der App ein möglichst geringer Aufwand. Zusätzlich wurde bei der Entwicklung darauf geachtet, dass die App einfach ergänzt werden kann.

11. Fazit

In diesem letzten Kapitel wird abschließend die ganze Arbeit zusammengefasst und ein Fazit gezogen.

Die Arbeit handelt von der Erstellung einer mobilen App zur Arbeitszeitverwaltung für die Firma SYSTECS Informationssysteme GmbH. Näher geht es darum Stunden und Abwesenheiten einzutragen sowie Urlaubsanfragen zu genehmigen bzw. abzulehnen.

Die Anforderungen sind entstanden durch die Analyse bereits existierender Apps sowie des Ist-Zustandes bei SYSTECS und durch Vorstellungen des Unternehmens selbst.

Als Entwicklungsansatz wurde für die neue App der PWA-Ansatz gewählt. Die Technologien sind somit webbasiert. Zusätzlich wird *ASP.NET Core*, *SASS* als *CSS*-Präprozessor, *Bootstrap*, *FullCalendar* und *Font Awesome* zur Entwicklung der PWA eingesetzt.

Bei der Realisierung wurden nach dem MVC-Entwurfsmuster einige Controller, Modelle und Haupt- sowie Teilansichten erstellt. Bei der Entwicklung wurde außerdem streng nach Anforderungen und Konzept verfahren.

Alle vorher ausgewählten Technologien kommen bei der Realisierung zum Einsatz und ergeben so die PWA. Diese wird auf einem iOS- sowie Android-Gerät getestet und dem Unternehmen vorgestellt, was einige kleine Änderungen des UI mit sich bringt und schließlich zur Fertigstellung der praktischen Arbeit führt.

In einigen Bereichen lässt sich die entstandene Anwendung nachträglich verbessern. Hierzu gehört zum einen die nicht erreichten Anforderungen: Überprüfung auf Übereinstimmung mit bestehenden Abwesenheiten und Push-Benachrichtigung bei neuen Abwesenheitsanfragen. Zusätzlich lässt sich festhalten, dass das UI ausschließlich für ein Smartphone in Hochformat ausgelegt ist. Innerhalb dieser Arbeit wurde lediglich darauf geachtet, dass sich die App auch auf größeren Bildschirmen oder anderen Formaten zumindest bedienen lässt. Es wäre allerdings möglich noch ein spezielleres Konzept für Smartphones im Querformat und für Tablets zu erstellen. Außerdem konnte die Anwendung im produktiven Einsatz, auf Grund der begrenzten Zeit, während der laufenden Arbeit nicht ausreichend getestet werden. Sicherlich würden dabei einige Fehler und Verbesserungsmöglichkeiten zum Vorschein kommen und müssten nachträglich außerhalb dieser Arbeit korrigiert werden. Es ist allerdings festzuhalten, dass die App mit ihrem aktuellen Stand einen sehr guten Entwicklungsstand erreicht hat und so verwendet werden kann.

Literatur

[1] H.-P. Schüler, „Job ab - Zeit läuft!: Neun Apps zur Erfassung von Arbeitszeiten“, *c't*, Nr. 17, S. 120–125, 2020.

[2] Gridvision Engineering GmbH, *Features.* [Online]. Verfügbar unter: https://www.gleeo.com/en/features (Zugriff am: 18. Januar 2021).

[3] CHIP Online, *Stempeluhr II - Android App.* [Online]. Verfügbar unter: https://www.chip.de/downloads/Stempeluhr-II-Android-App_170902045.html (Zugriff am: 18. Januar 2021).

[4] DynamicG Android Apps, *Zeiterfassung.* [Online]. Verfügbar unter: https://play.google.com/store/apps/details?id=com.dynamicg.timerecording (Zugriff am: 18. Januar 2021).

[5] T. Partl, *Timo Partl.* [Online]. Verfügbar unter: https://timopartl.com/ (Zugriff am: 18. Januar 2021).

[6] SYSTECS Software Engineering, *SYSTECS Software Engineering.* [Online]. Verfügbar unter: https://www.systecs.com/ (Zugriff am: 28. Oktober 2020).

[7] C. Ebert, *Systematisches Requirements Engineering: Anforderungen ermitteln, dokumentieren, analysieren und verwalten,* 6. Aufl. Heidelberg: dpunkt.verlag, 2019.

[8] F. Tenzer, *Anteile der verschiedenen Android-Versionen an der Internetnutzung von Geräten mit Android OS weltweit im August 2020.* [Online]. Verfügbar unter: https://de.statista.com/statistik/daten/studie/180113/umfrage/anteil-der-verschiedenen-android-versionen-auf-geraeten-mit-android-os/ (Zugriff am: 28. September 2020).

[9] Apple Developer, *App Store - Support.* [Online]. Verfügbar unter: https://developer.apple.com/support/app-store/ (Zugriff am: 1. Oktober 2020).

[10] Mandarina und S. Augusten, *Definition „Native Anwendung“: Was ist eine Native App?* [Online]. Verfügbar unter: https://www.dev-insider.de/was-ist-eine-native-app-a-827532/ (Zugriff am: 11. September 2020).

[11] D. Hermes und N. Mazloumi, *Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals*. Berkeley, CA: Apress; Imprint,Apress, 2019.

[12] Google Play, *trivago: Hotels vergleichen.* [Online]. Verfügbar unter: https://play.google.com/store/apps/details?id=com.trivago&gl=DE (Zugriff am: 24. Januar 2021).

[13] D. Sheppard, *Beginning Progressive Web App Development: Creating a Native App Experience on the Web,* 1. Aufl. Berkeley, CA: Apress, 2017.

[14] trivago, *trivago.* [Online]. Verfügbar unter: https://www.trivago.com/ (Zugriff am: 24. Januar 2021).

[15] 1&1 IONOS SE Digital Guide, *Hybrid-App: Was unterscheidet sie von anderen App-Formaten?* [Online]. Verfügbar unter: https://www.ionos.de/digitalguide/websites/web-entwicklung/hybrid-app-das-beste-aus-web-und-native-app/ (Zugriff am: 15. September 2020).

[16] C. Liebel, *Adobe stellt PhoneGap ein: Drama oder Chance?* [Online]. Verfügbar unter: https://www.heise.de/developer/artikel/Adobe-stellt-PhoneGap-ein-Drama-oder-Chance-4868133.html (Zugriff am: 15. September 2020).

[17] AppYourself GmbH, *PWAs unter iOS 12.2: Apple räumt nun auch Progressive Web Apps mehr Funktionen ein.* [Online]. Verfügbar unter: https://appyourself.net/de/blog/pwas-unter-ios-12.2/ (Zugriff am: 16. September 2020).

[18] Android Developers, *Debug your app.* [Online]. Verfügbar unter: https://developer.android.com/studio/debug (Zugriff am: 14. September 2020).

[19] M. Kratzenberg, *iOS-App programmieren - Voraussetzungen und Tipps.* [Online]. Verfügbar unter: https://www.giga.de/downloads/ios-11/tipps/ios-app-programmieren-voraussetzungen-und-tipps/ (Zugriff am: 21. September 2020).

[20] 1&1 IONOS SE Digital Guide, *HTTPS: Was es bedeutet und warum es wichtig ist.* [Online]. Verfügbar unter: https://www.ionos.de/digitalguide/hosting/hosting-technik/was-ist-https/ (Zugriff am: 16. September 2020).

[21] Jens, *Progressive Web Apps: Google zündet plötzlich den Turbo & Play Store wandelt sich wohl zum PWA Store.* [Online]. Verfügbar unter: https://www.googlewatchblog.de/2020/04/progressive-web-apps-google/ (Zugriff am: 16. September 2020).

[22] D. Petereit, *Webdesign: So steht es um Progressive-Web-Apps im Jahr 2020.* [Online]. Verfügbar unter: https://t3n.de/news/webdesign-steht-um-jahr-2020-1242754/2/ (Zugriff am: 24. September 2020).

[23] F. Tenzer, *Marktanteile der führenden mobilen Browser an der Internetnutzung mit Mobiltelefonen in Deutschland von Januar 2009 bis Juli 2020.* [Online]. Verfügbar unter: https://de.statista.com/statistik/daten/studie/184297/umfrage/marktanteile-mobiler-browser-bei-der-internetnutzung-in-deutschland-seit-2009/ (Zugriff am: 24. September 2020).

[24] Balsamiq Studios, *Balsamiq Cloud.* [Online]. Verfügbar unter: https://balsamiq.cloud/ (Zugriff am: 12. Januar 2021).

[25] J.-O. Kuhfuß, *Was ist eine Sitemap? Einfach erklärt.* [Online]. Verfügbar unter: https://praxistipps.chip.de/was-ist-eine-sitemap-einfach-erklaert_41300 (Zugriff am: 22. Oktober 2020).

[26] S. Augsten, *Was ist ein User Interface?* [Online]. Verfügbar unter: https://www.dev-insider.de/was-ist-ein-user-interface-a-735069/ (Zugriff am: 28. Oktober 2020).

[27] R. Schanze, *Was ist NET Framework? Einfach erklärt.* [Online]. Verfügbar unter: https://www.giga.de/downloads/windows-10/specials/was-ist-net-framework-einfach-erklaert/ (Zugriff am: 13. November 2020).

[28] P. Japikse, K. Grossnicklaus und B. Dewey, *Building Web Applications with .NET Core 2.1 and JavaScript: Leveraging Modern JavaScript Frameworks,* 2. Aufl. Berkeley, CA: Apress, 2020.

[29] S. Smith, *Übersicht über ASP.NET Core MVC.* [Online]. Verfügbar unter: https://docs.microsoft.com/de-de/aspnet/core/mvc/overview?view=aspnetcore-5.0 (Zugriff am: 16. November 2020).

[30] R. Anderson, T. Mullen und D. Vicales, *Razor-Syntax Referenz für ASP.net Core.* [Online]. Verfügbar unter: https://docs.microsoft.com/de-de/aspnet/core/mvc/views/razor?view=aspnetcore-5.0 (Zugriff am: 16. November 2020).

[31] A. Rick, T. Dykstra und S. Smith, *Anwendungsstart in ASP.NET Core.* [Online]. Verfügbar unter: https://docs.microsoft.com/de-de/aspnet/core/fundamentals/startup?view=aspnetcore-5.0 (Zugriff am: 18. November 2020).

[32] Dot Net Tutorials, *ASP.NET Core appsettings.json - Dot Net Tutorials.* [Online]. Verfügbar unter: https://dotnettutorials.net/lesson/asp-net-core-appsettings-json-file/ (Zugriff am: 30. November 2020).

[33] S. Smith und D. Brock, *Layout in ASP.NET Core.* [Online]. Verfügbar unter: https://docs.microsoft.com/de-de/aspnet/core/mvc/views/layout?view=aspnetcore-5.0 (Zugriff am: 30. November 2020).

[34] K. Nandwani, *Was ist NuGet, und welche Funktion hat es?* [Online]. Verfügbar unter: https://docs.microsoft.com/de-de/nuget/what-is-nuget (Zugriff am: 4. Januar 2021).

[35] K. Strube, *Turning a ASP.NET Core website into a Progressive Web App (PWA).* [Online]. Verfügbar unter: https://blog.elmah.io/turning-an-aspnet-core-website-into-a-progressive-web-app-pwa/ (Zugriff am: 17. November 2020).

[36] J. Wargo, *Learning Progressive Web Apps,* 1. Aufl. Addison-Wesley Professional, 2020.

[37] 1&1 IONOS SE Digital Guide, *SASS: CSS auf dem nächsten Level?* [Online]. Verfügbar unter: https://www.ionos.de/digitalguide/websites/web-entwicklung/sass/ (Zugriff am: 19. November 2020).

[38] Jan-Dirk, *Was ist Bootstrap?* [Online]. Verfügbar unter: https://www.it-talents.de/blog/it-talents/was-ist-bootstrap (Zugriff am: 23. November 2020).

[39] Tutorialzine, *What's New in the Bootstrap 4 Grid.* [Online]. Verfügbar unter: https://tutorialzine.com/2016/11/boostrap-4-regular-vs-flex-grid (Zugriff am: 27. November 2020).

[40] Bootstrap, *Modal.* [Online]. Verfügbar unter: https://getbootstrap.com/docs/5.0/components/modal/ (Zugriff am: 18. Dezember 2020).

[41] 1&1 IONOS SE Digital Guide, *Iconfonts – Webdesign mit vektorbasierten Piktogrammen.* [Online]. Verfügbar unter: https://www.ionos.de/digitalguide/websites/webdesign/iconfonts-vektorbasierte-zeichensaetze/ (Zugriff am: 25. November 2020).

[42] Fonticons, *Font Awesome.* [Online]. Verfügbar unter: https://fontawesome.com/icons/plus?style=solid (Zugriff am: 25. November 2020).

[43] FullCalendar LLC, *FullCalendar - JavaScript Event Calendar.* [Online]. Verfügbar unter: https://fullcalendar.io/ (Zugriff am: 25. November 2020).

[44] FullCalendar LLC, *Initialize Globals Demo - Demos | FullCalendar.* [Online]. Verfügbar unter: https://fullcalendar.io/docs/initialize-globals-demo (Zugriff am: 25. November 2020).

[45] Google Developers, *Lighthouse.* [Online]. Verfügbar unter: https://developers.google.com/web/tools/lighthouse/?utm_source=devtools (Zugriff am: 21. Dezember 2020).

[46] Google Developers, *Manifest doesn't have a maskable icon.* [Online]. Verfügbar unter: https://web.dev/maskable-icon-audit/?utm_source=lighthouse&utm_medium=devtools (Zugriff am: 21. Dezember 2020).

Anhang

Anhang 1: SCSS und daraus generiertes CSS

```scss
$red: #f62f18;
$font-size: 12;

//Erzeugt die Klassen .font-size-12 bis .font-size-18
@while $font-size <=18 {
    .font-size-#{$font-size} {
        font-size: ($font-size * 1px);
    }
    $font-size: $font-size + 2;
}
//Verschachtelung von span in .message und span mit Klasse .bold
.message{
    color:black;
    span{
        background-color: $red;
        &.bold{
            font-weight:700;
        }
    }
}
//Vererbung von message und lighten-Funktion für Farben
.message-light{
    @extend .message;
    background-color: lighten($red, 20%);
}
```

Quellcode 26: Eigens Beispiel für SCSS

```css
.font-size-12 {
  font-size: 12px; }

.font-size-14 {
  font-size: 14px; }

.font-size-16 {
  font-size: 16px; }

.font-size-18 {
  font-size: 18px; }

.message, .message-light {
  color: black; }
.message span, .message-light span {
  background-color: #f62f18;}
message span.bold, .message-light span.bold {
  font-weight: 700; }

.message-light {
  background-color: #fa877a; }
```

Quellcode 27: Generierte CSS-Datei

Anhang 2: Grundstruktur der App aus _Layout.cshtml

```
<body>
    <header>
        <nav class="navbar navbar-expand-lg bg-primary border-bottom shadow-sm
             w-100 fixed-top">
            <div class="container-fluid">
                @if (ViewData["Title"].ToString() != "Home")
                {
                    <a class="btn text-white"
                       asp-controller="@ViewData["LastController"]?.ToString()"
                       asp-action="@ViewData["LastView"]?.ToString()">
                        <i class="fas fa-chevron-left"></i>
                    </a>
                }
                else
                {
                    <a class="btn text-white" style="visibility: hidden;">
                        <i class="fas fa-chevron-left"></i>
                    </a>
                }
                <h4 class="navbar-brand text-white m-auto">@ViewData["Title"]</h4>
                <button class="navbar-toggler" type="button" data-toggle="collapse"
                        data-target=".navbar-collapse"
                        aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <i class="fas fa-bars text-white"></i>
                </button>
                <div class="navbar-collapse collapse d-lg-inline-flex
                     flex-lg-row-reverse bg-white p-3">
                    <ul class="navbar-nav flex-grow-1">
                        @if (User.HasPermissions())
                        {
                            <li class="nav-item">
                                <a class="nav-link text-dark" asp-controller="Home"
                                   asp-action="Index">
                                    Home
                                </a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark"
                                   asp-controller="ProjectTracking"
                                   asp-action="ProjectTrackingOverview">
                                    Project tracking
                                </a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark"
                                   asp-controller="Absences"
                                   asp-action="AbsencesOverview">Absences</a>
                            </li>
```

```
                        @if (User.HasRole("MANAGER")
                            || User.HasRole("HUMAN_RESOURCE")
                            || User.HasRole("DIRECTOR"))
                        {
                            <li class="nav-item">
                                <a class="nav-link text-dark"
                                   asp-controller="ApprovalAbsences"
                                   asp-action="ApprovalAbsencesOverview">
                                    Approval Absences
                                </a>
                            </li>
                        }
                    }
                        <partial name="_LoginPartial" />
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div id="ContentContainer">
        <main role="main">
            @RenderBody()
        </main>
    </div>
    <div id="Loader" class="position-absolute text-secondary">
        <div class="spinner-border" role="status">
            <span class="sr-only">Loading...</span>
        </div>
    </div>
</body>
```

Quellcode 28: Grundstruktur der App aus Layout-Datei