

DIPLOMARBEIT

Jugendliches Krems

Ausgeführt im Schuljahr 2022/23 von:

David Kaufmann
Johannes Spindelberger

5BHITM-05
5BHITM-13

Betreuer:

Dipl.-Ing. Anton Hauleitner
Dipl.-Ing. Anton Hauleitner

Krems, am 31.03.2023

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Krems, am 31. März 2023

Verfasser/Verfasserinnen:

David Kaufmann

Johannes Spindelberger

DIPLOMARBEIT DOKUMENTATION

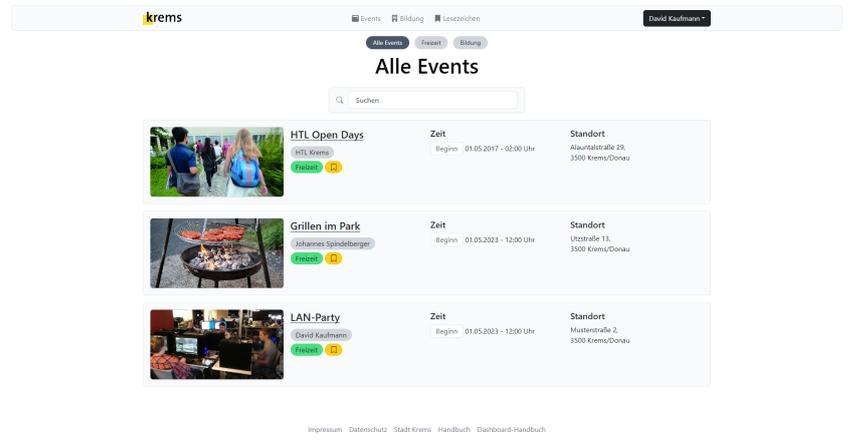
Namen der Verfasser/innen	David Kaufmann Johannes Spindelberger
Jahrgang / Klasse Schuljahr	5BHITM 2022/23
Thema der Diplomarbeit	Jugendliches Krems
Ansprechpartner	Stadt Krems an der Donau

Aufgabenstellung	Das Aufgabenstellung ist die Entwicklung eines Prototyps für die Webapplikation sowie einer Mobilapplikation für Android um das Thema auf seine Akzeptanz bei den Jugendlichen zu testen.
------------------	---

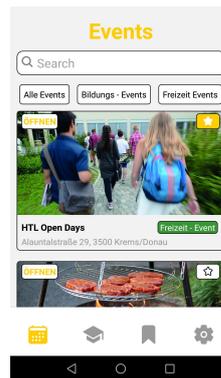
Realisierung	Durch den Einsatz der Technologien Laravel 9, Laravel Sanctum, Svelte, Inertia.js, MySQL, Tailwind und Bootstrap wurde eine Webapplikation entwickelt. Durch den Einsatz der Technologien Expo, React Native und Axios wurde eine Mobilapplikation entwickelt.
--------------	---

Ergebnisse	Im Zug der Diplomarbeit wurde eine Webapplikation und eine Mobilapplikation, auf der Informationen über Bildungsanstalten und bevorstehende Veranstaltungen in Krems, die speziell auf Jugendliche ausgerichtet sind, angezeigt werden, entwickelt.
------------	---

Typische Grafik, Foto etc.
(mit Erläuterung)



Diese Grafik zeigt die Events-Seite der Webapplikation, wo alle Events aufgelistet sind.



Diese Grafik zeigt Events-Seite der Mobilapplikation, wo alle Events aufgelistet sind.

Teilnahme an Wettbewerben, Auszeichnungen

Keine

Möglichkeiten der Einsichtnahme in die Arbeit

Schulbibliothek

Approbation (Datum / Unterschrift)

Prüfer/in

Abteilungsvorstand / Direktor/in

DIPLOMA THESIS

Documentation

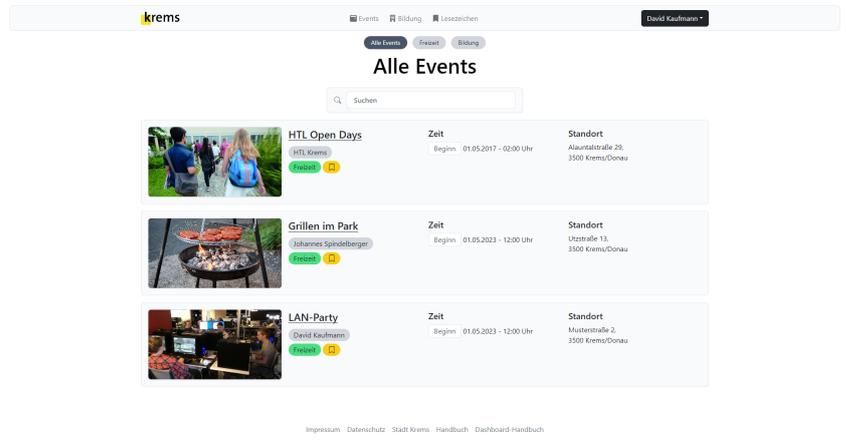
Authors	David Kaufmann Johannes Spindelberger
Form Academic year	5BHITM 2022/23
Topic	Jugendliches Krems
Contact partner	City Krems an der Donau

Assignment of tasks	The task is to develop a prototype for the web application and the mobile application for Android for testing the acceptance of the topic among young people.
---------------------	---

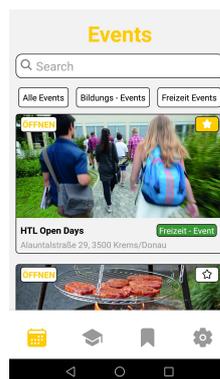
Realization	Using the technologies Laravel 9, Laravel Sanctum, Svelte, Inertia.js, MySQL, Tailwind and Bootstrap, a web application was developed. Using the technologies Expo, React Native and Axios, a mobile application was developed.
-------------	--

Results	In the course of the diploma thesis, a web application and a mobile application were developed that display information about educational institutions and upcoming events in Krems that are specifically aimed at young people.
---------	--

Illustrative graph, photo (incl. explanation)



This graphic shows the events page of the web application, where all events are listed.



This graphic shows the events page of the mobile application, where all events are listed.

Participation in competitions
Awards

None

Accessibility of diploma thesis

School library

Approval
(Date / Sign)

Examiner

Head of Department /
College

Inhaltsverzeichnis

1. Präambel	10
1.1. Team	10
1.2. Danksagung	10
1.3. Gendererklärung	10
2. Einleitung	11
2.1. Ausgangslage	11
2.2. Forschungsfrage	11
2.2.1. Spezifische Forschungsfrage - David Kaufmann	11
2.2.2. Spezifische Forschungsfrage - Johannes Spindelberger	11
3. Lösungsansätze für die Realisierung	12
3.1. Auswahl der Datenbank - KAUF	12
3.1.1. Kriterien	12
3.2. Auswahl der Backend Technologie - KAUF	13
3.2.1. Kriterien	13
3.2.2. Vergleich: Laravel und Next.js	14
3.3. Auswahl der Frontend Technologie - KAUF	17
3.3.1. Kriterien	17
3.3.2. Vergleich von Svelte, React und Vue 3	17
3.4. Auswahl der Technologie für die mobilen Applikation - SPIN	19
3.4.1. Kriterien	19
3.4.2. Vergleich von React Native, Flutter, Xamarin	21
3.4.3. Expo oder klassisches React Native	26
3.5. Technologieauswahl für mobile Kommunikation mit Laravel API - SPIN	28
3.5.1. React Native Bibliotheken	28
3.5.2. Drittanbieter Möglichkeiten	31
3.5.3. Fazit	33
4. Dokumentation der Implementierung	34
4.1. Datenstruktur - KAUF	34
4.1.1. Datenbankschema	34
4.1.2. Kriterien	34
4.1.3. Laravel Models	37
4.1.4. Laravel Migrations	39
4.1.5. Datenmanipulation mit Eloquent	41
4.2. HTTP Routing - KAUF	42
4.2.1. Definition	42
4.2.2. Routing-Dateien	44
4.2.3. Inertia Funktionen für Routen	44
4.2.4. Controller	45
4.2.5. Laravel Traits	46
4.2.6. Laravel Middleware	47
4.2.7. Strukturierung	50
4.3. E-Mail-System - KAUF	52
4.3.1. Verifizierung	52

4.3.2.	Passwort Zurücksetzung	53
4.3.3.	Laravel Notification	55
4.3.4.	Laravel Mailable	57
4.3.5.	E-Mail Vorlagen	58
4.4.	Authentifizierung - KAUF	58
4.4.1.	Webseite	58
4.4.2.	API	58
4.5.	Frontend - KAUF	59
4.5.1.	Webseitenstruktur	59
4.5.2.	Vorteile	59
4.5.3.	Webseitenstruktur von Jugendliches Krems	61
4.5.4.	CSS Frameworks	63
4.5.5.	Programmierung mit Svelte	63
4.6.	Arbeiten mit React Native und Expo - SPIN	66
4.6.1.	Grundlegendes	66
4.6.2.	Struktur von React Native	67
4.6.3.	Beispiel einer React Native Seite	68
4.6.4.	Funktionsweise von React Native	69
4.7.	Navigation in React Native - SPIN	70
4.7.1.	Struktur der mobilen Applikation	70
4.7.2.	Navigation der Hauptbereiche der Anwendung	71
4.7.3.	Navigation durch bestimmte Ereignisse	74
4.8.	Speicherung von Daten in React Native - SPIN	77
4.8.1.	Syntax und Grundlagen	77
5.	Installation	79
5.1.	Installation der Laravel Applikation - KAUF	79
5.1.1.	Voraussetzungen	79
5.1.2.	Dependencies installieren	79
5.1.3.	Datenbank	80
5.1.4.	Angabe des E-Mail Servers	80
5.1.5.	Run development environment	80
5.1.6.	Deployment	81
5.2.	Installation der Mobilapplikation - SPIND	81
5.2.1.	Voraussetzungen	81
5.2.2.	Dependencies installieren	81
5.2.3.	Anpassen der API	82
5.2.4.	Ausführen der Applikation	82
5.2.5.	Verbinden über Expo Go	82
6.	Zusammenfassung und Ausblick	83
6.1.	Kaufmann David	83
6.2.	Spindelberger Johannes	84
I.	Literaturverzeichnis	85
II.	Abbildungsverzeichnis	88

III. Tabellenverzeichnis	89
IV. Quellcodeverzeichnis	90
V. Abkürzungsverzeichnis	92
A. Anhang	93
A.1. Projektstagebücher	93
A.1.1. Projektstagebuch David Kaufmann	93
A.1.2. Projektstagebuch Johannes Spindelberger	94
A.2. Datenträgerbeschreibung	95

1. Präambel

1.1. Team

Das Projektteam besteht aus Herrn David Kaufmann (KAUF) und Herrn Johannes Spindelberger (SPIND). Der Betreuer des Projektteams ist Herr Prof. Dipl.-Ing. Anton Hauleitner, Abteilungsvorstand für Informationstechnologie an der HTL Krems.

1.2. Danksagung

Das Projektteam möchte sich ganz herzlich bei der Stadt Krems für Ihre Kooperation und Zurverfügungstellung von Server-Infrastruktur bedanken. Ebenfalls bedankt sich das Projektteam bei ihrem Betreuer Hr. Prof. Dipl.-Ing. Anton Hauleitner.

1.3. Gendererklärung

Zur besseren Lesbarkeit der Diplomarbeit wurde ausschließlich die männliche Form verwendet. Da Begriffe wie "Benutzerinnen und Benutzer" den Text unleserlich machen, wurde es schlicht auf "Benutzer" gekürzt, dies soll jedoch keine Geschlechterdiskriminierung zum Ausdruck bringen

2. Einleitung

2.1. Ausgangslage

Als Schul- und Universitätsstadt erfreut sich die Stadt Krems an der Donau einer hohen Anzahl von Jugendlichen und jungen Erwachsenen, welche in Krems ihre Studien- und Freizeit verbringen. In der Stadt gibt es daher auch viele Angebote für Jugendliche und junge Erwachsene. Diese Angebote werden der Zielgruppe jedoch derzeit nicht im expliziten Maße vermittelt. Um dies zu ändern, möchte die Stadt Krems dieser Zielgruppe eine Anwendung zur Verfügung stellen, um die Angebote ansprechend zu vermitteln. Eine pilothafte Umsetzung wird mit der Erstellung einer Webapplikation und Mobilapplikation gewünscht.

2.2. Forschungsfrage

2.2.1. Spezifische Forschungsfrage - David Kaufmann

Wie sieht die optimale Umsetzung aus, eine Webapplikation mit Administratorfunktionen, welche außerdem eine API für Kommunikation mit externen Applikationen zur Verfügung stellt, zu programmieren und wie wird diese implementiert?

2.2.2. Spezifische Forschungsfrage - Johannes Spindelberger

Wie sieht die optimale Umsetzung aus, eine Mobilapplikation, welche mit einer externen API kommuniziert, zu programmieren und diese zu implementieren?

3. Lösungsansätze für die Realisierung

3.1. Auswahl der Datenbank - KAUF

Die richtige Auswahl der Datenbank hat eine entscheidende Rolle in der Leistung und Skalierbarkeit von Jugendliches Krems. Vor allem bei einer großen Benutzeranzahl und großen Datenmengen sind Abfragegeschwindigkeit und Eigenschaften der Skalierbarkeit der Datenbank hier essentiell.

3.1.1. Kriterien

Bei der Auswahl der richtigen Datenbank sind mehrere Faktoren zu beachten:

3.1.1.1. Kompatibilität

Aufgrund der Auswahl von Laravel als Backend-Technologie soll eine Datenbank ausgewählt werden, für welche Laravel eine integrierte Implementation bereitstellt. Momentan bietet Laravel für die Datenbanken MySQL, MariaDB, PostgreSQL, SQL Server und First-Party-Support. [1] Weitere Datenbanken wie MongoDB können durch das Installieren von Paketen ebenfalls einfach verwendet werden.

3.1.1.2. Datenmodell

Die Daten von Jugendliches Krems ändern sich nicht mit großer Häufigkeit. Die meisten Daten werden einmal eingetragen und werden danach nicht mehr oder nur selten verändert. Außerdem handelt es sich ausschließlich um strukturierte Daten. Aufgrund der Möglichkeit der klaren Definition von Datentypen und systemischen Eintragung der Daten eignen sich hierfür relationale Datenbanken. Ein Nachteil von relationalen Datenbanken ist jedoch, dass sie in der Regel keine Failover-Funktionalität bieten, um eine bessere Verfügbarkeit zu bieten. Failover-Funktionalität sorgt dafür, dass im Falle eines Ausfalls der primären Datenbank automatisch auf eine andere Datenbank gewechselt wird, damit die erforderlichen Daten weiterhin verfügbar sind.

3.1.1.3. Sicherheit

Es ist eine Notwendigkeit, sich für eine Datenbank zu entscheiden, welche ausgetestet ist und aktiv weiterentwickelt wird, um zu vermeiden, dass Unbefugter Zugang zu den Daten von Jugendliches Krems erhalten. Dieses Kriterium schränkt die Auswahlmöglichkeiten jedoch nicht näher ein, da MySQL, MariaDB, PostgreSQL, SQL Server, SQLite auch MongoDB alle aktiv weiterentwickelt werden und eine große Community haben.

3.1.1.4. Open Source und Lizenzierung

Die Datenbanken MySQL, MariaDB, PostgreSQL, SQLite und MongoDB sind Open Source. SQL Server hat eine kostenlose Developer-Edition, diese darf man jedoch nur für die Entwicklung verwenden. [2] PostgreSQL und SQLite haben praktisch keine Einschränkungen

hinsichtlich der kommerziellen Nutzung. MongoDB bietet eine SSPL (Server Side Public License), welche es erlaubt, unter der Einhaltung von bestimmten Bedingungen, MongoDB kostenlos zu verwenden. [3] Oft erfordert MongoDB jedoch den Erwerb einer kommerziellen Lizenz. MongoDB, SQL Server, MariaDB und MySQL enthalten in ihren kommerziellen Produkten erweiterte Funktionalität. Da man es der Stadt ermöglichen will, die Anwendungen verwenden zu können, ohne Lizenzen kaufen zu müssen oder bestimmte Kriterien einhalten zu müssen, sind für Jugendliches Krems die Datenbanken MySQL, MariaDB, PostgreSQL und SQLite am besten geeignet.

3.1.1.5. Beliebtheit und Ressourcen

Da auch Skalierbarkeit, Flexibilität und Leistung keine Ausschlusskriterien für MySQL, MariaDB, PostgreSQL, SQL Server, SQLite und MogoDB darstellen, sind die Letztkriterien die Beliebtheit und verfügbaren Ressourcen der jeweiligen Datenbanken. Hier kann MySQL am meisten punkten. MySQL ist die zweitbeliebteste Datenbank der Welt [4] (Stand 22.03.2023) und bietet somit sehr viele Ressourcen.

3.1.1.6. Fazit

Aufgrund der hohen Beliebtheit und Anzahl von Ressourcen und der bereits existierenden Erfahrung der Entwickler wurde MySQL als Datenbank für Jugendliches Krems ausgewählt.

3.2. Auswahl der Backend Technologie - KAUF

Die richtige Auswahl der Backend-Server-Technologie für Jugendliches Krems ist essentiell.

3.2.1. Kriterien

Folgende Faktoren müssen hierbei berücksichtigt werden:

3.2.1.1. Leistung

Ladezeiten und generelle Leistungsmerkmale einer Webseite hängen stark von deren Backend-Technologie ab. Daraus ergibt sich die Wichtigkeit, aus den Auswahlmöglichkeiten (siehe Kapitel 3.2.2) eine Backend-Technologie auszuwählen, die starke Leistungsmerkmale aufweist und Nutzern attraktive Ladezeiten bietet.

3.2.1.2. Skalierbarkeit

Eine Backend Technologie sollte dazu in der Lage sein, bei einem rasanten Wachstum der Nutzerzahlen mitzuhalten. Die Ladezeiten sollten auch bei starken Abweichungen der regulären Nutzerzahlen gering bleiben, um die Benutzerfreundlichkeit der Webseite nicht einzuschränken. Eine Möglichkeit von Backend-Frameworks, dies zu garantieren, besteht zum

Beispiel in einer effizienten Ressourcenverwaltung. Das Backend-Framework sollte in der Lage sein, für mindestens alle 24944 Einwohner der Stadt Krems [5] (Stand 2022) stabil zu laufen.

3.2.1.3. Sicherheit

Um sensitive Information wie Nutzerdaten zu schützen und den Server vor Hackerangriffen zu schützen, ist es essentiell, ein modernes Framework auszuwählen, das Methoden beinhaltet, um vor diesen Problemen zu schützen.

Es ist essentiell, ein Framework zu wählen, welches aktiv weiterentwickelt wird und regelmäßig Sicherheitsupdates erhält. Wenn die Weiterentwicklung eingestellt wird oder nur begrenzt stattfindet, besteht die Gefahr, dass Sicherheitslücken und Schwachstellen nicht rechtzeitig gepatcht werden, und das System anfällig für Hacker und Fehler wird.

3.2.1.4. Wartbarkeit

Die richtige Backend-Technologie erhöht, ebenso wie eine gute Strukturierung, die Wartbarkeit der Webseite. Durch smarte Design Patterns und Abläufe ermöglichen Backend Frameworks Entwicklern effizient und schnell Änderungen in der Applikation vorzunehmen.

3.2.2. Vergleich: Laravel und Next.js

Im Zuge der Analyse dieser Faktoren ist es möglich, verschiedene Backend-Frameworks miteinander zu vergleichen und eine Auswahl zu treffen. Es gibt im Bereich von Backend Frameworks jedoch zwei weit verbreitete unterschiedliche Ansätze. Es gibt Frameworks wie Laravel, welche nur auf der Serverseite ausgeführt werden und Frameworks wie Next.js, welche sowohl auf der Serverseite als auch auf der Clientseite ausgeführt werden. Um beide Ansätze zu vergleichen, ist es sinnvoll, einen Blick auf Next.js und Laravel als Vertreter der jeweiligen Ansätze zu werfen:

3.2.2.1. Programmiersprache

Während Laravel auf PHP basiert, baut Next.js auf Node.js auf und ist somit ein JavaScript Framework. Ein grundlegender Unterschied zwischen PHP und Node.js ist: "Node.js ist im Gegensatz zu PHP keine Sprache, sondern eine Runtime Environment (RTE), eine Laufzeitumgebung, die JavaScript für die Entwicklung von serverseitigen Webanwendungen verwendet." [6] (Stand 22.03.2023)

Das bedeutet, dass bei einer Entscheidung für Next.js JavaScript für die serverseitige Entwicklung verwendet wird, während bei einer Entscheidung für Laravel PHP verwendet wird.

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems	
	Abteilung:	Informationstechnologie
Ausbildungsschwerpunkt:		Medientechnik

3.2.2.2. Frontend Ansatz

Im Gegensatz zu Next.js ist Laravel ausschließlich ein Backend-Framework. Das bedeutet, dass es keine integrierten Frontend-Tools bietet, und, wenn als Wunsch moderne Frontend-Ansätze wie Komponenten implementieren werden wollen, die gesamte Frontend-Darstellung separat mit einem Frontend-Framework umgesetzt werden muss. Nach der Entscheidung für ein Frontend-Framework wie zum Beispiel Vue, React oder Svelte, müssen beide Komponenten noch miteinander verbunden werden. Dies kann entweder über die Entwicklung von zwei unterschiedlichen Projekten, die mit einer API miteinander kommunizieren, oder mithilfe einer Technologie wie Inertia.js funktionieren.

“Inertia isn’t a framework, nor is it a replacement for your existing server-side or client-side frameworks. Rather, it’s designed to work with them. Think of Inertia as glue that connects the two. Inertia does this via adapters. We currently have three official client-side adapters (React, Vue, and Svelte) and two server-side adapters (Laravel and Rails).” [7] (Stand 22.03.2023)

Next.js hingegen ist für die Entwicklung von SPAs mit React ausgelegt. Es bietet bereits out-of-the-box integrierte Tools für die Entwicklung von Frontend-Komponenten mit React.

3.2.2.3. Server Side Rendering

Ein Vorteil von Next.js gegenüber Laravel ist die integrierte Funktionalität für Server Side Rendering. Next.js bietet bereits direkt umfassende Unterstützung für SSR.

“Beim serverseitigen Rendering verarbeitet der Server Ihrer Website die Daten und wandelt sie in leicht lesbare Informationen für die Webbrowser der Besucher um. Die Besucher senden eine Anfrage, wenn sie versuchen, eine Webseite aufzurufen. Der Server Ihrer Website antwortet daraufhin, indem er alle Daten der Seite abrufen und sie in einen einzelnen, vollständigen Hypertext umwandelt Markup Language (HTML)-Webseite, die sie an die Webbrowser der Besucher übermittelt.” [8] (Stand 22.03.2023)

Laravel hingegen ist nicht für SSR ausgelegt und enthält keine nativen Funktionen für diese Technologie. Jedoch heißt dies nicht, dass es nicht möglich ist, SSR in Laravel umzusetzen. Es erfordert jedoch zusätzlichen Aufwand in der Konfiguration und Technologien wie zum Beispiel Inertia.js, welche Funktionalitäten für SSR bieten.

3.2.2.4. Architektur

Laravel verwendet das MVC Design Pattern. Das heißt, dass Code für die Datenmodelle (Model), die Darstellung im Browser (View) und Funktionalität für die Manipulation der Anwendung (Controller) separat geschrieben werden. Next.js verwendet hingegen eine Komponenten-basierte Architektur. Funktionen werden hier in wiederverwendbaren Komponenten organisiert. die sowohl die Darstellung als auch die Logik implementieren.

3.2.2.5. Datenbank-Integration

Laravel bietet Entwicklern integrierte Unterstützung der Datenbanken MySQL, MariaDB, PostgreSQL, SQL Server und SQLite. [1] Es bietet für diese Datenbanken außerdem integrierte Unterstützung für Eloquent, den standardmäßigen ORM von Laravel. Dies vereinfacht die Arbeit mit Daten aus einer Datenbank erheblich, da anders als bei Next.js keine Technologien mehr installiert werden müssen.

Um in einer Next.js Applikation mit einer Datenbank zu arbeiten, müssen für die benötigte Funktionalität externe Technologien installiert werden. Technologisch gesehen ist dies keine Herausforderung, jedoch stellt es einen Mehraufwand dar.

3.2.2.6. Community und Ressourcen

Damit auf Probleme oder Fragen in der Entwicklung möglichst schnell eine Antwort gefunden wird, ist es von Vorteil, Zugriff auf eine gute Dokumentation und eine große Community von Leuten, die einem weiterhelfen können, zu haben.

Mit 103K Github-Sternen [9] (Stand 22.03.2023) hat Next.js im Vergleich zu Laravel mit 72.9K GitHub Sternen [10] (Stand 22.03.2023) eine größere Community als Laravel.

Beide Frameworks haben eine ausführliche Dokumentation mit detaillierten Erklärungen und Beispielen zu Funktionen und Konzepten.

Aufgrund der langen Entwicklungszeit von Laravel gibt es insgesamt mehr Ressourcen zu Laravel als zu Next.js. Auf Stackoverflow wurden zum Beispiel 204472 Fragen zu Laravel gestellt, während es bei Next.js noch 58230 Fragen sind.

Schlussendlich ist eine Entscheidung zwischen den beiden Frameworks applikationsspezifisch und abhängig von den Präferenzen des Programmierers. Da die Applikation für Jugendliches Krems eine umfangreiche Kommunikation mit der Datenbank benötigt, und eine API zur Kommunikation mit der Mobilapplikation bereitstellen können muss, wird Laravel mit seinen umfangreicheren integrierten Backend-Funktionen als das besser geeignete Framework befunden.

3.3. Auswahl der Frontend Technologie - KAUF

Die Auswahl des richtigen Frontend-Frameworks ist von großer Bedeutung, da der Prozess der Erstellung des Frontends arbeitsintensiv und von hoher Wichtigkeit ist, und das Frontend-Framework bestimmt, wie dieser Prozess abläuft. Ein gutes Framework muss dafür sorgen, dass Entwickler Benutzeroberflächen effektiv und effizient erstellen können. Aber auch abgesehen von der Developer-Experience gibt es wichtige Kriterien für die Wahl eines Frontend-Frameworks:

3.3.1. Kriterien

3.3.1.1. Leistung

Geringe Ladezeiten einer Webseite sind essentiell, um eine hohe Benutzerfreundlichkeit für Endnutzer bereitzustellen. Eine wesentliche Komponente ist hierbei das Frontend-Framework.

3.3.1.2. Code-Komplexität

Für eine gute DX ist die Code-Komplexität eines Frontend-Frameworks von großer Bedeutung. Wenn häufig derselben Code neu geschrieben werden muss, führt dies zu unnötiger Arbeitszeit und einem ineffizienten Arbeitsprozess.

3.3.1.3. Wartbarkeit

Ein gutes Frontend-Framework bietet klare Strukturen und Richtlinien, wie eine Komponente aussehen muss. Außerdem ist es unbedingt zu empfehlen, dass diese Strukturen und Richtlinien logisch nachvollziehbar sind, damit auch Entwickler, die nicht sehr gut mit dem Framework vertraut sind, im Falle einer benötigten Änderung einer Komponente, jene effizient ausführen können, ohne dass viel Zeit für die Einarbeitung beansprucht wird.

3.3.1.4. Kompatibilität

Jedenfalls muss das Frontend-Framework, welches im Auswahlverfahren ausgewählt wurde, mit anderen Technologien und dem Backend kompatibel sein. In dem Fall der Webapplikation ist dies Laravel (siehe Kapitel 3.2). Das Frontend-Framework wurde im Auswahlverfahren auf der Grundlage dieser Anforderungen ausgewählt.

3.3.2. Vergleich von Svelte, React und Vue 3

Um nicht über eine API zwischen Frontend und Backend kommunizieren zu müssen, wird die Verwendung des Frontend-Framework in Kombination mit Inertia.js empfohlen. Inertia.js unterstützt die Frontend-Frameworks Vue 2, Vue 3, React und Svelte. Da Vue 2 veraltet ist und nur mehr im Sinne von Kompatibilität weiterentwickelt wird, muss für die Webapplikation zwischen folgenden Möglichkeiten gewählt werden: den Frontend-Frameworks Vue 3, React und Svelte. Im Sinne dieser Möglichkeiten folgt ein Vergleich jener Frontend-Frameworks.

3.3.2.1. Virtual DOM vs Compiler

React und Vue verwenden einen Virtual DOM. Dies ist eine virtuelle Kopie des Document Object Models. Diese dient als Zwischenschicht zwischen der Anwendungslogik und dem tatsächlichen DOM. Immer wenn dann die Benutzeroberfläche geändert wird, wird zuerst der virtuelle DOM aktualisiert und anschließend mit dem tatsächlichen DOM verglichen. Danach wird der tatsächliche DOM ausschließlich dort aktualisiert, wo er vom Virtual DOM abweicht.

Svelte hingegen benutzt keinen Virtual DOM sondern enthält einen Compiler.

“Der Code wird zur Übersetzungszeit vom Svelte-Compiler in JavaScript-Code übersetzt, der von keinen externen Programmbibliotheken abhängig ist. Andere Frameworks setzen auf das Konzept des Virtual DOM, das Änderungen im Zustand mithilfe komplexer Algorithmen erkennt und den Anwendungsbaum selektiv aktualisiert. Diese rechenaufwändigen Operationen werden umgangen, indem der von Svelte zur Verfügung gestellte Compiler automatisch Hilfsfunktionen zum manuellen Manipulieren des DOM generiert.” [11] (Stand 22.03.2023)

Die Verwendung dieses Ansatzes beschleunigt die Aktualisierung der Benutzeroberfläche und verursacht weniger überflüssigen Code.

3.3.2.2. Syntax

Alle 3 Frameworks verwenden eine anderen Syntax. React verwendet JSX, “[...] eine von Facebook für React entwickelte Domänenspezifische Sprache (DSL), um mit einer HTML-artigen Syntax innerhalb von JavaScript Oberflächen zu beschreiben.” [12] (Stand 22.03.2023)

Vue 3 verwendet hingegen einen Template-Syntax, der sich HTML sehr nah ähnelt.

“Das template stellt das zu rendernde HTML der Komponente dar und kann unter anderem mit Directives und Interpolations angereichert werden, um die Logik abzubilden. Nach diesem Prinzip arbeiten nahezu alle Template-Engines.” [13] (Stand 22.03.2023)

Svelte verwendet eine komplett eigenständige Syntax, die Vanilla Javascript-Code ähnelt. Im Vergleich zu den Syntaxen von React und Vue 3 ist diese aber sehr einfach aufgebaut und zu lernen, da sie keine komplexen Konzepte oder spezielle Schreibweisen verwendet.

Welche Syntax bevorzugt wird, ist im Endeffekt ein Gegenstand der eigenen Beurteilung. Jedoch werden die meisten Menschen die Syntax von Svelte bevorzugen, da diese keine neuen Schreibweisen einführt und sehr einfach gehalten ist. Auf den ersten Blick wird nicht einmal ein Unterschied zwischen Svelte-Code und normalem HTML-Code erkannt.

3.3.2.3. Größe und Leistung

Durch den Compileransatz, den Svelte verwendet, reduziert Svelte auch die Größe der Applikation, da unnötiger Code in der Übersetzungszeit eliminiert wird. Der Virtual-DOM-Ansatz von React und Vue führt hier zu einem wesentlich größeren Speicherverbrauch.

3.3.2.4. Community und Dokumentation

React und Vue 3 bieten beide sehr große Communities und eine ausführliche Dokumentation. Svelte hingegen ist noch sehr neu, und hat in Folge eine kleine Community und noch keine ausführliche Dokumentation. Wenn also eine Entscheidung getroffen wird, eine Applikation mit Svelte zu programmieren, sollte die Bewusstmachung im Vordergrund stehen, dass womöglich länger gebraucht wird, um Fehler oder Fragen zu lösen.

3.3.2.5. Fazit

Wenngleich Svelte den Nachteil einer neueren Technologie mit einer kleineren Community und weniger Dokumentation aufweist, bietet jene Technologie jedoch viele Vorteile wie kleinere Anwendungen, bessere Leistung, und eine einfacheren Syntax. Aus diesem Grund wird für die Webapplikation Svelte als Frontend-Framework ausgewählt.

3.4. Auswahl der Technologie für die mobilen Applikation - SPIN

Die Auswahl der richtigen Technologie zur Entwicklung der Mobilapplikation ist für Jungendliches Krems ein essentieller Faktor.

3.4.1. Kriterien

Folgende Faktoren spielen hierbei eine wesentliche Rolle:

3.4.1.1. Leistung

Kurze Ladezeiten einer mobilen Anwendung sind von entscheidender Bedeutung, um ein hohes Maß an Benutzerfreundlichkeit für die Endnutzer zu gewährleisten.

3.4.1.2. Plattform

Die Wahl einer geeigneten Plattform für die Entwicklung einer mobilen Anwendung ist in erster Linie von der Zielgruppe abhängig. Das Hauptanliegen von Jungendliches Krems besteht darin, eine möglichst breite Masse anzusprechen. Daher sollte die Anwendung sowohl für iOS- als auch für Android-Benutzer verfügbar sein.

Im Bereich der iOS-Entwicklung wird die Programmiersprache Swift verwendet. Swift ist effizient und leistungsfähig, ermöglicht Echtzeit-Feedback und kann problemlos in bestehenden Objective-C-Code integriert werden. Dies führt dazu, dass Entwickler einen sichereren und verlässlicheren Code erstellen, Zeit einsparen und ihre Anwendungen weiter optimieren können.

Um eine mobile Anwendung für iOS zu entwickeln, ist ein Mac-Computer erforderlich. Dies liegt daran, dass die iOS-App-Entwicklung in der Anwendung Xcode erfolgt, welche ausschließlich auf Apple-Software auf iMac, MacBook oder iPad läuft [14].

Darüber hinaus ist es notwendig, Mitglied im Apple Developer Program zu sein, um die Anwendung zu veröffentlichen und zu testen. Die Mitgliedschaft kostet 99 US-Dollar pro Jahr, ermöglicht jedoch den Zugang zu allen relevanten Softwareprogrammen [14].

Bei der Entwicklung von Android-Anwendungen ist eine Android-Entwicklungsumgebung erforderlich. Google stellt hierfür das kostenlose Android Studio als offizielle Entwicklungsumgebung bereit. Die Programmierung von Android-Apps erfolgt in der objektorientierten Programmiersprache Java, die ursprünglich von Sun Microsystems entwickelt und 2010 von Oracle übernommen wurde [15]. Java-Technologie besteht grundlegend aus dem Java Development Kit (JDK) zum Erstellen von Java-Programmen und der Java Runtime Environment (JRE) für deren Ausführung.

Da es verschiedene Versionen der Android-Plattform gibt, die von unterschiedlichen Geräten unterstützt werden, sind auch verschiedene API-Level vorhanden. Diese definieren die jeweils verfügbaren Funktionen und Komponenten der Plattform. Die Nutzung der Android SDK (Software Development Kit) ermöglicht die Arbeit mit den verschiedenen API-Leveln. Das Android SDK umfasst eine Sammlung von Entwicklerwerkzeugen, Bibliotheken und APIs (Application Programming Interfaces), die für die Erstellung von Android-Anwendungen benötigt werden.

Für die Veröffentlichung einer Anwendung im Google Play Store sind einige Voraussetzungen zu beachten. Neben einem Google-Konto ist ein kostenpflichtiger Zugang zur Google Play Developer Console erforderlich [16].

Die Cross-Plattform-Entwicklung ermöglicht es, eine mobile Anwendung auf verschiedenen Betriebssystemen auszuführen, wobei zwischen der nativen und der hybriden Cross-Plattform-Entwicklung unterschieden wird.

Bei der nativen Cross-Plattform-Entwicklung kommen Frameworks wie React Native, Xamarin oder Flutter zum Einsatz, um native Anwendungen für mehrere Plattformen zu entwickeln. Entwickler können dabei eine einzige Codebasis verwenden, um native Apps für iOS, Android und andere Plattformen zu erstellen. Der Hauptvorteil dieser Methode liegt in der nativen und leistungsfähigen Ausführung der Anwendung auf jeder Plattform, da sie die Vorteile der jeweiligen plattformspezifischen Sprachen und APIs nutzt.

Die hybride Cross-Plattform-Entwicklung hingegen setzt auf Frameworks wie Apache Cordova, Ionic oder PhoneGap, um Anwendungen mit Webtechnologien wie HTML, CSS und Javascript zu erstellen. Diese Anwendungen werden anschließend in eine native App konvertiert, indem sie in einem nativen Container ausgeführt werden. Der Vorteil dieser Methode besteht darin, dass Anwendungen für verschiedene Plattformen mit einer gemeinsamen Codebasis entwickelt werden können, ohne sich um die Unterschiede in den Programmiersprachen sorgen zu müssen. Der Nachteil besteht jedoch darin, dass hybride Anwendungen im Vergleich zu nativ entwickelten Anwendungen weniger leistungsfähig sind und der Zugriff auf native Funktionen und APIs schwieriger ist.

Abschließend ist für Jugendliches Krems eine leistungsfähige mobile Anwendung gefordert, die für alle Nutzer zugänglich sein soll. Daher stellt die native Cross-Plattform-Entwicklung die optimale Lösung dar.

3.4.1.3. Code Komplexität

Die Qualität der Entwicklererfahrung bei der Verwendung eines Frameworks für mobile Anwendungen hängt es maßgeblich von der Code-Komplexität ab. Das wiederholte Schreiben ähnlicher oder identischer Codeabschnitte kann zu überflüssigem Arbeitsaufwand und einer ineffektiven Arbeitsweise führen.

3.4.1.4. Wartbarkeit

Ein leistungsfähiges Framework für mobile Anwendungen stellt deutliche Strukturen und Vorgaben bereit, die das Erscheinungsbild einer Komponente bestimmen. Es ist zudem wichtig, dass diese Strukturen und Vorgaben logisch verständlich gestaltet sind, sodass auch Entwickler, die weniger vertraut mit dem Framework sind, bei Bedarf Änderungen an einer Komponente effektiv durchführen können, ohne einen erheblichen Zeitaufwand für die Einarbeitung aufwenden zu müssen.

3.4.1.5. Kompatibilität

In jedem Fall sollte das im Auswahlprozess gewählte Framework für mobile Anwendungen mit anderen Technologien und dem Backend kompatibel sein. Im Kontext der betrachteten mobilen Anwendung ist dies die Laravel API der Webanwendung.

3.4.2. Vergleich von React Native, Flutter, Xamarin

3.4.2.1. React Native

React Native ist ein Open-Source-Framework für die plattformübergreifende Entwicklung mobiler Anwendungen, das von Facebook ins Leben gerufen wurde und somit als Computersoftware öffentlich verfügbar ist. Erstmals präsentiert wurde React Native im Jahr 2015 auf der React.js-Konferenz von Facebook und im März desselben Jahres erschien die Beta-Version. Ursprünglich unterstützte das Framework ausschließlich iOS-Betriebssysteme,

doch aufgrund seines starken Wachstums können React Native-Anwendungen inzwischen problemlos auch auf Plattformen wie Android, Windows und Tizen eingesetzt werden. Die Anwendungen werden hauptsächlich in Javascript geschrieben. [17] [18]

React Native verwendet einen sogenannten Virtual DOM, eine virtuelle Kopie des Document Object Models. Dieser dient als Vermittlungsschicht zwischen der Anwendungslogik und dem eigentlichen DOM. Bei Änderungen der Benutzeroberfläche wird zunächst der virtuelle DOM aktualisiert und anschließend mit dem tatsächlichen DOM abgeglichen. Der tatsächliche DOM wird dann nur dort aktualisiert, wo er vom Virtual DOM abweicht. [19]

Das Framework arbeitet mit Komponenten, die mithilfe der Native API in den entsprechenden Code des Ziel-Betriebssystems umgewandelt werden. Durch diese Funktionalität kann eine Anwendung auf mehreren Betriebssystemen mit nur einer Codebasis betrieben werden.

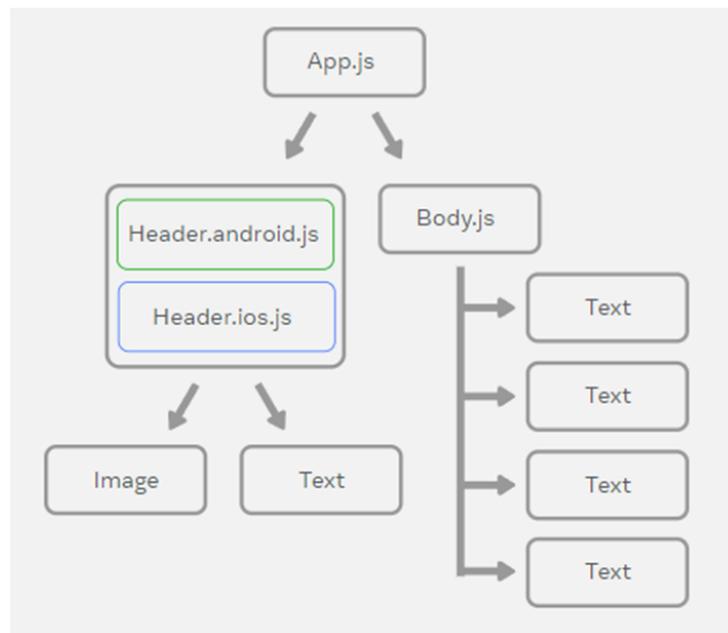


Abbildung 3.1.: Funktionsweise React Native API
[17]

React Native setzt auf Javascript XML (JSX), eine Syntaxerweiterung für Javascript, um Komponenten zu erstellen und zu modifizieren, ohne dabei die Struktur oder den Code komplizierter zu gestalten.

```
1 const user = 'Max_Mustermann';  
2 const element = (  
3   <div>  
4     <h1 className='greeting'>Hallo, {user}</h1>  
5     <a href='https://www.reactjs.org'>link</a>  
6   </div>  
7 );
```

Quellcode 3.1: Beispiel eines JSX basierten Codes zum Festellen der Code Komplexität

Zudem ermöglicht React Native das sogenannte Hot Reload. Dies bedeutet, dass sich die Anwendung nach jeder Speicherung des Quellcodes automatisch aktualisiert. Der Vorteil dieses Ansatzes liegt darin, dass die Applikation nicht bei jeder Änderung komplett neu erstellt werden muss.

3.4.2.2. Flutter

Flutter kann als Framework zur Entwicklung von Apps für verschiedene Plattformen bezeichnet werden. Es wurde grundsätzlich von Google entwickelt im Jahr 2018 erstmals als Open-Source-Projekt veröffentlicht. Bei Flutter gibt es eine große Anzahl an Bibliotheken für sogenannte Standard-UI-Elemente, die bei Android oder iOS benötigt werden. Das Framework wird allerdings auch zur Entwicklung von klassischen Desktop-Webanwendungen eingesetzt. [20]

Flutter besteht aus zwei wichtigen Komponenten:

- einem Software Development Kit (SDK)
- einem Framework (Widget-basierte UI-Library)

Von anderen Frameworks unterscheidet sich Flutter unter anderem deshalb, weil es weder WebView noch OEM-Widgets verwendet, die sich ab Werk auf dem Endgerät befinden. Wenn eine Native App mit der Plattform interagiert, um ein Widget zu erstellen, greift sie in der Regel auf die OEM-Widgets (Original Equipment Manufacturer Widgets) auf dem Smartphone oder Tablet zurück. Bei Flutter entfällt dieser Zwischenschritt. Das Framework nutzt stattdessen die eigene Hochleistungsrender-Engine Skia [21], um die notwendigen Widgets zu kreieren. Die Open Source Grafik-Library zur Erstellung von 2D-Grafiken verbindet sich über Schnittstellen mit den jeweils vorhandenen, Betriebssystem abhängigen, nativen Software Development Kits.

Flutter wird mit Hilfe von Ahead-of-time-Compilern (AOT-Compiler) ausgeführt. Lediglich in der Entwicklungsphase wird für schnellere Testprozesse auf Just-in-time-Compiler (JIT-Compiler) zurückgegriffen. Jede auf Flutter basierende App ist aus Container-, Text-, Image-, Icon- und anderen Widgets zusammengesetzt, die auf einem von der Grafik-Library Skia betriebenen Canvas-Element einzeln interpretiert und dargestellt werden. Die Plattform analysiert die so konstruierten Widgets dann und leitet die durch die Interaktion des Endnutzers provozierten Events an die App weiter. [20]

Flutter bietet genauso wie React Native eine Hot Reload funktion.

```
1 import 'package:flutter/material.dart';
2 void main() => runApp(MyApp());
3 class MyApp extends StatelessWidget {
4   @override
5   Widget build(BuildContext context) {
6     return MaterialApp(
```

```

7 title: 'Welcome_to_Flutter',
8 home: Scaffold(
9 appBar: AppBar(
10 title: Text('Welcome_to_Flutter'),
11 ),
12 body: Center(
13 child: Text('Hello_World'),
14 ),
15 ),
16 );
17 }
18 }

```

Quellcode 3.2: Beispiel eines Dart basierten Codes zum Festellen der Code Komplexität

3.4.2.3. Xamarin

Xamarin ist ein Unternehmen, das 2016 von Microsoft erworben wurde. Es wurde gegründet, um die von Microsoft entwickelte Programmiersprache C# und die zugehörigen APIs des .NET-Frameworks auch auf mobilen Plattformen verfügbar zu machen. [22]

Durch die Verwendung von Xamarin können Entwickler etwa 90 Prozent ihrer Anwendung über verschiedene Plattformen hinweg gemeinsam nutzen. Dies ermöglicht es ihnen, die gesamte Geschäftslogik in einer einzigen Programmiersprache zu verfassen (oder vorhandenen Anwendungscode zu nutzen) und gleichzeitig native Performance, ein natives Erscheinungsbild und eine native Servicequalität auf jeder Plattform zu gewährleisten. Xamarin-Anwendungen können auf einem PC oder einem Mac erstellt werden und werden in native Anwendungspakete, wie zum Beispiel APK-Dateien für Android oder IPA-Dateien für iOS, kompiliert. [23] [24]

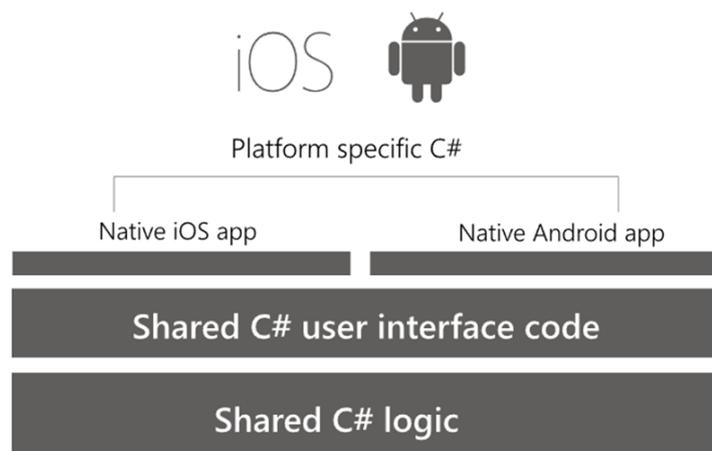


Abbildung 3.2.: Funktionsweise Xamarin
[24]

```
1 using System;
2 using Xamarin.Forms;
3
4 namespace MyXamarinApp
5 {
6     public class App : Application
7     {
8         public App()
9         {
10            MainPage = new ContentPage
11            {
12                Content = new StackLayout
13                {
14                    VerticalOptions = LayoutOptions.Center,
15                    Children = {
16                        new Label {
17                            HorizontalTextAlignment = TextAlignment.Center
18                        },
19                        new Label {
20                            Text = 'Welcome_to_Xamarin_Forms!'
21                        }
22                    }
23                }
24            };
25        }
26        public void SayHello()
27        {
28            Console.WriteLine('Hello ,_World! ');
29        }
30    }
```

Quellcode 3.3: Beispiel eines Xamarin basierten Codes zum Festellen der Code Komplexität

Ein Nachteil von Xamarin ist, dass der Einarbeitungsprozess in die Syntax und die Funktionsweise des Frameworks relativ langwierig sein kann.

3.4.2.4. Community und Dokumentation

Alle erwähnten Frameworks verfügen über eine solide Dokumentation. Sowohl Flutter als auch React Native profitieren davon, dass sie regelmäßig aktualisiert werden und stetig neue Funktionen hinzugefügt werden.

Die Dokumentation von React Native ist besonders übersichtlich gestaltet. Sie bietet Informationen zu sämtlichen Funktionen der Native API und enthält zudem Beispiele für deren Implementierung. Darüber hinaus besteht die Möglichkeit, den eigenen Code direkt über einen auf der Webseite integrierten Emulator zu testen und anzupassen.

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie Ausbildungsschwerpunkt: Medientechnik

3.4.2.5. Fazit

Basierend auf der erforderlichen Einarbeitungszeit und den bereits vorhandenen Kenntnissen stellt React Native die optimale Basis für die Nutzung bei Jugendlichen des Krems dar.

Ein zusätzlicher Vorteil von React Native liegt in der gut strukturierten Dokumentation sowie der leicht verständlichen Syntax.

3.4.3. Expo oder klassisches React Native

3.4.3.1. Expo

beschleunigte Methode zur Entwicklung von mobilen Anwendungen für iOS und Android bereitstellt. Expo stellt zahlreiche vordefinierte Komponenten und Bibliotheken zur Verfügung, die Entwicklern dabei helfen, Apps rascher zu entwickeln, ohne sich mit einigen der komplexeren Aspekte von React Native befassen zu müssen. Expo ermöglicht zudem einen einfachen und schnellen Entwicklungszyklus, da die App direkt auf einem Gerät mit der Expo-Anwendung laufen kann, ohne dass sie auf einem Simulator oder einem physischen Gerät installiert werden muss. [25]

Expo bietet auch die Möglichkeit, anstelle von klassischem Javascript, ein Projekt mit Typescript zu verwenden.

Typescript ist eine von Microsoft entwickelte, kostenlose und quelloffene Programmiersprache, die als Superset-Sprache von Javascript fungiert. Dies bedeutet, dass sie auf Javascript aufbaut und zusätzliche Funktionen und Vorteile bietet. Im Grunde genommen erweitert Typescript die Javascript-Funktionalität durch Hinzufügen von statischer Typisierung, Klassen, Schnittstellen, Generika und weiteren Funktionen. Typescript ermöglicht es Entwicklern, die Fehlererkennung während der Kompilierung zu verbessern, indem sie Typen für Variablen, Funktionen und Parameter deklarieren, bevor sie den Code ausführen. Dies trägt zur Verbesserung der Codequalität und -wartbarkeit bei und reduziert das Risiko von Fehlern und Bugs. [26]

3.4.3.2. Klassisches React Native

React Native hingegen ist ein Framework zur Entwicklung von nativen mobilen Anwendungen mit Javascript und React. Es handelt sich um eine solide und flexible Plattform, die Entwicklern die Möglichkeit bietet, native Anwendungen für iOS und Android zu erstellen, die auf die spezifischen Funktionen und APIs jeder Plattform zugreifen können. React Native ermöglicht den Entwicklern auch eine größere Kontrolle über ihre Anwendung und bietet die Möglichkeit, die Anwendung vollständig anzupassen, um eine optimale Benutzererfahrung zu erzielen.

Selbstverständlich kann bei einem Projekt, das auf klassischem React Native basiert, anstelle von Javascript auch Typescript eingesetzt werden.

3.4.3.3. Vergleich

Im Vergleich zu React Native bietet Expo eine eingeschränktere Kontrolle über die Anwendung, da Entwickler auf die vordefinierten Komponenten und Bibliotheken von Expo angewiesen sind. Die Nutzung von klassischem React Native erfordert jedoch mehr Wissen und Erfahrung, um effektiv eingesetzt zu werden.

3.4.3.4. Typescript oder Javascript

Der Hauptunterschied zwischen Typescript und herkömmlichem Javascript liegt in der statischen Typisierung. Dadurch können Typen für Variablen, Funktionen und Parameter vor der Codeausführung deklariert werden. Diese Vorgehensweise verbessert die Codequalität und -wartbarkeit, indem sie die Fehlererkennung während der Kompilierung optimiert und das Auftreten von Fehlern und Bugs verringert.

Im Folgenden wird ein einfaches Javascript-Programm vorgestellt, das eine Funktion namens 'greet' definiert, welche einen Namen als Parameter erwartet und eine Begrüßungsnachricht zurückgibt. Anschließend wird eine Variable 'result' definiert, die das Ergebnis der 'greet'-Funktion enthält, wenn sie mit dem Namen 'John' aufgerufen wird. Schließlich wird das Ergebnis in der Konsole ausgegeben.

```
1 function greet(name) {  
2   return 'Hello ,␣' + name;  
3 }  
4  
5 let result = greet('John');  
6 console.log(result);
```

Quellcode 3.4: Beispiel eines Javascript Codes

Der nachfolgende Code ist in Typescript verfasst und entspricht dem zuvor beschriebenen Javascript-Programm. Beachten Sie, dass die 'greet'-Funktion mit dem Typ 'string' annotiert wurde, um darauf hinzuweisen, dass sie einen String als Parameter erwartet und einen String als Ergebnis zurückgibt. Ebenso wurde die Variable 'result' mit dem Typ 'string' annotiert, um anzugeben, dass sie das Ergebnis der 'greet'-Funktion als String beinhaltet.

```
1 function greet(name: string): string {  
2   return 'Hello ,␣' + name;  
3 }  
4  
5 let result: string = greet('John');  
6 console.log(result);
```

Quellcode 3.5: Beispiel eines Typescript Codes

3.4.3.5. Fazit

Da Expo insbesondere für Einsteiger empfohlen wird und den Entwicklungsprozess durch die Expo App erleichtert, erscheint es für Jugendliches Krems als die passende Wahl. Zudem

wird statt des herkömmlichen Javascripts Typescript eingesetzt, um den Code übersichtlicher zu gestalten und potenzielle Fehler bereits vor der Ausführung zu identifizieren.

3.5. Technologieauswahl für mobile Kommunikation mit Laravel API - SPIN

Um die Unterschiede in Syntax und Komplexität der verschiedenen Bibliotheken zu untersuchen, wurden die vier häufigsten HTTP-Methoden verglichen. Diese Methoden sind:

- GET: Eine Anfrage zum Abrufen von Informationen von einem Server.
- POST: Eine Anfrage zum Senden von Informationen an einen Server, um neue Ressourcen zu erstellen.
- PUT: Eine Anfrage zum Aktualisieren einer vorhandenen Ressource auf einem Server.
- DELETE: Eine Anfrage zum Löschen einer vorhandenen Ressource auf einem Server.

Durch den Vergleich dieser Methoden können wir besser verstehen, wie die verschiedenen Bibliotheken in Bezug auf ihre Syntax und Komplexität bei der Implementierung dieser HTTP-Methoden abschneiden.

3.5.1. React Native Bibliotheken

3.5.1.1. XMLHttpRequest API

Die XMLHttpRequest-API ist eine in Browsern und React Native verfügbare API, die den Datenaustausch zwischen einem Client und einem Server ermöglicht. Sie unterstützt das Senden von HTTP-Anfragen und das Empfangen von Antworten in Form asynchroner Ereignisse. [27] [28]

GET-Methode

Im folgenden Beispiel wird eine GET-Anfrage an die URL 'https://example.com/api/data' gesendet. Die 'open()-Methode initialisiert die Anfrage, während die 'onload'-Eigenschaft auf eine Callback-Funktion gesetzt wird, die ausgeführt wird, sobald die Antwort vom Server eintrifft. Anschließend wird die Antwort in ein JSON-Objekt umgewandelt und auf der Konsole angezeigt.

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('GET', 'https://example.com/api/data');
3 xhr.onload = () => {
4   if (xhr.status === 200) {
5     const data = JSON.parse(xhr.responseText);
6     console.log(data);
7   } else {
8     console.error('Error:', xhr.statusText);
9   }
10 };
11 xhr.onerror = () => console.error('Error:', xhr.statusText);
```

```
12 xhr.send();
```

Quellcode 3.6: Beispiel einer GET-Anfrage mit XMLHttpRequest

POST-Methode

In einem weiteren Beispiel wird eine POST-Anfrage an die URL `https://example.com/api/data` gesendet. Die `open()`-Methode initialisiert die Anfrage, und die `setRequestHeader()`-Methode wird verwendet, um den `Content-Type`-Header auf `application/json` zu setzen. Die Anfragedaten werden im `send()`-Aufruf als JSON-Objekt übermittelt.

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('POST', 'https://example.com/api/data');
3 xhr.setRequestHeader('Content-Type', 'application/json');
4 xhr.onload = () => {
5   if (xhr.status === 200) {
6     const data = JSON.parse(xhr.responseText);
7     console.log(data);
8   } else {
9     console.error('Error:', xhr.statusText);
10  }
11 };
12 xhr.onerror = () => console.error('Error:', xhr.statusText);
13 const requestData = { name: 'John', age: 30 };
14 xhr.send(JSON.stringify(requestData));
```

Quellcode 3.7: Beispiel einer POST-Anfrage mit XMLHttpRequest

PUT-Methode

In einem dritten Beispiel wird eine PUT-Anfrage an die URL `https://example.com/api/data/123` gesendet. Die `open()`-Methode initialisiert die Anfrage, und die `setRequestHeader()`-Methode wird verwendet, um den `Content-Type`-Header auf `application/json` zu setzen. Die Anfragedaten werden im `send()`-Aufruf als JSON-Objekt übermittelt.

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('PUT', 'https://example.com/api/data/123');
3 xhr.setRequestHeader('Content-Type', 'application/json');
4 xhr.onload = () => {
5   if (xhr.status === 200) {
6     const data = JSON.parse(xhr.responseText);
7     console.log(data);
8   } else {
9     console.error('Error:', xhr.statusText);
10  }
11 };
12 xhr.onerror = () => console.error('Error:', xhr.statusText);
13 const requestData = { name: 'John', age: 31 };
14 xhr.send(JSON.stringify(requestData));
```

Quellcode 3.8: Beispiel einer PUT-Anfrage mit XMLHttpRequest

DELETE-Methode

In einem vierten Beispiel wird eine DELETE-Anfrage an die URL `https://example.com/api/data/123` gesendet. Die `open()`-Methode initialisiert die Anfrage, und im `send()`-Aufruf werden keine Anfragedaten übermittelt.

```

1 const xhr = new XMLHttpRequest();
2 xhr.open('DELETE', 'https://example.com/api/data/123');
3 xhr.onload = () => {
4   if (xhr.status === 200) {
5     console.log('Data deleted');
6   } else {
7     console.error('Error:', xhr.statusText);
8   }
9 };
10 xhr.onerror = () => console.error('Error:', xhr.statusText);
11 xhr.send();

```

Quellcode 3.9: Beispiel einer DELETE-Anfrage mit XMLHttpRequest

3.5.1.2. Fetch API

Sie ermöglicht es Benutzern, HTTP-Anfragen zu senden und zu empfangen. Die Fetch API ist in der Regel unkomplizierter und intuitiver als XMLHttpRequest und bietet eine einfachere Möglichkeit, HTTP-Anfragen zu senden und auf Antworten zu reagieren. [28]

GET-Methode

Im folgenden Beispiel wird eine GET-Anfrage an die URL 'https://example.com/api/data' gesendet. Die Fetch API sendet die Anfrage und gibt ein 'Promise'-Objekt zurück. Der 'then()-Block wird ausgeführt, wenn die Antwort vom Server eintrifft und konvertiert die Antwort in ein JSON-Objekt. Das Ergebnis wird anschließend auf der Konsole angezeigt. Tritt ein Fehler auf, wird der 'catch()-Block ausgeführt und der Fehler auf der Konsole dargestellt.

```

1 fetch('https://example.com/api/data')
2   .then(response => response.json())
3   .then(data => console.log(data))
4   .catch(error => console.error(error));

```

Quellcode 3.10: Beispiel einer GET-Anfrage mit FETCH

POST-Methode

In einem weiteren Beispiel wird eine POST-Anfrage an die URL 'https://example.com/api/data' gesendet. Die 'method'-Option wird auf POST gesetzt, und die 'headers'-Option enthält den 'Content-Type'-Header, der angibt, dass die Daten im JSON-Format übermittelt werden. Die 'body'-Option beinhaltet die zu sendenden Daten als JSON-Objekt, das in einen String konvertiert wird.

```

1 fetch('https://example.com/api/data', {
2   method: 'POST',
3   headers: {
4     'Content-Type': 'application/json'
5   },
6   body: JSON.stringify({ name: 'John', age: 30 })
7 })
8 .then(response => response.json())

```

```
9 .then(data => console.log(data))  
10 .catch(error => console.error(error));
```

Quellcode 3.11: Beispiel einer POST-Anfrage mit FETCH

PUT-Methode

In einem dritten Beispiel wird eine PUT-Anfrage an die URL 'https://example.com/api/data/123' gesendet. Die 'method'-Option wird auf PUT gesetzt, und die 'headers'-Option enthält den 'Content-Type'-Header, der angibt, dass die Daten im JSON-Format übermittelt werden. Die 'body'-Option beinhaltet die zu aktualisierenden Daten als JSON-Objekt, das in einen String konvertiert wird.

```
1 fetch('https://example.com/api/data/123', {  
2   method: 'PUT',  
3   headers: {  
4     'Content-Type': 'application/json'  
5   },  
6   body: JSON.stringify({ name: 'John', age: 31 })  
7 })  
8 .then(response => response.json())  
9 .then(data => console.log(data))  
10 .catch(error => console.error(error));
```

Quellcode 3.12: Beispiel einer PUT-Anfrage mit FETCH

DELETE-Methode

In einem vierten Beispiel wird eine DELETE-Anfrage an die URL 'https://example.com/api/data/123' gesendet. Die 'method'-Option wird auf DELETE gesetzt, und die 'headers'-Option enthält den 'Content-Type'-Header, der angibt, dass die Anfrage im JSON-Format übermittelt wird. Im Body werden keine Daten benötigt, da es sich um eine DELETE-Anfrage handelt.

```
1 fetch('https://example.com/api/data/123', {  
2   method: 'DELETE',  
3   headers: {  
4     'Content-Type': 'application/json'  
5   }  
6 })  
7 .then(response => console.log('Data_deleted'))  
8 .catch(error => console.error(error));
```

Quellcode 3.13: Beispiel einer DELETE-Anfrage mit FETCH

3.5.2. Drittanbieter Möglichkeiten

3.5.2.1. Axios

React Native ermöglicht die Integration externer Bibliotheken wie Axios.

„Axios is a simple promise based HTTP client for the browser and node.js. Axios provides a simple to use library in a small package with a very extensible interface.“ [29]

Es bietet zahlreiche Funktionen, darunter automatische Konvertierung von Datenformaten wie JSON und XML, Authentifizierungsunterstützung und Interception-Methoden, die es Entwicklern ermöglichen, Anfragen und Antworten zu bearbeiten und zu verarbeiten.

Axios zeichnet sich im Vergleich zu anderen HTTP-Bibliotheken wie XMLHttpRequest und Fetch durch seine Benutzerfreundlichkeit aus und bietet eine umfangreichere Palette von Funktionen. Dazu gehören automatische Fehlerbehandlung sowie Unterstützung für Promises und async/await. [29] [30]

GET-Methode

In einem Beispiel zur GET-Anfrage wird die URL 'https://example.com/api/data' abgefragt. Die Axios 'get()'-Methode sendet die Anfrage und gibt ein 'Promise'-Objekt zurück. Der 'then()'-Block wird ausgeführt, sobald die Antwort vom Server eintrifft, und zeigt den Inhalt der Antwort auf der Konsole an. Im Falle eines Fehlers wird der 'catch()'-Block ausgeführt und der Fehler auf der Konsole angezeigt.

```
1 axios.get('https://example.com/api/data')
2   .then(response => {
3     console.log(response.data);
4   })
5   .catch(error => {
6     console.error(error);
7   });
```

Quellcode 3.14: Beispiel einer GET-Anfrage mit Axios

POST-Methode

In einem weiteren Beispiel wird eine POST-Anfrage an die URL 'https://example.com/api/data' gesendet. Die Axios 'post()'-Methode sendet die Anfrage und gibt ein 'Promise'-Objekt zurück. Die Anfragedaten werden als JSON-Objekt im zweiten Parameter der 'post()'-Methode übergeben. Der 'then()'-Block wird ausgeführt, sobald die Antwort vom Server eintrifft, und zeigt den Inhalt der Antwort auf der Konsole an. Im Falle eines Fehlers wird der 'catch()'-Block ausgeführt und der Fehler auf der Konsole angezeigt.

```
1 const requestData = { name: 'John', age: 30 };
2 axios.post('https://example.com/api/data', requestData)
3   .then(response => {
4     console.log(response.data);
5   })
6   .catch(error => {
7     console.error(error);
8   });
```

Quellcode 3.15: Beispiel einer POST-Anfrage mit Axios

PUT-Methode

In einem weiteren Beispiel für eine PUT-Anfrage wird an die URL 'https://example.com/api/data/123' gesendet. Die Axios 'put()'-Methode sendet die Anfrage und gibt ein 'Promise'-Objekt zurück. Die Anfragedaten werden als JSON-Objekt im zweiten Parameter der 'put()'-Methode übergeben. Der 'then()'-Block wird ausgeführt, sobald die Antwort vom Server eintrifft, und

zeigt den Inhalt der Antwort auf der Konsole an. Im Falle eines Fehlers wird der 'catch()'-Block ausgeführt und der Fehler auf der Konsole angezeigt.

```
1 const requestData = { name: 'John', age: 31 };
2 axios.put('https://example.com/api/data/123', requestData)
3   .then(response => {
4     console.log(response.data);
5   })
6   .catch(error => {
7     console.error(error);
8   });
```

Quellcode 3.16: Beispiel einer PUT-Anfrage mit Axios

DELETE-Methode

In einem Beispiel zur DELETE-Anfrage wird die URL 'https://example.com/api/data/123' abgefragt. Die Axios 'delete()'-Methode sendet die Anfrage und gibt ein 'Promise'-Objekt zurück. Der 'then()'-Block wird ausgeführt, sobald die Antwort vom Server eintrifft, und zeigt "Data deleted" auf der Konsole an. Im Falle eines Fehlers wird der 'catch()'-Block ausgeführt und der Fehler auf der Konsole angezeigt.

```
1 axios.delete('https://example.com/api/data/123')
2   .then(response => {
3     console.log('Data deleted');
4   })
5   .catch(error => {
6     console.error(error);
7   });
```

Quellcode 3.17: Beispiel einer DELETE-Anfrage mit Axios

3.5.3. Fazit

Da React Native die Integration externer Bibliotheken ermöglicht, wäre es für Jugendliche Krems ratsam, Axios zu verwenden.

Axios bietet dieselben Funktionalitäten wie Fetch und XMLHttpRequest, ergänzt jedoch um eine ausgezeichnete Dokumentation und eine leicht verständliche Syntax.

4. Dokumentation der Implementierung

4.1. Datenstruktur - KAUF

4.1.1. Datenbankschema

Die Verwendung des korrekten Datenschemas ist essentiell für unsere Applikation.

4.1.2. Kriterien

Die Wahl der Aspekte unserer Applikation sind stark von folgenden Kriterien abhängig:

4.1.2.1. Datenintegrität

Um Dateninkonsistenzen und fehlerhafte Datensätze zu vermeiden, ist es zu empfehlen, inkorrekte Eingaben schon von Beginn an nicht zuzulassen. Dies kann durch ein korrektes Datenschema realisiert werden. Das Datenschema sollte hierbei mit der Datenüberprüfung der Applikation übereinstimmen. Zum Beispiel: Wenn in der Applikation ein neuer Nutzer registriert wird, soll die Applikation überprüfen, ob eingegebene Daten wie der Nutzernamen nicht länger sind, als es das Datenschema maximal erlaubt. Erst dann sollte eine SQL Query in der Datenbank ausgeführt werden. Die Applikation übernimmt dann zudem die Aufgabe, Datenmuster, dessen Definition im Datenbankschema nicht möglich ist, vorzulegen. Zum Beispiel: Das Passwort muss mindestens einen Großbuchstaben enthalten.

4.1.2.2. Leistung

Durch ein effizientes Datenbankschema können SQL Queries schneller ausgeführt werden, dies wiederum reduziert Ladezeiten für den Nutzer und erhöht die Benutzerfreundlichkeit.

4.1.2.3. Wartbarkeit

Ein gutes Datenbankschema sollte so definiert sein, dass im Nachhinein effizient Änderungen vorgenommen oder die Datenbank um neue Tabellen ergänzt werden kann, ohne bereits existierende Tabellen zu verändern.

4.1.2.4. ER Diagramm

Das Datenbankschema für Jungdliches Krems ist in der nächsten Abbildung zu erkennen.

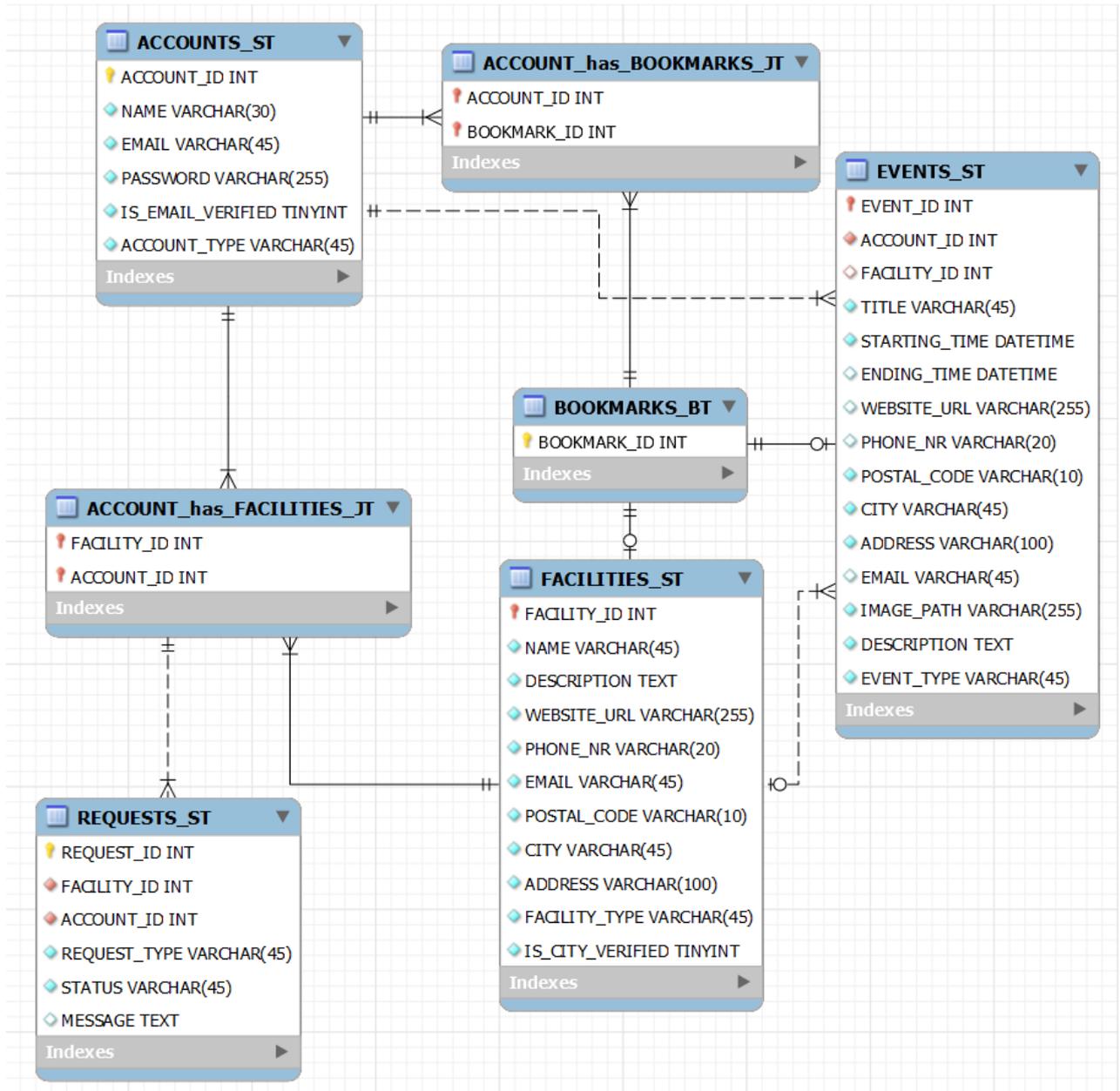


Abbildung 4.1.: ER Diagramm

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie Ausbildungsschwerpunkt: Medientechnik

4.1.2.5. Erklärung der Tabellen

BOOKMARKS_BT

In dieser Tabelle wird ein Primary Key gespeichert, der später von FACILITIES_ST und EVENTS_ST verwendet wird.

Datenfelder: BOOKMARK_ID

ACCOUNTS_ST

In dieser Tabelle werden Nutzerdaten und ein Primary Key zur Identifikation gespeichert:

Datenfelder: ACCOUNT_ID, NAME, EMAIL, PASSWORD, IS_EMAIL_VERIFIED, ACCOUNT_TYPE

FACILITIES_ST

In dieser Tabelle werden die Daten von einer Bildungsanstalt und ein Primary Key zur Identifikation gespeichert:

Datenfelder: FACILITY_ID, NAME, DESCRIPTION, WEBSITE_URL, PHONE_NR, EMAIL, POSTAL_CODE, CITY, ADDRESS, FACILITY_TYPE, IS_CITY_VERIFIED

EVENTS_ST

In dieser Tabelle werden die Daten von einem Event und ein Primary Key zur Identifikation gespeichert. Zusätzlich wird die ID von dem Account gespeichert, welcher jenes Event angelegt hat. Falls das Event ein Schul-Event ist, wird außerdem die ID der Schule gespeichert, welche jenem Event zuzuordnen ist.

Datenfelder: EVENT_ID, ACCOUNT_ID, FACILITY_ID, TITLE, STARTING_TIME, ENDING_TIME, WEBSITE_URL, PHONE_NR, POSTAL_CODE, CITY, ADDRESS, EMAIL, IMAGE_PATH, DESCRIPTION, EVENT_TYPE

ACCOUNT_has_FACILITIES_JT

In dieser Tabelle wird gespeichert, welcher Account welche Bildungsanstalt verwalten darf.

Datenfelder: FACILITY_ID, ACCOUNT_ID

ACCOUNT_has_BOOKMARKS_JT

In dieser Tabelle wird gespeichert, welcher Account welche Lesezeichen gespeichert hat.

Datenfelder: ACCOUNT_ID, BOOKMARK_ID

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie Ausbildungsschwerpunkt: Medientechnik

REQUESTS_ST

In dieser Tabelle werden die Daten der Anträge und ein Primary Key gespeichert. Darüber hinaus wird die ID des Antragstellers und die ID der Bildungsanstalt, für welche er den Antrag stellt, gespeichert.

Datenfelder: REQUEST_ID, FACILITY_ID, ACCOUNT_ID, REQUEST_TYPE, STATUS, MESSAGE

4.1.2.6. Besondere Merkmale

1:1 zwischen BOOKMARKS_BT und EVENTS_ST/FACILITIES_ST

Ein Account kann sowohl Events als auch Bildungsanstalten mit einem Lesezeichen versehen. Da es aber unübersichtlich wäre, für Events und Bildungsanstalten eine separate n:m Tabelle zu erstellen, wird eine Tabelle "BOOKMARKS_BT" implementiert, welche Primary Keys für "EVENTS_ST" und "FACILITIES_ST" zur Verfügung stellt. "ACCOUNTS_ST" hat dann eine n:m Verbindung über "ACCOUNT_has_BOOKMARKS_JT" mit dieser Tabelle. Diese Verbindung stellt die Lesezeichen von einem User dar.

1:1 zwischen ACCOUNT_HAS_FACILITIES_JT und REQUESTS_ST

Da in Anträgen sowohl die ID des Antragstellers als auch die ID der Bildungsanstalt, für welche der Antrag gestellt wurde, gespeichert wird, besteht zwischen "REQUESTS_ST" und "ACCOUNT_has_FACILITIES_JT" eine 1:n Verbindung. Durch die Wiederverwendung der "ACCOUNT_has_FACILITIES_JT" Tabelle verringert sich die Anzahl der Verbindungen im Datenschema und die Komplexität wird vermindert.

4.1.3. Laravel Models

In Laravel ist ein Model eine PHP-Klasse, die mit einer bestimmten Tabelle der Datenbank verknüpft ist.

"In Laravel enthält das Model die logische Struktur (Schema) und die Beziehungen (Relations) der dahinterliegenden Datenressourcen. Bei Laravel hat jede Datenbank Tabelle ein Model mit dem es mit der Anwendung kommuniziert. Also lesen, schreiben, updaten, löschen wird über das Model erst ermöglicht und verwaltet." [31] (Stand 22.03.2023)

Man kann Models als eine Abstraktionsschicht über der Datenbank sehen. Das sorgt dafür, dass Entwickler nicht direkt mit der Datenbank arbeiten müssen, und heißt ORM (Object Relational Mapping). Laravel nennt sein ORM Eloquent und jedes Model ist automatisch Teil von Eloquent. Es ist daher nicht wie bei vielen anderen Frameworks nötig, das ORM extra einzurichten, da es schon standardmäßig konfiguriert ist.

Man kann mit Hilfe von artisan ein neues Model über die Kommandozeile erstellen:

```
1 php artisan make:model [name]
```

Nach Ausführung von diesem Code, wird in `App\Models` eine neue Datei, welche eine Vorlage für ein Model repräsentiert, erstellt. Diese Vorlage sieht folgendermaßen aus:

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Book extends Model
9 {
10     use HasFactory;
11 }
```

Quellcode 4.1: Model-Template

Im nächsten Schritt der Entwicklung ist es nötig, ein Model für jede Tabelle, welche in der Datenbank laut dem ER Diagramm definiert ist, zu erstellen.

In diesem Model können nun verschiedene Variablen definiert werden, um das Verhalten des Models anzupassen.

- `protected $table`: Definiert den Tabellennamen
- `protected $primaryKey`: Definiert den primären Schlüssel
- `protected $foreignKey`: Definiert einen Foreign Key
- `public $incrementing`: Definiert ob der Primary Key automatisch inkrementiert, `true` oder `false`
- `public $timestamps`: definiert ob die SQL-Spalten `created_at` und `updated_at` der Tabelle hinzugefügt werden sollen, `true` oder `false`
- `protected $fillable`: Alle Spalten, für die ein mass assignment möglich ist
- `protected $casts`: Alle Spalten mit einem speziellen Datentyp und ihr Datentyp
- `protected $hidden`: Alle Spalten die nicht mitgeliefert werden wenn das Model in ein Array oder JSON-Objekt umgewandelt wird

Darüber hinaus können in Models noch Funktionen definiert werden, welche später durch das Model aufgerufen werden können.

Das Model, welches die Tabelle `'FACILITIES_ST'` repräsentiert ist als Beispiel für ein fertiges Model durch den folgenden Code beschrieben:

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Facility extends Model
```

```
9 {
10     use HasFactory;
11
12     protected $table = 'FACILITIES_ST';
13     protected $primaryKey = 'FACILITY_ID';
14
15     public $timestamps = true;
16     public $incrementing = false;
17
18     protected $fillable = [
19         'NAME', 'DESCRIPTION', 'WEBSITE_URL', 'PHONE_NR',
20         'EMAIL', 'POSTAL_CODE', 'CITY', 'ADDRESS', 'IS_CITY_VERIFIED', '
FACILITY_TYPE', 'IMAGE_PATH', 'FACILITY_ID'
21     ];
22
23     protected $casts = [
24         'IS_CITY_VERIFIED' => 'boolean',
25     ];
26
27     public function managers()
28     {
29         return $this->belongsToMany(Account::class, '
ACCOUNT_has_FACILITIES_JT', 'FACILITY_ID', 'ACCOUNT_ID')->select(['
ACCOUNTS_ST.ACCOUNT_ID', 'ACCOUNTS_ST.NAME', 'ACCOUNTS_ST.EMAIL']);
30     }
31
32     public function events()
33     {
34         return $this->hasMany(Event::class, 'FACILITY_ID', 'FACILITY_ID')
->with(['account', 'facility']);
35     }
36 }
```

Quellcode 4.2: Facility.php

4.1.4. Laravel Migrations

Während Models in Laravel die Abbildung einer Datenbanktabelle in einer Klasse definieren, repräsentieren Migrations eine exakte Kopie des Datenbankschemas

Während Beziehungen zu anderen Klassen im Model mit Funktionen beschrieben werden, werden in einer Migration dafür Foreign-Key-Spalten benutzt, welche den Primary Key einer Tabelle, zu der eine Beziehung definiert werden soll, beinhalten. Ein Model kann von der tatsächlichen Datenbankstruktur abweichen und zusätzliche Funktionen beinhalten, eine Migration ist jedoch eine 1:1 Abbildung der Datenbank und verantwortlich, um das Schema jener zu verändern.

Es ist also nicht redundant, die Struktur einmal in einem Model und einmal in einer Migration zu definieren, da beide Sachen einen unterschiedlichen Zweck haben.

Um Migrationen zu definieren, wird in Laravel der Schema Builder verwendet.

“The Laravel Schema class provides a database agnostic way of manipulating tables. It works well with all of the databases supported by Laravel, and has a unified API across all of these systems.” [32] (Stand 22.03.2023)

Die Migration für das Model „Facilities“ ist als Beispiel für eine fertige Migration durch den folgenden Code beschrieben:

```

1 <?php
2 use Illuminate\Database\Migrations\Migration;
3 use Illuminate\Database\Schema\Blueprint;
4 use Illuminate\Support\Facades\Schema;
5
6 return new class extends Migration
7 {
8     public function up()
9     {
10         Schema::create('FACILITIES_ST', function (Blueprint $table) {
11             $table->foreignId("FACILITY_ID")->primary()->constrained('
BOOKMARKS_BT', 'BOOKMARK_ID')->onDelete('cascade');
12             $table->string('NAME')->unique();
13             $table->text('DESCRIPTION');
14             $table->string('WEBSITE_URL')->unique();
15             $table->string('PHONE_NR')->unique();
16             $table->string('EMAIL')->unique();
17             $table->string('POSTAL_CODE');
18             $table->string('CITY');
19             $table->string('ADDRESS');
20             $table->boolean('IS_CITY_VERIFIED')->default(false);
21             $table->string('FACILITY_TYPE');
22             $table->string('IMAGE_PATH');
23             $table->timestamps();
24         });
25     }
26     public function down()
27     {
28         Schema::dropIfExists('FACILITIES_ST');
29     }
30 };

```

Quellcode 4.3: Facility migration

Um die Migrationen auszuführen und damit die Datenbank zu aktualisieren, wird artisan verwendet:

```
1 php artisan migrate
```

Alle Migrationen werden in einer PHP Datei gespeichert. Dem Namen dieser Datei ist immer ein Zeitstempel vorangestellt. Durch den Vergleich der Zeitstempel ermittelt Laravel die Reihenfolge, in der die Migrationen ausgeführt werden. Aus diesem Grund ist es wichtig, die Migrationen in der richtigen Reihenfolge zu erstellen oder im Nachhinein die Zeitstempel anzupassen, damit es zu keinem Fehler in der Ausführung kommt. Falls zum Beispiel Migrationen für eine Student-Tabelle und eine School-Tabelle und eine 1:n Relation zwischen diesen Tabellen erstellt werden möchte, muss die Tabelle ohne Foreign-Key - in diesem Fall

School - zuerst erstellt werden. Falls eine Migration im Nachhinein verändert oder gelöscht und eine neue neue Migration erstellt wurde, diese aber aufgrund von dem bereits existierenden Datenbank-Schema nicht ausgeführt werden kann, gibt es die Option, mit einem Command automatisch alle Migrationen rückgängig zu machen und neu zu migrieren:

```
1 migrate:refresh
```

Über folgenden Befehl kann in Laravel eine neue Migration erstellt werden:

```
1 php artisan make:model [name]
```

Es gibt auch die Entscheidungsmöglichkeit, eine Migration bereits bei der Ausarbeitung des zusammenhängenden Models zu erstellen:

```
1 php artisan make:model [name] -m
```

4.1.5. Datenmanipulation mit Eloquent

Im Normalfall sind Models, die in Laravel erstellt werden, Kindklassen von der Klasse “Model”, eine Klasse welche von Eloquent vordefiniert ist. Diese bietet standardmäßig Funktionen, um Objekte der Klasse zu erstellen, zu ändern und zu löschen.

Wenn zum Beispiel ein Model “School” erstellt wurde, dann kann eine neue Instanz dieses Models mit der vordefinierten Methode “create” angelegt werden. Als Eingabewert benötigt es standardmäßig ein Array, in dem alle Variablen, welche auch in Model definiert worden sind, die weder automatisch inkrementiert werden oder einen Standardwert haben, enthalten sind. Dies schaut beispielhaft folgendermaßen aus:

```
1 School::create([
2     'NAME' => 'HTL_Krems',
3     'DESCRIPTION' => 'Die_HTL_Krems_ist_eine_BHS_in_Krems.'
4 ]);
```

Quellcode 4.4: Aufruf der Create-Methode von School

Soll diese Methode jedoch geändert, ist dies möglich, indem eine Methode mit dem gleichen Namen im Model von “School” definiert wird. Ist es ein Wunsch bzw. Ziel zum Beispiel bei jedem Aufruf der “create”-Methode von “School” automatisch einen Direktor mit dem Namen “Max Mustermann” anzulegen, dann sieht dies folgendermaßen aus:

```
1 public static function create(array $data)
2 {
3     Principle::create([
4         'NAME' => 'Max_Mustermann'
5     ]);
6
7     $school = new School();
8     $school->NAME = $data['NAME'];
```

```

9     $school->'DESCRIPTION' = $data[ 'DESCRIPTION' ];
10    $school->save();
11    return $school;
12 }

```

Quellcode 4.5: Überschreiben der Funktionalität der Create-Methode von School

Alle diese vordefinierten Methoden finden sich in der “Model” Basisklasse und den zahlreichen Klassen, welche implementiert werden.

4.2. HTTP Routing - KAUF

“Das Routing entscheidet, welche Aktion bei welcher Url ausgeführt wird. Im einfachsten Fall wird eine View angezeigt. Dabei lassen sich unter anderem für GET und POST verschiedene Aktionen zuweisen. Auch Variablen in der Url lassen sich einfach nutzen: Sie werden dann zum Beispiel automatisch als Parameter in einer Methode übergeben. Natürlich ist es auch möglich komplexere Routing-Regeln festzulegen. Dafür lassen sich reguläre Ausdrücke einsetzen.” [33] (Stand 22.03.2023)

4.2.1. Definition

Die Definition der Routen in Laravel besteht aus drei Teilen:

4.2.1.1. HTTP-Methode

Die HTTP-Methode definiert, um welche Art von Aktion es sich handelt. Für die Webapplikation fanden folgende Methoden Verwendung:

- GET: Für den Abruf von Ressourcen
- POST: Um neue Ressourcen zu erstellen
- PUT: Um vorhandene Ressourcen zu aktualisieren
- DELETE: Um Ressourcen zu löschen

Die Implementation der HTTP-Methoden muss jedoch technologisch gesehen keinen bestimmten Mustern folgen. Für die API ist es sinnvoll, für Routes, welche für das Löschen von Objekten verantwortlich sind, die HTTP-Methoden DELETE zu erwarten. Für API-Calls, welche im Frontend der Webseite abspielen, ist dies jedoch nicht nötig, da jene für Endnutzer nicht einsehbar sind und nicht in Verbindung mit anderen Entwicklern stehen, welche Muster und einen standardisierten Syntax wie bei der Arbeit mit einer API erwarten würden.

Es hat darüber hinaus einige Vorteile, die HTTP-Methoden für Routes, die nicht für die API definiert wurden, auf GET und POST einzuschränken:

- Gleicher Syntax: Es ist nicht notwendig, sich um Unterschiede zwischen dem Aufruf von z.B. Routen, die für Delete-Operationen und Routen, die für andere Operationen verantwortlich sind, zu kümmern, und man kann immer den gleichen Syntax für den Aufruf von Routes verwenden.
- Kompatibilität: Manche Tools haben keine oder nur eingeschränkte Funktionalität für Requests mit anderen Methoden als GET oder POST. Durch das ausschließliche verwenden von GET und POST wird sichergestellt, dass es zu keinen Problemen in der Kompatibilität kommt.

Für die Webapplikation wurden für die API die Methoden GET, POST, DELETE und PUT verwenden. In den Svelte-Seiten kamen jedoch ausschließlich POST-Methoden zum Einsatz. Die Webseite verwendet also nur die HTTP-Methoden GET und POST.

4.2.1.2. URL

Hier wird definiert, bei welcher URL die Aktion ausgeführt wird. Es ist auch möglich, in der URL Parameter zu definieren. Jene müssen dann in geschwungene Klammern gesetzt werden. Diese werden beim Aufruf der Aktion jener automatisch als Argument übergeben.

4.2.1.3. Aktion

Hier wird definiert, welche Aktion beim Aufruf der Route ausgeführt werden soll. Die Aktion kann entweder eine Funktion von einem Controller sein, oder direkt in der Route definiert werden.

4.2.1.4. Namensgebung

Es ist möglich, einer Route mithilfe von “->name()” einen Namen zuzuweisen. Dies ermöglicht, dass die Route auch durch diesen Namen angesprochen werden kann.

Manche Tools in Laravel wie zum Beispiel dem Tool “CanResetPassword”, welches Funktionalität für das Zurücksetzen von Passwörtern bietet, rufen automatisch Routes ab, so zum Beispiel “password.reset”. In diesem Fall ist es nötig, der Route, welche dafür verantwortlich ist, die Seite, welche verwendet wird um das Passwort zurückzusetzen, zurückzugeben, den Namen “password.reset” zuzuweisen:

```
1 Route::inertia('/resetpassword', 'PasswordReset')->name('password.reset');
```

4.2.2. Routing-Dateien

Routen werden in Laravel in dem Ordner `/routes` gespeichert. Einige Dateien für das Speichern von Routen sind bereits vordefiniert. [33] (Stand 22.03.2023)

- `api.php`: Routen für REST APIs
- `channels.php`: Definition von Broadcasting-Channels
- `console.php`: Definition von Artisan-Befehlen
- `web.php`: Reguläres HTTP-Routing

Für die Webapplikation sind die Dateien `api.php` und `web.php` interessant.

Jedoch wurde noch eine weitere Routing-Datei hinzugefügt: `render.php`. Je nach Größe der Applikation, wird es schnell unübersichtlich, alle Web-Routes in einer Datei zu definieren. Um dies zu vermeiden, gibt es die Möglichkeit, mehrere Dateien für Routes zu erstellen und jene dann in `web.php` einzubinden. Angesichts der Größe der Webapplikation wurden alle Routes für das Rendering der Webseite, also alle Routen für die Webseite mit der HTTP-Methode GET `render.php` definiert und alle Routes für die Logik der Webseite, also Routes mit der HTTP-Methode POST in `web.php` definiert. `render.php` wurde dann in `web.php` eingebunden

4.2.3. Inertia Funktionen für Routen

Inertia stellt Entwicklern für die Definition des HTTP Routing Funktionalität zur Verfügung, um bei dem Aufruf von Routes Inertia-Seiten zurückzugeben, und eine Abkürzung, um Routen zu definieren, welche eine Inertia-Seite zurückgeben und dabei keine zusätzlichen Eingabeparameter benötigen.

Um eine als Aktion einer Route die Rückgabe einer Inertia-Seite zu definieren, verlangt es nach folgendem Code:

```
1 Route::get('/dashboard/accounts', function () {
2     return Inertia::render('Dashboard/Accounts', [
3         'accounts' => Account::all()
4     ]);
5 });
```

Quellcode 4.6: Route gibt Inertia-Seite mit Eingabeparametern zurück

Um als Aktion einer Route die Rückgabe einer Inertia-Seite ohne zusätzliche Parameter zu definieren, wird folgende Abkürzung verwendet:

```
1 Route::inertia('/bookmarks', 'Bookmarks');
```

4.2.4. Controller

Um die Aktionen von Routes nicht jedes Mal in den Routing-Dateien definieren zu müssen, ist es ein Best Practice, die Logik von Aktionen von Routes in Controllern zu definieren. Controller werden in dem Ordner `App\Http\Controllers` gespeichert.

Außerdem besteht die Möglichkeit, die Funktionalität der Aktionen von Routes für die API und die Webseite in separaten Controllern zu definieren. Diese Controller werden in der Regel in den Ordnern `'App\Http\Controllers\Api'` und `'App\Http\Controllers'` gespeichert.

Innerhalb dieser Controller ist dann im Regelfall die gesamte Funktionalität des Backends, also alle Änderungen an den Modellen, welche durch den Aufruf von Aktionen in Routes passieren, und die Rückgabewerte der Routes, implementiert.

Wird in der Routing-Datei folgende Route definiert:

```
1 Route::post('/logout', [AccountController::class, 'logout']);
```

Dann muss in der Klasse `AccountController` eine Funktion "logout" definiert werden:

```
1 public function logout()  
2 {  
3     Auth::logout();  
4 }
```

Quellcode 4.7: Logout-Funktion im Account-Controller

4.2.5. Laravel Traits

Da API-Controller und Web-Controller oft Funktionalität teilen, wäre es ineffizient, die Funktionen, die in Controllern verwendet werden, ausschließlich in den Controllern zu definieren. Es ist ein Best Practice, diese geteilte Funktionalität in sogenannte Traits auszulagern. Traits werden in Laravel standardmäßig im Ordner 'App\Http\Traits' gespeichert. Durch die großen Funktionen, welche wir für die Validierung von Daten einsetzen, empfiehlt es sich besonders, diese in den Traits zu definieren.

Eine Trait kann so aussehen:

```

1 <?php
2
3 namespace App\Http\Traits;
4 use Illuminate\Http\Request;
5
6 trait FacilityTrait
7 {
8     public function validateRequest(Request $request)
9     {
10         ...
11     }
12 }
```

Quellcode 4.8: Struktur eines Traits

Um diese Trait in einem Controller zu benutzen muss sie zuerst mithilfe eines use-Statements definiert werden:

```

1 use App\Http\Traits\FacilityTrait;
```

Danach muss in der Klassendefinition ein weiteres use-Statement definiert werden:

```

1 class FacilityController extends Controller
2 {
3     use FacilityTrait;
4     ...
5 }
```

Quellcode 4.9: Einbindung eines Traits in einen Controller

Um dann die Funktionen des Traits auszuführen, kann dies wie folgt durchgeführt werden:

```

1 public function createFacility()
2 {
3     $this->validateRequest(request());
4     ...
5 }
```

Quellcode 4.10: Aufruf von Trait-Funktionalität innerhalb eines Controllers

4.2.6. Laravel Middleware

Eine Möglichkeit, um sicherzustellen, dass Nutzer nur Aktionen durchführen können, für die sie die benötigten Berechtigungen haben, ist, die Authentifizierung der Nutzer in jeder Route eigens zu überprüfen. Dies ist jedoch, vor allem bei vielen Routes, ineffizient. Abhilfe hierbei bietet Middleware.

“Middleware bietet einen praktischen Mechanismus zum Filtern von HTTP-Anfragen, die in Ihre Anwendung eingehen. Laravel enthält zum Beispiel eine Middleware, die überprüft, ob der Benutzer Ihrer Anwendung authentifiziert ist. Wenn der Benutzer nicht authentifiziert ist, leitet die Middleware den Benutzer zum Anmeldebildschirm um. Ist der Benutzer jedoch authentifiziert, lässt die Middleware zu, dass die Anfrage weiter in die Anwendung gelangt.” [34] (Stand 22.03.2023)

Um für eine bessere Strukturierung zu sorgen, ist es sinnvoll, Middleware in die Ordner “Api” und “Web” zu unterteilen.

Bevor Middleware verwendet wird, muss diese in der Datei “App/Http/Kernel.php” registriert werden. In dieser Datei gibt es mehrere Variablen, welche Middleware in verschiedenen Ansätzen registriert:

- `protected $middleware`: Middleware, die bei jeder Anfrage an die Applikation ausgeführt wird.

```
1 protected $middleware = [  
2     \App\Http\Middleware\TrustProxies::class,  
3     ...  
4 ];
```

Quellcode 4.11: `$middleware` in `app/Http/Kernel.php`

- `protected $middlewareGroups`: Gruppen, welcher verschiedene Middleware zugewiesen werden kann. Diese Gruppen können dann Routen zugewiesen werden, anstatt ihnen jede Middleware einzeln zuzuweisen.

```
1 protected $middlewareGroups = [  
2     'web' => [  
3         \App\Http\Middleware\EncryptCookies::class,  
4         ...  
5     ],  
6  
7     'api' => [  
8         'forceJsonApi',  
9         ...  
10    ],  
11 ];
```

Quellcode 4.12: `$middlewareGroups` in `app/Http/Kernel.php`

- protected \$routeMiddleware: Einzelne Middleware. Diese können dann Gruppen oder Routen zugewiesen werden.

```
1 protected $routeMiddleware = [  
2     'auth' => \App\Http\Middleware\Authenticate::class,  
3     ...  
4 ];
```

Quellcode 4.13: \$routeMiddleware in app/Http/Kernel.php

Innerhalb einer Middleware muss eine Funktion “handle” definiert werden. Diese wird beim Aufruf der Middleware ausgeführt. Eine Middleware schaut dann folgendermaßen aus:

```
1 <?php  
2  
3 namespace App\Http\Middleware\Web;  
4  
5 use App\Models\AccountHasBookmarks;  
6 use Closure;  
7 use Illuminate\Http\Request;  
8  
9 class BookmarkOwner  
10 {  
11     public function handle($request, Closure $next)  
12     {  
13         $bookmarkId = $request->route('id');  
14  
15         if($request->user()->bookmarks->contains('BOOKMARK_ID',  
16 $bookmarkId) || $request->user()->ACCOUNT_TYPE == 'Systemverwalter')  
17             return $next($request);  
18         return abort(403, 'Sie_sind_nicht_der_Besitzer_dieses_Lesezeichens  
19 .');  
20     }  
21 }
```

Quellcode 4.14: app/Http/Middleware/Web/BookmarkOwner.php

Middleware können auch Argumente übergeben werden. Bei einer Middleware, die überprüft, ob der Accounttyp der aktuellen Nutzers gleich dem Eingabeparameter “\$userType” entspricht, schaut die handle-Methode folgendermaßen aus:

```

1 public function handle($request, Closure $next, $userType)
2     {
3         if (auth()->check() && auth()->user()->ACCOUNT_TYPE === $userType)
4             {
5                 return $next($request);
6             }
7         return abort(403, 'Sie_sind_kein_' . $userType . '.');
8     }
  
```

Quellcode 4.15: handle-Methode einer Middleware mit Argumenten

4.2.6.1. Zuweisung an Routen

Um Middleware oder Middleware-Gruppen in Controllern anzuwenden, wird folgende Syntax verwendet:

```

1 Route::middleware('verified')->group(function() {
2     Route::post('/facilities', [FacilityController::class, '
   addFacility']);
3     ...
4 }
  
```

Quellcode 4.16: Zuweisung einer Middleware für mehrere Routes

```

1 Route::post('/facilities', [FacilityController::class, 'addFacility'])->
   middleware('verified');
  
```

Quellcode 4.17: Zuweisung einer Middleware für eine einzelne Route

```

1 Route::post('admin/accounts/delete/{id}', [AccountController::class
2 , 'adminDeleteAccount'])->middleware('accountTypeWeb: Systemverwalter');
  
```

Quellcode 4.18: Zuweisung einer Middleware mit Argumenten

4.2.7. Strukturierung

Für das Routing der Webapplikation wurde folgende Strukturierung umgesetzt:

1. Controller

a) Api

- i. app/Http/Controllers/Api/AccountController.php
- ii. app/Http/Controllers/Api/BookmarkController.php
- iii. app/Http/Controllers/Api/EventController.php
- iv. app/Http/Controllers/Api/FacilityController.php
- v. app/Http/Controllers/Api/PasswordController.php
- vi. app/Http/Controllers/Api/RequestController.php
- vii. app/Http/Controllers/Api/VerificationController.php.php

b) Web

- i. app/Http/Controllers/Web/AccountController.php
- ii. app/Http/Controllers/Web/BookmarkController.php
- iii. app/Http/Controllers/Web/EventController.php
- iv. app/Http/Controllers/Web/FacilityController.php
- v. app/Http/Controllers/Web/PasswordController.php
- vi. app/Http/Controllers/Web/RequestController.php
- vii. app/Http/Controllers/Web/VerificationController.php

2. Middleware

a) Api

- i. app/Http/Middleware/Api/EventOwner.php
- ii. app/Http/Middleware/Api/FacilityManager.php
- iii. app/Http/Middleware/Api/ForceJsonResponse.php

b) Web

- i. app/Http/Middleware/Web/AccountType.php
- ii. app/Http/Middleware/Web/BookmarkOwner.php
- iii. app/Http/Middleware/Web/EventOwner.php
- iv. app/Http/Middleware/Web/FacilityManager.php
- v. app/Http/Middleware/Web/FacilityManagerById.php
- vi. app/Http/Middleware/Web/RequestOwner.php

3. Traits

- a) app/Http/Traits/AccountTrait.php
- b) app/Http/Traits/EventTrait.php
- c) app/Http/Traits/FacilityTrait.php
- d) app/Http/Traits/PasswordTrait.php
- e) app/Http/Traits/RequestTrait.php

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie Ausbildungsschwerpunkt: Medientechnik

4.3. E-Mail-System - KAUF

Um die Möglichkeit, seine E-Mail Adresse zu verifizieren und sein Passwort zurückzusetzen, zu garantieren, wurde in der Webapplikation ein E-Mail-System implementiert.

4.3.1. Verifizierung

Dieses basiert auf “MustVerifyEmail”, einem in Laravel integrierten System, welches Funktionalität für die Verifizierung von E-Mail-Adressen bereitstellt.

4.3.1.1. Implementierung im Account Model

Um dieses System verwenden zu können, müssen in der Account-Klasse die folgenden Klassen implementiert werden. Die Account-Klasse findet man unter ‘app/Http/Models/Account.php’.

- `Illuminate\Auth\MustVerifyEmail`
- `Illuminate\Contracts\Auth\MustVerifyEmail`

4.3.1.2. MustVerifyEmail Trait

‘`Illuminate\Auth\MustVerifyEmail`’ ist ein Trait, welches die Methoden `hasVerifiedEmail`, `markEmailAsVerified`, `sendEmailVerificationNotification` und `getEmailForVerification` enthält. Diese sind für die E-Mail-Verifizierung in Laravel erforderlich.

Um das Trait zu benutzen muss innerhalb der Account-Klasse folgendes use-Statement gesetzt werden:

```
1 use MustVerifyEmail;
```

4.3.1.3. MustVerifyEmail Contract

‘`Illuminate\Contracts\Auth\MustVerifyEmail`’ ist ein Contract. Contracts in Laravel geben vor, welche Methoden eine Klasse implementieren muss. Mit der Implementierung dieser Klasse wird also vorgegeben, dass die Funktionen, welche in dem Contract definiert sind, implementiert werden müssen. [35] (Stand 22.03.2023) Das sind die gleichen Funktionen, die auch in dem Trait definiert sind. Die Implementierung des Contracts kann wie folgt umgesetzt werden:

```
1 use Illuminate\Contracts\Auth\MustVerifyEmail as MustVerifyEmailContract;
2 class Account extends Model implements MustVerifyEmailContract, ...
3 {
4     ...
5 }
```

Quellcode 4.19: Implementation von MustVerifyEmail in Account-Model

4.3.1.4. Klassen Implementierung

Die Funktionalität des E-Mail-Systems wird vorgegeben, in dem die Klassen, welche auch innerhalb des Traits und des Contracts vorgegeben sind, im Account Model definiert werden. Dadurch wird vorhandene Funktionalität ersetzt und die E-Mail-Verifizierung kann angepasst werden.

Anhand des Models wurden die Klassen wie folgt definiert:

```

1 public function getEmailForVerification ()
2     {
3         return $this->EMAIL;
4     }
5
6     public function sendEmailVerificationNotification ()
7     {
8         $this->notify (new VerifyEmailNotification ());
9     }
10
11    public function markEmailAsVerified ()
12    {
13        return $this->forceFill ([
14            'IS_EMAIL_VERIFIED' => true,
15        ])->save ();
16    }
17
18    public function hasVerifiedEmail ()
19    {
20        return $this->IS_EMAIL_VERIFIED;
21    }

```

Quellcode 4.20: Implementation der benötigten Klassen von MustVerifyEmail im Account-Model

4.3.1.5. Nutzung

Um jetzt eine eine Verifizierungs-E-Mail abzusenden, wird folgendes getan:

```

1 $account->sendEmailVerificationNotification ();

```

4.3.2. Passwort Zurücksetzung

Ebenfalls ist es ein Wunsch, ein System für das Zurücksetzen des Passworts eines Benutzer zu implementieren.

Dies wird in Jugendliches Krems mit dem Laravel-Tool “CanResetPassword” gemacht. Die Implementation geschieht sehr ähnlich dem Modus der Implementation von MustVerifyEmail. Aus diesem Grund wird die Implementation der Klassen in dem Account Model (app/Http/Models/Account.php) und die Nutzung vereinfacht beschrieben.

4.3.2.1. Klassen Implementation

```
1 public function getAuthIdentifierName ()
2     {
3         return $this->getKeyName ();
4     }
5
6     public function getAuthPassword ()
7     {
8         return $this->PASSWORD;
9     }
10
11    public function getRememberToken ()
12    {
13        return $this->{$this->getRememberTokenName () };
14    }
15
16    public function setRememberToken ($value)
17    {
18        $this->{$this->getRememberTokenName () } = $value;
19    }
20
21    public function getRememberTokenName ()
22    {
23        return 'REMEMBER_TOKEN';
24    }
25    public function getAuthIdentifier ()
26    {
27        return $this->{$this->getAuthIdentifierName () };
28    }
29    public function getEmailForPasswordReset ()
30    {
31        return $this->EMAIL;
32    }
33
34    public function sendPasswordResetNotification ($token)
35    {
36        $this->notify (new ResetPasswordNotification ($token));
37    }
```

Quellcode 4.21: Implementation der benötigten Klassen von CanResetPassword im Account-Model

4.3.2.2. Nutzung

Die Passwort-Rücksetzfunktion wird für die Webapplikation in dem Controller 'app/Http/Controllers/Web/PasswordController.php' verwendet:

```
1 ...
2
3 class PasswordController extends Controller
4 {
5     use PasswordTrait;
6     public function sendResetLinkEmail()
7     {
8         $this->validateResetEmail(request());
9
10        $status = Password::sendResetLink(
11            request()->only('email')
12        );
13    }
14    public function resetPassword()
15    {
16        $this->validateReset(request());
17
18        $token = request('token');
19        $email = request('email');
20
21        $resettable = PasswordResets::where('email', $email)->first();
22
23        if (!$resettable) {
24            return response()->json(['error' => 'Invalid_email.'], 401);
25        }
26
27        if (!Hash::check($token, $resettable->token)) {
28            return response()->json(['error' => 'Invalid_token.'], 401);
29        }
30
31        $email = $resettable->email;
32
33        $account = Account::where('EMAIL', $email)->first();
34
35        $account->PASSWORD = Hash::make(request('PASSWORD'));
36        $account->save();
37
38        $resettable->delete();
39    }
40 ...
41 }
```

Quellcode 4.22: Nutzung der CanResetPassword-Funktionalität in einem Controller

4.3.3. Laravel Notification

Notifications in Laravel sind eine Option, Nachrichten via verschiedener Kanäle wie zum Beispiel SMS und E-Mail zu senden. Notifications werden innerhalb des Ordners "App\Http\Notifications"

gespeichert.

Die Webapplikation verwendet nur E-Mails als Form der Kommunikation extern der Webseite. Es könnte deshalb von einer Implementierung von E-Mails via Notifications abgesehen werden. Allerdings ist es ein Best Practice, jene gleich mit Notifications umzusetzen, da es dadurch leichter wird, in der Zukunft das System auf SMS-Benachrichtigungen auszuweiten.

Innerhalb der Notifications für E-Mail-Verifizierung und Passwort-Zurücksetzung werden die Mail-Klassen in der “toMail”-Methode als Rückgabewert definiert. Für die ResetPassword Notification (app/Notifications/ResetPasswordNotification.php) sieht dies folgendermaßen aus:

```

1 public function toMail($notifiable)
2     {
3         // return new verifyEmail
4         return (new ResetPasswordMail)
5             ->with([
6                 'reset_link' => $this->resetUrl($notifiable),
7                 'name' => $notifiable->NAME,
8             ])
9             ->to($notifiable->getEmailForPasswordReset());
10    }

```

Quellcode 4.23: toMail-Methode von ResetPasswordNotification.php

Hier wird ersichtlich, dass innerhalb der toMail-Funktion die Funktion resetUrl aufgerufen wird. Diese muss ebenfalls in der Notification definiert werden:

```

1 protected function resetUrl($notifiable)
2     {
3         return url(route('password.reset', [
4             'token' => $this->token,
5             'email' => $notifiable->getEmailForPasswordReset(),
6             ], false));
7     }

```

Quellcode 4.24: resetUrl-Methode von ResetPasswordNotification.php

Wenn beim Senden einer Notification einen Eingabeparameter übermittelt wird, dann ist innerhalb der Notification eine neue Variable, welche die Funktion hat, diesen Eingabeparameter zu setzen, und ein Konstruktor, welcher diese Variable gleich dem Eingabeparameter setzt, zu definieren. Dies sieht folgendermaßen aus:

```

1 protected $token;
2
3 public function __construct($token)
4     {
5         $this->token = $token;
6     }

```

Quellcode 4.25: Konstruktor von ResetPasswordNotification.php

4.3.4. Laravel Mailable

Innerhalb der Notifications werden Mail-Klassen, also Klassen, die Kindklassen von Mailable sind, zurückgegeben.

“Mit Mailables in Laravel werden vorgefertigte Klassen erstellt, die später verschickt werden können. Grundsätzlich sind Mailables verantwortlich für die Datenzusammenstellung und diese dann an die Views weiterzugeben.”
[36] (Stand 22.03.2023)

Die zwei Klassen innerhalb von Mail-Klassen, die interessant sind, sind “envelope” und “content”.

Innerhalb der “envelope”-Funktion wird der Titel der E-Mail definiert. Bei der Mail-Klasse für das Zurücksetzen des Passwortes sieht jene wie folgt aus:

```
1 public function envelope()  
2 {  
3     return new Envelope(  
4         subject: 'Passwort_zuruecksetzen',  
5     );  
6 }
```

Quellcode 4.26: envelope-Funktion von app/Mail/ResetPassword.php

In der “content”-Funktion wird daraufhin der Inhalt der E-Mail definiert. Hier empfiehlt es sich, den Inhalt auf eine externe Datei zu verlegen, welche in “app/resources/views/emails” definiert werden kann: sieht jene wie folgt aus:

```
1 public function content()  
2 {  
3     return new Content(  
4         view: 'emails.reset-password',  
5     );  
6 }
```

Quellcode 4.27: content-Funktion von app/Mail/ResetPassword.php

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie Ausbildungsschwerpunkt: Medientechnik

4.3.5. E-Mail Vorlagen

Die E-Mail-Vorlagen sind als Blade-Dateien zu erstellen.

”Blade ist eine Template-Engine vom PHP-Webframework Laravel. Alle Blade-Templates haben die Dateiendung “.blade.php”, und können mit der Funktion „view“ vom Controller aus aufgerufen werden.” [37] (Stand 22.03.2023)

Innerhalb einer E-Mail Vorlage kann mithilfe des Blade Syntax auf Variablen, die in der “toMail”-Methode der Notification definiert wurden, zugegriffen werden:

```
1 {{ $reset_link }}
```

4.4. Authentifizierung - KAUF

Es gibt viele verschiedene Authentifizierungskonzepte in Laravel. Ein großer Vorteil hierbei ist, dass Laravel verschiedene Authentifizierungen für eine API und eine Webseite zulässt.

Die verwendeten Authentifizierungskonzepte sind in der Konfigurationsdatei “config\auth.php” festzulegen.

4.4.1. Webseite

Für die Webseite wurde Session-basierte Authentifizierung ausgewählt.

“Laravel includes built-in authentication and session services which are typically accessed via the Auth and Session facades. These features provide cookie-based authentication for requests that are initiated from web browsers. They provide methods that allow you to verify a user’s credentials and authenticate the user. In addition, these services will automatically store the proper authentication data in the user’s session and issue the user’s session cookie.” [38] (Stand 22.03.2023)

4.4.2. API

Laravel bietet für die Authentifizierung von APIs zwei verschiedene Methoden: Laravel Sanctum und Laravel Passport. Laravel Sanctum verwendet eine Token-basierte Authentifizierungsmethode, während Laravel Passport OAuth, ein Protokoll, dass eine standardisierte, sichere API-Autorisierung für Desktop-, Web- und Mobile-Anwendungen erlaubt. [39] (Stand 22.03.2023)

Aufgrund der Empfehlung von Laravel wurde als Authentifizierungskonzept für die API Laravel Sanctum ausgewählt:

“If your application absolutely needs to support OAuth2, then you should use Laravel Passport. However, if you are attempting to authenticate a single-page application, mobile application, or issue API tokens, you should use Laravel Sanctum. Laravel Sanctum does not support OAuth2; however, it provides a much simpler API authentication development experience.” [40] (Stand 22.03.2023)

”Bei Verwendung eines tokenbasierten Authentifizierungssystems werden die Anmeldeinformationen der Benutzer nur einmal verifiziert. Im Anschluss an die Verifizierung erhalten sie ein Token, das ihnen für einen von Ihnen festgelegten Zeitraum Zugriff gewährt.” [41] (Stand 22.03.2023)

4.5. Frontend - KAUF

4.5.1. Webseitenstruktur

Ein wesentlicher Faktor für Webseiten ist die gute Strukturierung dieser.

4.5.2. Vorteile

Eine gute und klare Strukturierung bietet folgende Vorteile:

4.5.2.1. Benutzerfreundlichkeit

Durch eine klare und logische Struktur ist es für Benutzer einfacher, die gewünschten Inhalte auf einer Webseite zu finden oder sich schnell einen Überblick über die Webseite zu verschaffen. Laut einer Studie der Missouri University of Science and Technology sind vor allem die ersten Sekunden ausschlaggebend, daher ist es notwendig, eine einfache Struktur zu implementieren, welche Benutzer nicht überfordert und von diesen sehr schnell verstanden werden kann:

“When viewing a website, it takes users less than two-tenths of a second to form a first impression, according to recent eye-tracking research conducted at Missouri University of Science and Technology. But it takes a little longer – about 2.6 seconds – for a user’s eyes to land on that area of a website that most influences their first impression.” [42] (Stand 22.03.2023)

4.5.2.2. Suchmaschinenoptimierung

Suchmaschinen wie Google bewerten Webseiten mit einer klaren Struktur höher und zeigen jene in den Suchergebnissen von Suchanfragen höher an. Da heutzutage viele Benutzer/User Suchmaschinen verwenden, um Inhalte im Internet zu finden, ist es wichtig, eine gute Strukturierung umzusetzen, um ein höheres Ranking von Suchmaschinen zu erhalten.

4.5.2.3. Barrierefreiheit

Eine gute Strukturierung erhöht die Zugänglichkeit für Menschen mit Behinderungen und besonderen Bedürfnissen.

“Der Struktur einer Website kommt eine zentrale Bedeutung für die Barrierefreiheit zu. Der Absender, also die Identität, und der Auftrag der Website müssen jederzeit deutlich sein, so dass dem Website-Besucher klar ist, auf welcher Seite er sich befindet und welche Informationen hier zu finden sind. Das Menü sollte daher übersichtlich sein und nicht mehr als 5 – 7 Menüpunkte erhalten. Diese sollten kurz, verständlich und eindeutig formuliert sein.

Alle Inhalte sollten innerhalb weniger Klicks erreichbar sein. Auf den Seiten selbst helfen klare Strukturen beim Zurechtfinden und Sortieren des Inhalts. Achten Sie beispielsweise auf einen standardisierten Seitenaufbau in Überschriften, Absätze, Zwischenüberschriften und – wo sinnvoll – Aufzählungen. Verzichten Sie außerdem auf Tabellen auf der Website. Tabellen führen immer wieder zu Problemen in der Darstellung, insbesondere auf mobilen Endgeräten. Auch die Bedienbarkeit der Seiten für Menschen mit Sehbehinderung wird durch Tabellen verschlechtert. Viele Inhalte lassen sich auch durch gute Formatierung und Strukturierung ansprechend und übersichtlich darstellen.” [43] (Stand 22.03.2023)

4.5.2.4. Wartbarkeit

Durch eine gute Strukturierung ist es einfacher, Webseiten zu warten und zu aktualisieren. Durch einen logischen Aufbau können Änderungen im Code schnell und effizient umgesetzt werden, ohne Code umstrukturieren zu müssen, oder vor der Aktualisierung lange nach dem betreffenden Code suchen zu müssen.

4.5.3. Webseitenstruktur von Jugendliches Krems

1. Events

a) Alle Events

- URL: /events
- Datei: resources/js/Pages/Events.svelte

b) Event

- URL: /events/{id}
- Datei: resources/js/Pages/Event.svelte

c) Event erstellen

- URL: /newevent
- Datei: resources/js/Pages/CreateEvent.svelte

2. Bildung

a) Alle Bildungsanstalten

- URL: /facilities
- Datei: resources/js/Pages/Facilities.svelte

b) Bildungsanstalt

- URL: /facilities/{id}
- Datei: resources/js/Pages/Facility.svelte

c) Bildungsanstalt erstellen

- URL: /newfacility
- Datei: resources/js/Pages/CreateFacility.svelte

3. Lesezeichen

a) Alle Lesezeichen

- URL: /bookmarks
- Datei: resources/js/Pages/Bookmarks.svelte

4. Account

a) Mein Account

- URL: /account
- Datei: resources/js/Pages/Account.svelte

b) Meine Bildungsanstalten

- URL: /account/facilities
- Datei: resources/js/Pages/Account/Facilities.svelte

c) Meine Events

- URL: /account/events
- Datei: resources/js/Pages/Account/Events.svelte

d) Meine Anträge

- URL: /account/requests
- Datei: resources/js/Pages/Account/Requests.svelte

e) E-Mail Bestätigung

- URL: /account/verify
- Datei: resources/js/Pages/Account/Verify.svelte

5. Dashboard

a) Dashboard

- URL: /dashboard
- Datei: resources/js/Pages/Dashboard.svelte

b) Konten

- URL: /dashboard/accounts
- Datei: resources/js/Pages/Dashboard/Accounts.svelte

c) Events

- URL: /dashboard/events
- Datei: resources/js/Pages/Dashboard/Events.svelte

d) Bildungsanstalten

- URL: /dashboard/facilities
- Datei: resources/js/Pages/Dashboard/Facilities.svelte

e) Anträge

- URL: /dashboard/requests
- Datei: resources/js/Pages/Dashboard/Requests.svelte

6. Autorisierung

a) Anmelden

- URL: /login
- Datei: resources/js/Pages/Login.svelte

b) Abmelden

- URL: /logout
- Datei: resources/js/Pages/Logout.svelte

c) Registrieren

- URL: /register
- Datei: resources/js/Pages/Register.svelte

d) Passwort zurücksetzen

- URL: /forgotpassword
- Datei: resources/js/Pages/PasswordForgot.svelte

4.5.4. CSS Frameworks

Die Entwicklung eines ansprechenden Designs mit CSS kann sich als sehr aufwendig herausstellen. Styles müssen oft mehrmals definiert werden und es ist schwierig, im Prozess der Entwicklung immer eine logische Klassenstruktur einzuhalten. Abhilfe schaffen hierbei CSS Frameworks. CSS Frameworks beinhalten vordefinierte Klassen und Farbschemen, die Entwickler unterstützen, effizienten Code zu schreiben und einen effizienten Arbeitsablauf einzuhalten.

Doch auch CSS Frameworks arbeiten mit unterschiedlichen Ansätzen. Manche Frameworks wie zum Beispiel Bootstrap setzen viel auf vordefinierte Komponenten, während andere Frameworks vordefinierte Komponenten vermeiden, mit dem Wunsch, dass Komponenten vom Entwickler selbst entworfen werden. Ein Beispiel für ein solches CSS Framework ist Tailwind, welches sich momentan an hoher Popularität erfreut.

Beide Ansätze bieten Vorteile und Nachteile. Das Erstellen eines ansprechendes Design ist grundsätzlich mit weniger Aufwand verbunden als das Erstellen von responsive Komponenten. Falls bestimmte Aspekte von mehreren Frameworks angebracht sind, ist es auch möglich, einfach mehrere Frameworks in die Applikation einzubinden.

In der Webapplikation werden zwei Frameworks verwendet: Bootstrap und Tailwind. Tailwind wird für den Großteil der Webseite verwendet, während Bootstrap Anwendung bei responsive Komponenten wie der Navbar und dem Footer findet.

4.5.5. Programmierung mit Svelte

4.5.5.1. Struktur

Die Struktur von Svelte Seiten ist sehr einfach. Es gibt einige unterschiedliche Komponenten, welche nacheinander in die Seite eingefügt werden:

<script>

Die erste Komponente ist die script-Komponente. In dieser sind Tools, welche nicht in der Basisfunktionalität von Svelte inkludiert sind, zu importieren und JavaScript-Code auszuführen.

In Error-Seite von Jugendliches Krems (resources/js/Pages/Error.svelte):

```
1 <script>
2   import CenterDiv from '../Shared/CenterDiv.svelte';
3   import Button from '../Shared/Button.svelte';
4   import H1 from '../Shared/H1.svelte';
5
6   export let status;
7
8   $: title = {
```

```

 9      503: '503:_Service_nicht_verfuegbar ',
10      500: '500:_Server_Fehler ',
11      404: '404:_Seite_nicht_gefunden ',
12      403: '403:_Zugriff_verweigert ',
13      401: '401:_Nicht_authorized ',
14      419: '419:_Sitzung_abgelaufen ',
15    }[status]
16 </script>

```

Quellcode 4.28: Script-Komponente von Error.svelte

<svelte:head>

Die nächste Komponente ist die svelte:head-Komponente. Diese Komponente funktioniert so wie der <head> einer HTML-Seite. Es werden hier zum Beispiel der "title" der Seite und meta-tags definiert. Innerhalb dieser Komponente ist es ebenfalls möglich geschwungene Klammern zu verwenden, um auf JavaScript-Variablen zuzugreifen oder JavaScript-Code auszuführen.

In Error-Seite von Jugendliches Krems:

```

1 <svelte:head>
2   <title>Error</title>
3   <meta name="description" content="{description}">
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0,
   maximum-scale=1.0" />
6 </svelte:head>

```

Quellcode 4.29: svelte:head-Komponente von Error.svelte

<style>

Die Style-Komponente verhält sich genau so wie in HTML. Hier findet sich die Definition von Styles für die Seite. Diese werden nur auf die momentane Seite angewendet, was die Wartung der Style-Komponenten erleichtert.

Body

Der Code, der nach diesen drei Komponenten steht, wird im Browser dargestellt. Es ist nicht nötig, eine weitere Komponente zu definieren. Auch hier kann mithilfe von geschwungenen Klammern auf JavaScript Variablen zugegriffen oder JavaScript Code ausgeführt werden.

In Error-Seite von Jugendliches Krems:

```

1 <CenterDiv>
2   <div class="tw-p-3">
3     <H1 mb={false}>{title}</H1>
4     <Button link="/">Zur Startseite</Button>
5   </div>
6 </CenterDiv>

```

Quellcode 4.30: svelte:Body von Error.svelte

4.5.5.2. Router

Inertia bietet für die Entwicklung mit Svelte einen "router", mit welchem innerhalb von Svelte-Seiten Routing-Operationen durchgeführt werden können. In der Webapplikation wurden folgende Funktionen von router verwendet:

- `router.reload()`: Diese Funktion aktualisiert die Daten der Seite.

```
1 router.reload();
```

- `router.get()`: Diese Funktion erstellt einen GET-Request. Es erfolgt demgemäß eine Weiterleitung zu der Route. Beispiel:

```
1 router.get('/');
```

- `router.post()`: Diese Funktion erstellt einen POST-Request. Es geschieht dementsprechend eine Veränderung der Daten der Applikation. Beispiel:

```
1 router.post('/requests/delete/' + request.REQUEST_ID);
```

Es ist möglich den Router wie folgt zu importieren:

```
1 import { router } from '@inertiajs/svelte';
```

4.5.5.3. Link

Link ist eine weitere Klasse, welche von Inertia für die Entwicklung mit Svelte zur Verfügung gestellt wird. Link ersetzt den anchor-tag `<a>` und lädt den Inhalt der verknüpften Seite mit AJAX, anstelle die komplette Seite neu zu laden. Dies führt zu einer schnellen und flüssigeren UX.

Link wird wie folgt importiert:

```
1 import { router, Link } from "@inertiajs/svelte";
```

Der Syntax für die Verwendung sieht wie folgt aus:

```
1 <Link href="/account/facilities">Bildungs Event erstellen </Link>
```

4.5.5.4. Axios

Axios ist ein weiterer HTTP-Client und stellt ähnliche Funktionalität wie router zur Verfügung.

Die Entscheidung zwischen einer der Beiden ist stark von den eigenen Präferenzen zur Syntax abhängig, da die Syntax von axios und der Syntax von router sich stark unterscheiden

können.

In Jugendliches Krems finden beide Technologien ihren Einsatz.

4.5.5.5. Page

Page ist ein weiteres Tool von Inertia und bietet Funktionalität, um Variablen über alle Svelte Seiten hinweg zu definieren. Um page zu verwenden, muss es zuerst importiert werden:

```
1 import { router, page } from "@inertiajs/svelte";
```

Die Variablen, die überall dort zur Verfügung gestellt werden sollen, wo page importiert wurde, müssen in der Funktion "share" innerhalb der Datei 'HandleInertiaRequets.php' in 'app\Http\Middleware' definiert werden.

Im Fall von Jugendliches Krems sieht diese Funktion wie folgt aus:

```
1 public function share(Request $request): array
2     {
3         return array_merge(parent::share($request), [
4             'appName' => config('app.name'),
5             'auth.user' => fn () => $request->user()
6                 ? $request->user()
7                 : null,
8             'bookmarks' => fn () => $request->user()
9                 ? $request->user()->bookmarks/* ->load('event', 'facility
10                */
11                 : null,
12             'path' => fn () => $request->path(),
13             'csrfToken' => csrf_token(),
14         ]);
15     }
```

Quellcode 4.31: share-Funktion von HandleInertiaRequets.php

Der Zugriff auf page-Variablen geschieht dann folgendermaßen:

```
1 $page.props.auth.user?.ACCOUNT_ID
```

4.6. Arbeiten mit React Native und Expo - SPIN

4.6.1. Grundlegendes

Expo ermöglicht die Erstellung einer vereinfachten Version einer React Native-Anwendung und bietet die Möglichkeit, diese während der Entwicklung auf einem Endgerät mit der App 'Expo Go' zu testen. React Native arbeitet mit JavaScript-Dateien oder, wie im Fall Jugendliches Krems, mit TypeScript-Dateien.

4.6.1.1. JSX

React Native verwendet JSX, das eine XML-ähnliche Struktur aufweist. Dies bedeutet, dass es ähnlich wie HTML in Blöcke unterteilt ist. Dabei darf immer nur ein solcher Block vorhanden sein, und alle anderen Blöcke müssen in diesem verschachtelt sein. React Native bietet dafür eine Komponente namens 'View' an. Diese Komponente besitzt keine besonderen Eigenschaften und kann somit als oberste Komponente verwendet werden, in der andere Komponenten ausgeführt werden. Ein Beispiel hierfür wäre:

```
1 <View>
2   <Text>Ein Text Block</Text>
3   <Text>Ein anderer Text Block</Text/>
4 </View>
```

Quellcode 4.32: Beispiel einer JSX Komponente

In diesem Beispiel wird deutlich, dass es genau eine übergeordnete Komponente gibt, nämlich 'View'. Alle anderen Komponenten sind innerhalb dieser verschachtelt. Die 'Text'-Komponente dient lediglich dazu, React Native mitzuteilen, dass es sich hierbei um einen Text handelt, der auf dem Endgerät angezeigt werden soll.

4.6.2. Struktur von React Native

In React Native hat eine Seite stets den gleichen grundlegenden Aufbau.

4.6.2.1. Imports

Zunächst finden sich am Anfang der Seite immer die 'import'-Anweisungen. Diese dienen dazu, die benötigten Bibliotheken zu laden, damit sie später im Code verwendet werden können. Die grundlegendsten Import-Anweisungen sind:

```
1 import React from 'react';
2 import { ... } from 'react-native';
```

Quellcode 4.33: Beispiel von Import Statements

In diesem Beispiel steht '...' für einen Platzhalter, der durch Komponenten aus der React Native API ersetzt werden kann. [44]

4.6.2.2. Funktion

In einer React Native Datei befindet sich stets eine Funktion, in der der auszuführende Code enthalten ist. Der Code, der auf dem Endgerät angezeigt werden soll, befindet sich im 'return'-Statement. Ein Beispiel hierfür wäre:

```
1 function Example () {
2   return (
3     <Text>Hello World</Text>
```

```
4     );  
5 }
```

Quellcode 4.34: Beispiel einer Funktion in React Native

In diesem Beispiel handelt es sich um eine Funktion namens 'Example', die beim Aufruf einen Text mit dem Inhalt 'Hello World' auf dem Endgerät anzeigt.

4.6.2.3. Export Statement

In einer React Native Datei können mehrere Funktionen definiert sein. Um React Native mitzuteilen, welche Funktion standardmäßig ausgeführt werden soll, wird am Ende jeder Datei ein 'export'-Statement hinzugefügt. Ein Beispiel hierfür wäre:

```
1 export default Example;
```

Quellcode 4.35: Beispiel eines Export Statements

Durch diese Anweisung weiß React Native, dass beim Aufruf der Datei standardmäßig die Funktion 'Example' ausgeführt werden soll.

4.6.3. Beispiel einer React Native Seite

Eine voll funktionsfähige React Native-Seite könnte wie folgt aussehen:

```
1 import React from 'react';  
2 import {Text} from 'react-native';  
3  
4 function Example () {  
5     return (  
6         <Text>Hello World</Text>  
7     );  
8 }  
9 export default Component;
```

Quellcode 4.36: Beispiel einer React Native Seite

Der oben gezeigte Code repräsentiert eine React Native-Komponente, die eine Textnachricht ('Hello World') anzeigt. In der ersten Zeile werden die 'React'-Bibliothek und die 'Text'-Komponente aus der 'react-native'-Bibliothek importiert. Anschließend wird eine neue Komponente namens 'Component' definiert, die in diesem Fall eine einfache Textnachricht zurückgibt.

Innerhalb der 'return'-Anweisung befindet sich der eigentliche Inhalt der Komponente, in diesem Beispiel ein 'Text'-Element mit dem Text 'Hello World'. Abschließend wird die 'Component'-Komponente als Standardexport der Datei festgelegt, sodass sie von anderen Dateien und Komponenten innerhalb der App importiert und verwendet werden kann.

Insgesamt zeigt dieser Code, wie man eine einfache React Native-Komponente erstellt und einen Text innerhalb der App darstellt.

4.6.4. Funktionsweise von React Native

Nachdem ein Expo-Projekt erstellt wurde, generiert es die Datei 'App.tsx'. Diese Typescript-Datei enthält den gesamten Code, der auf dem Endgerät angezeigt wird. Um den Code auf mehrere Dateien aufzuteilen, können diese importiert und eingefügt werden.

```
1 import EventScreen from './app/screens/EventScreen';
```

Quellcode 4.37: Beispiel für importieren einer anderen Seite

Im obigen Beispiel importieren wir die Datei 'EventScreen.tsx' und speichern sie als Konstante 'EventScreen' ab. Um den Code aus dieser Datei in 'App.tsx' einzubetten, muss man lediglich eine Komponente mit der konstanten Variable erstellen.

```
1 ...  
2 <EventScreen/>  
3 ...
```

Quellcode 4.38: Beispiel Verwendung importierter Seiten

Dieses Prinzip funktioniert nicht nur in 'App.tsx', sondern in jeder Datei. Dadurch kann der Code in verschiedenen Dateien unterteilt und genau dort eingefügt werden, wo er benötigt wird.

4.7. Navigation in React Native - SPIN

Für eine optimale Navigation in der mobilen Anwendung ist es zunächst erforderlich, die Struktur der Applikation festzulegen.

4.7.1. Struktur der mobilen Applikation

Die Strukturierung wurde entsprechend wie folgt festgelegt:

1. Events
 - a) Alle Events
 - b) Bildungs-Events
 - c) Freizeit-Events
 - d) Falls Benutzer ein Event gehört
 - i. Event bearbeiten
 - ii. Event löschen
2. Bildungsanstalten
 - a) Falls Benutzer eine Bildungsanstalt verwaltet
 - i. Bildungsanstalt bearbeiten
 - ii. Event für diese Bildungsanstalt erstellen
3. Lesezeichen
 - a) Falls Benutzer angemeldet ist
 - i. Freizeit-Events
 - ii. Bildungs-Events
 - iii. Bildungsanstalten
 - b) Falls Benutzer nicht angemeldet ist
 - i. Anmelden
4. Account
 - a) Falls Benutzer angemeldet ist
 - i. Passwort ändern
 - ii. Datenschutzbestimmungen
 - iii. Event erstellen
 - iv. Bildungsanstalt registrieren
 - v. Abmelden
 - b) Falls Benutzer nicht angemeldet ist
 - i. Anmelden
 - ii. Datenschutzbestimmungen
 - iii. Nutzungsbedingungen

4.7.2. Navigation der Hauptbereiche der Anwendung

Für die Navigation zu den Hauptbereichen der Anwendung, wie Events, Bildungsanstalten, Lesezeichen und Account, wurde eine Navigationsleiste eingesetzt.

Die Navigationsleiste wurde mithilfe einer React Native-Bibliothek namens 'Bottom Tabs Navigator' erstellt. [45]

4.7.2.1. Syntax

Um die Bibliothek zu verwenden, muss sie zunächst in das Programm importiert und einer konstanten Variable zugewiesen werden.

```
1 import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
2 const Tab = createBottomTabNavigator();
```

Quellcode 4.39: Beispiel für Importieren wie es im Projekt verwendet wurde

Zur Nutzung in JSX muss ein 'NavigationContainer' angelegt werden.

```
1 <NavigationContainer>
2 </NavigationContainer>
```

Quellcode 4.40: Beispiel NavigationContainer

Diese Komponente benötigt zur korrekten Funktionsweise zwei weitere Komponenten: einen Navigator, der angibt, welche Seite zuerst angezeigt werden soll, und Screen-Komponenten, die die einzelnen Seiten definieren.

Beispiel für einen 'Navigator':

```
1 <Tab.Navigator initialRouteName={EventName} screenOptions={({route}) => ({
2     tabBarIcon : ({focused, color, size}) => {
3         let iconName : string = '';
4         let rn = route.name;
5         color = focused ? '#FFCC00' : '#999999'
6         size = 35;
7
8         if(rn === EventName) {
9             iconName = 'calendar';
10            //iconName = focused ? 'calendar' : 'calendar-outline';
11        } else if (rn === EducationName) {
12            iconName = 'school';
13            //iconName = focused ? 'school' : 'school-outline';
14        } else if (rn === FavoriteName) {
15            iconName = 'bookmark';
16            //iconName = focused ? 'bookmark' : 'bookmark-outline';
17        } else if (rn === SettingName) {
18            iconName = 'settings';
19            //iconName = focused ? 'settings' : 'settings-outline';
20        }
21    })
22 >
```

```

21
22         return <Icons name={IconName} size={size} color={color}/>
23     },
24     headerShown : false ,
25     tabBarShowLabel : false ,
26     tabBarStyle : {
27         padding : 10,
28         height : 80,
29         //display: 'none'
30         //display : visibility
31     },
32     tabBarLabelStyle : {
33         fontSize : 12,
34         color : '#FFCC00',
35         fontWeight : 'bold'
36     }
37     }>>>

```

Quellcode 4.41: Beispiel Navigator

Die Verwendung des Begriffs 'Tab' vor 'Navigator' bedeutet, dass wir die zuvor definierte Konstante Tab verwenden möchten, um unsere Navigationsleiste zu erstellen. Der Parameter 'initialRouteName' gibt an, welche Seite beim Start der App angezeigt werden soll, in diesem Beispiel die Seite mit den Events. Mit 'screenOptions' können das Aussehen und das Verhalten der Navigationsleiste auf dem Endgerät festgelegt werden. In diesem Beispiel erhalten die einzelnen Navigationspunkte passende Icons, die ihre Farbe ändern, je nachdem, ob sie ausgewählt sind oder nicht. Zusätzlich wurden weitere Anpassungen vorgenommen, wie das Deaktivieren der schriftlichen Bezeichnung und das Ausblenden des automatisch erzeugten Headers.

Beispiel für einen 'Screen':

```

1 <Tab.Screen name={SettingName} component={SettingsNavigation} options={()
  => ({tabBarStyle: {display: 'none'}})}>

```

Quellcode 4.42: Beispiel Screen

Der Begriff 'Tab' wird erneut verwendet, um die zuvor definierte Konstante zu verwenden. Der Parameter 'name' gibt der Komponente einen Wert, auf den referenziert werden kann, falls ein Wechsel auf diese Seite durch bestimmte Ereignisse erforderlich ist. Der Parameter 'component' definiert, welche Seite beim Auswählen des Punktes in der Navigationsleiste angezeigt werden soll, in diesem Beispiel die Account-Seite. Mit dem Parameter 'options' können verschiedene Einstellungen getroffen werden, die speziell für diese Seite gelten. Der Unterschied zum Navigator besteht darin, dass dieser alle Seiten anspricht. In diesem Beispiel wurde die Navigationsleiste ausgeblendet, wenn die Seite angezeigt wird.

Beispiel für eine zusammengesetzte Navigationsleiste:

```

1 <NavigationContainer>
2   <Tab.Navigator initialRouteName={EventName} screenOptions={{route}}
  => ({

```

```

3      tabBarIcon : ({ focused , color , size }) => {
4          let iconName : string = '';
5          let rn = route.name;
6          color = focused ? '#FFCC00' : '#999999'
7          size = 35;
8          if(rn === EventName) {
9              iconName = 'calendar';
10             //iconName = focused ? 'calendar' : 'calendar-outline';
11         } else if (rn === EducationName) {
12             iconName = 'school';
13             //iconName = focused ? 'school' : 'school-outline';
14         } else if (rn === FavoriteName) {
15             iconName = 'bookmark';
16             //iconName = focused ? 'bookmark' : 'bookmark-outline';
17         } else if (rn === SettingName) {
18             iconName = 'settings';
19             //iconName = focused ? 'settings' : 'settings-outline';
20         }
21         return <Ionicons name={iconName} size={size} color={color}/>
22     },
23     headerShown : false ,
24     tabBarShowLabel : false ,
25     tabBarStyle : {
26         padding : 10,
27         height : 80,
28         //display: 'none'
29         //display : visibility
30     },
31     tabBarLabelStyle : {
32         fontSize : 12,
33         color : '#FFCC00',
34         fontWeight : 'bold'
35     }
36     }>>>
37     <Tab.Screen name={EventName} component={EventScreen}/>
38     <Tab.Screen name={EducationName} component={EducationScreen}/>
39     <Tab.Screen name={FavoriteName} component={FavoriteScreen}/>
40     <Tab.Screen name={SettingName} component={SettingsNavigation}
41     options={() => ({tabBarStyle: {display: 'none'}})}>
42     </Tab.Navigator>
43 </NavigationContainer>

```

Quellcode 4.43: Beispiel einer Navigationsleiste wie sie im Projekt angewendet wurde

Die dargestellte Navigationsleiste besteht aus einem 'Navigator', der vier 'Screen'-Komponenten enthält, um zwischen den Bereichen Events, Bildungsanstalten, Lesezeichen und Account zu wechseln.

Auf dem Endgerät sieht diese so aus:

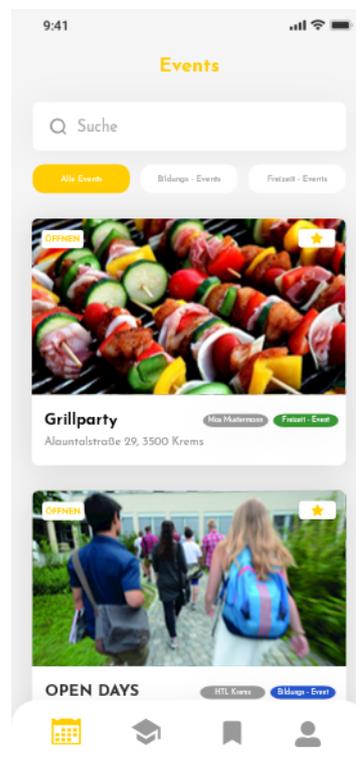


Abbildung 4.2.: Veranschaulichung der Bottom Navigation Bar

4.7.3. Navigation durch bestimmte Ereignisse

Neben den vier Hauptbereichen der Anwendung wird auch der 'Stack Navigator' für die Navigation verwendet. Der Stack Navigator verwaltet eine Liste von Bildschirmen und ermöglicht es Benutzern, zwischen ihnen zu navigieren, indem sie auf Schaltflächen, Links klicken. Zudem merkt sich der Stack Navigator die zuvor besuchte Seite, sodass man durch Klicken auf 'Zurück' zu dieser zurückkehren kann. [46]

4.7.3.1. Syntax

Ähnlich wie beim Bottom Tabs Navigator muss zunächst die Bibliothek importiert und einer Konstanten zugewiesen werden.

```
1 import { createStackNavigator } from '@react-navigation/stack';  
2 const Stack = createStackNavigator();
```

Quellcode 4.44: Beispiel für Importieren wie es im Projekt verwendet wurde

Der Aufbau in JSX ist ebenfalls sehr ähnlich, und es werden wieder ein 'Navigator' und 'Screen'-Komponenten benötigt.

Ein Beispiel dafür ist:

```

1 <Stack.Navigator initialRouteName='Settings' >
2   <Stack.Screen name='ResetPass' component={ResetPassword} options={{
3     headerShown: false }}/>
4   <Stack.Screen name='Settings' component={SettingScreen} options={{
5     headerShown: false }}/>
6   <Stack.Screen name='DataRights' component={DataRightsScreen}
7     options={{headerTitle: 'Datenschutzbestimmungen'}} />
8   <Stack.Screen name='ChangePassword' component={ChangePasswordScreen}
9     options={{headerTitle: 'Passwort_aendern'}}/>
10  <Stack.Screen name='UseRights' component={UsingRightsScreen}
11  options={{headerTitle: 'Nutzungsbedingungen'}}/>
12  <Stack.Screen name='CreateEvent' component={FormFreetime} options={{
13    headerTitle: 'Event_anmelden'}}/>
14  <Stack.Screen name='CreateFacility' component={SignUpEducation}
15    options={{headerTitle: 'Bildungseinrichtung_registrieren'}}/>
16  <Stack.Screen name='VerifiyEmail' component={VerifiyEmail} options
17    ={{headerTitle: 'Email_verifizieren'}}/>
18  <Stack.Screen name='LoginSettingsNav' component={LoginNavigation}
19    options={{ headerShown: false }}/>
20 </Stack.Navigator>

```

Quellcode 4.45: Beispiel Navigator

Im gezeigten Beispiel haben wir einen 'Navigator' mit 'initialRouteName' und dem Wert 'Settings'. Das bedeutet, dass beim Verweisen auf den Stack zuerst die Account-Seite angezeigt wird.

Die 'Screen'-Komponenten enthalten wieder die einzelnen Seiten, zu denen navigiert werden kann, sowie bestimmte Optionen, um sie anzupassen. In diesem Beispiel haben wir die Option 'headerShown: false', was bedeutet, dass auf dieser Seite kein automatisch generierter Header angezeigt wird. Darüber hinaus gibt es die Option 'headerTitle', deren zugewiesener Wert im automatisch generierten Header angezeigt wird.

Die Navigation zwischen den verschiedenen Seiten erfolgt nun jedoch anders als zuvor über eine Navigationsleiste. Um hier zwischen den Seiten navigieren zu können, muss durch eine bestimmte Aktion eine Funktion ausgeführt werden, die auf den 'name'-Parameter einer 'Screen'-Komponente zeigt.

Beispiel dafür ist:

```

1 <Pressable style={styles.container} onPress={() => navigation.navigate('
2   DataRights')}>
3   <Text style={styles.textstyle}>{'Datenschutzbestimmungen'}</Text>
4 </Pressable>

```

Quellcode 4.46: Beispiel einer Komponente welche eine Navigation auslöst

In diesem Beispiel wird durch Klicken auf den Text 'Datenschutzbestimmungen' die 'onPress'-Funktion der Komponente 'Pressable' ausgeführt. Diese navigiert über 'navigation.navigate('DataRights')' zu der 'Screen'-Komponente mit dem Wert 'name='DataRights''.

Auf dem Endgerät würde das so aussehen:

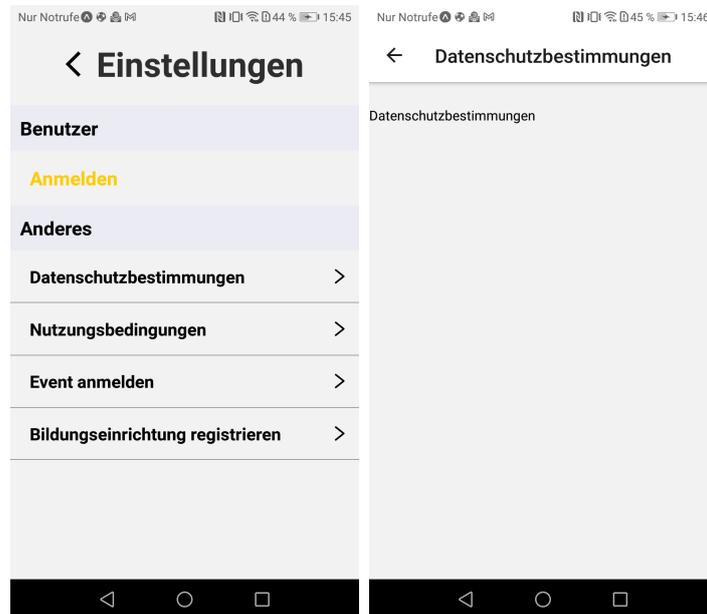


Abbildung 4.3.: Veranschaulichung Stack Navigator

4.8. Speicherung von Daten in React Native - SPIN

Es kann vorkommen, dass Daten für längere Zeit gespeichert werden müssen, zum Beispiel, um sich nicht bei jedem Start der App neu anmelden zu müssen. Dafür stellt React Native die Bibliothek 'react-native-async-storage' zur Verfügung. Mit dieser lassen sich Daten auf dem lokalen Speicher des Geräts speichern und lesen.

4.8.1. Syntax und Grundlagen

Async Storage speichert Daten als 'Key-Value'-Paare ab. Das bedeutet, dass man dem abzuspeichernden Wert einen Schlüssel zuweist, mit dem der Datensatz gespeichert, gelesen und verändert werden kann.

Um React Native Async Storage zu verwenden, muss man es zunächst dem Projekt hinzufügen. Dies geschieht über den npm (Node Packet Manager):

```
1 npm install @react-native-async-storage/async-storage
```

Quellcode 4.47: Beispiel Installation AsyncStorage

Anschließend muss es auf der jeweiligen Seite in den 'imports' hinzugefügt werden:

```
1 import AsyncStorage from '@react-native-async-storage/async-storage';
```

Quellcode 4.48: Beispiel Importieren AsyncStorage

4.8.1.1. Speichern von Daten

Um Daten mit Async Storage zu speichern, gibt es die Funktion 'setItem()'. Ein Beispiel:

```
1 const storeData = async (value) => {  
2   try {  
3     await AsyncStorage.setItem('Key', value)  
4   } catch (e) {  
5     Console.log(e);  
6   }  
7 }
```

Quellcode 4.49: Beispiel speichern mit AsyncStorage

In diesem Beispiel speichert die Funktion 'storeData' den übergebenen Wert unter dem Schlüssel 'Key'. Bei einem Fehler wird die Fehlermeldung in der Konsole ausgegeben.

4.8.1.2. Lesen von Daten

Um Daten mit Async Storage zu lesen, gibt es die Funktion 'getItem()'. Ein Beispiel:

```
1 const getData = async () => {  
2   try {  
3     const value = await AsyncStorage.getItem('Key')  
4     if(value !== null) {  
5       return(value);  
6     }  
7   } catch(e) {  
8     console.log(e);  
9   }  
10 }
```

Quellcode 4.50: Beispiel lesen mit AsyncStorage

In diesem Beispiel liest die Funktion 'getData' den Wert mit dem Schlüssel 'Key' vom lokalen Speicher. Wenn dies erfolgreich ist, wird der gespeicherte Wert zurückgegeben. Sollte der Schlüssel 'Key' nicht existieren, wird nichts zurückgegeben. Bei einem Fehler wird die Fehlermeldung in der Konsole ausgegeben.

5. Installation

5.1. Installation der Laravel Applikation - KAUF

5.1.1. Voraussetzungen

- Installation von PHP Version 8.0 oder höher
- Installation von PHP Extensions
 - BCMath PHP Extension
 - Ctype PHP Extension
 - Fileinfo PHP extension
 - JSON PHP Extension
 - Mbstring PHP Extension
 - OpenSSL PHP Extension
 - PDO PHP Extension
 - Tokenizer PHP Extension
 - XML PHP Extension
- NPM Installation
- Git Installation
- Composer Installation
- MySQL Server
- E-Mail Server
- Quellcode der Anwendung

5.1.2. Dependencies installieren

Nach dem man sich in das Root-Verzeichnis des Projekts begeben hat, kann man damit anfangen, die Dependencies zu installieren.

5.1.2.1. Installation aller composer dependencies

```
1 composer install
```

5.1.2.2. Installation aller NPM dependencies

```
1 npm install
```

5.1.3. Datenbank

5.1.3.1. Angabe des MySQL-Server

Innerhalb von .env:

```
1 DB_CONNECTION=mysql
2 DB_HOST=<IP>
3 DB_PORT=<PORT>
4 DB_DATABASE=<DATABASE>
5 DB_USERNAME=<USER>
6 DB_PASSWORD=<PASSWORD>
```

5.1.3.2. Migration der Datenbank

```
1 php artisan migrate:--fresh
```

5.1.4. Angabe des E-Mail Servers

Innerhalb von .env:

```
1 MAIL_MAILER=<MAIL PROTOKOLL>
2 MAIL_HOST=<HOST>
3 MAIL_PORT=<PORT>
4 MAIL_USERNAME=<MAIL ADRESS>
5 MAIL_PASSWORD=<PASSWORD>
6 MAIL_ENCRYPTION=<ENCRYPTION>
7 MAIL_FROM_ADDRESS="<MAIL_ADRESS>"
8 MAIL_FROM_NAME="${APP_NAME}"
```

5.1.5. Run development environment

5.1.5.1. Ausführung des NPM Development Skripts

```
1 npm run dev
```

5.1.5.2. Ausführung der Applikation

```
1 php artisan serve
```

5.1.5.3. Besuchen der Applikation

Nun ist es möglich, die Applikation in einem Webbrowser unter <http://127.0.0.1:8000/> aufzurufen.

5.1.6. Deployment

Um das Projekt auf einem Webserver zu deployen, benötigt es einen konfigurierten Webserver, der PHP ausführen kann. Hier empfiehlt sich Nginx mit PHP-FPM, dies ist jedoch nicht verpflichtend. Außerdem ist zu versichern, dass PHP-FPM die Rechte hat, die es benötigt, um alle Aktionen, wie zum Beispiel Uploads, die im Rahmen der Applikation durchgeführt werden, durchzuführen.

Unsicher:

```
1 chmod -R 777
```

5.1.6.1. Änderung des app environments

In .env:

```
1 APP_ENV=production
```

5.1.6.2. Ausführung des NPM Build Skripts

```
1 npm run build
```

5.1.6.3. Besuchen der Applikation

Nun ist es möglich den "production build" der Applikation in einem Webbrowser aufzurufen.

5.2. Installation der Mobilapplikation - SPIND

5.2.1. Voraussetzungen

- Node.js v18.13.0 oder höher
- Lauffähige Version der Laravel Webapplikation von Kaufmann David

5.2.2. Dependencies installieren

Nachdem man sich in das Root-Verzeichnis des Projekts begeben hat, kann man damit anfangen, die Dependencies zu installieren.

```
1 npm install
```

5.2.3. Anpassen der API

In jeder Komponente, die mit der API kommuniziert, ist eine konstante Variable namens 'Server' vorhanden, die die Serveradresse enthält. Diese Variable sollte entsprechend der Adresse des gewünschten Ziel-Servers angepasst werden.

Beispiel für die Definition der Server-Adresse in einer Komponente:

```
1 const server = 'Zieladresse';
```

Die obige Codezeile zeigt die Definition einer Konstanten namens 'server', die die gewünschte Zieladresse enthält. Dieser Wert sollte angepasst werden, um den korrekten Server für die API-Kommunikation anzusprechen.

5.2.4. Ausführen der Applikation

Starten des Programmes mit NPM:

```
1 npm start
```

5.2.5. Verbinden über Expo Go

Nach dem Ausführen des Programms erscheint ein QR-Code in der Konsole. Durch Scannen dieses Codes mit dem Endgerät, kann das Programm in der Expo GoApp getestet werden.

6. Zusammenfassung und Ausblick

Zusammenfassend war diese Diplomarbeit ein sehr lehrreiches Projekt, bei dem wir viele neue Erfahrungen gemacht haben.

6.1. Kaufmann David

In dieser Diplomarbeit wurde analysiert, welche Technologien geeignet sind, um eine moderne Applikation zu programmieren, welche eine REST API für Kommunikation mit externen Programmen bietet, und eine Webseite darstellt, mit der Endnutzer die Applikation aufrufen können, und inwiefern die Umsetzung jener Applikation realisiert werden kann.

Die Untersuchung hat gezeigt, dass es eine Vielzahl an Lösungsansätzen für die Umsetzung einer modernen Applikation gibt, und dass die Entscheidung zwischen den jeweiligen Technologien nicht von einfacher Natur ist.

Mithilfe der Untersuchung wurde eine Antwort auf die zu Beginn der Arbeit angeführte Forschungsfrage gefunden. Diese lautete: Wie sieht ein optimales Prozedere aus, um eine Webapplikation mit Administratorfunktionen, welche außerdem eine API für Kommunikation mit externen Applikationen zur Verfügung stellt, zu programmieren und wie wird diese implementiert?

Sie kann wie folgt beantwortet werden: Als geeignete Technologien für den Einsatz in einer modernen Applikation, welche eine REST API für Kommunikation mit externen Programmen bietet und eine Webseite darstellt, mit der Endnutzer die Applikation aufrufen können, stellten sich folgende Technologien heraus:

- Laravel
- Laravel Sanctum
- Tailwind
- Bootstrap
- MySQL
- Inertia.js

Die Frage, wie diese moderne Applikation umgesetzt werden kann, wurde innerhalb der Entwicklung des Prototyps beantwortet.

Weiterführende Forschungen könnten an die gesammelten Ergebnisse und Erkenntnisse der Untersuchung anknüpfen, indem die Applikation erweitert wird, und Lösungsansätze der Umsetzung von neuer Funktionalität gesucht werden.

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie Ausbildungsschwerpunkt: Medientechnik

6.2. Spindelberger Johannes

In dieser Studie wurde untersucht, welche Technologien am besten geeignet sind, um eine mobile Anwendung zu entwickeln, die über eine REST-API mit einer Laravel-Webanwendung interagiert. Darüber hinaus wurde erörtert, wie die ausgewählte Technologie am effektivsten eingesetzt werden kann, um dem Endbenutzer die beste Benutzererfahrung auf seinem Gerät zu bieten.

Die Analyse zeigte, dass es eine Vielzahl von Lösungsansätzen gibt, die je nach Zielplattform variieren können. Durch die Untersuchung wurde eine Antwort auf die zu Beginn der Arbeit gestellte Forschungsfrage gefunden: Wie sieht eine optimale Vorgangsweise und Umsetzung aus, um eine Mobilapplikation, welche mit einer externen API kommuniziert, zu programmieren und diese zu implementieren?

Die Antwort auf die Forschungsfrage lautet wie folgt: Eine geeignete Technologie ist das Framework React Native. Für Anfänger wird empfohlen, sich das Framework Expo anzuschauen, das eine große Hilfe bietet, um das React Native-Programm auf dem eigenen Gerät mit der App Expo Go zu testen. Darüber hinaus bietet die Drittanbieter-Bibliothek Axios eine ausgezeichnete Möglichkeit, in seinem Programm mit einer REST-API zu kommunizieren. Folgende Technologien wurden in dieser Studie verwendet:

- Expo
- React Native
- Axios

Die Frage, wie diese mobile Anwendung implementiert werden kann, wurde in dieser Untersuchung beantwortet. Zukünftige Forschungen könnten auf den gesammelten Ergebnissen und Erkenntnissen dieser Studie aufbauen, indem die Anwendung erweitert und Lösungsansätze für die Umsetzung von und mit neuer Funktionalität untersucht werden.

I. Literaturverzeichnis

- [1] *Laravel Database*. Online in Internet: URL: <https://laravel.com/docs/9.x/database>.
- [2] *SQL Server Developer Edition*. Online in Internet: URL: <https://www.sqlshack.com/sql-server-developer-edition-a-faq-article/>.
- [3] *MongoDB Licensing*. Online in Internet: URL: <https://www.mongodb.com/community/licensing>.
- [4] *DB-Engine Ranking*. Online in Internet: URL: <http://db-engines.com/de/ranking>.
- [5] *Abgestimmte Erwerbsstatistik 2020*, Juli 2022. Online in Internet: URL: <https://www.statistik.at/blickgem/ae6/g30101.pdf>.
- [6] *PHP und Node.js Vergleich*, April 2018. Online in Internet: URL: <https://www.mindtwo.de/blog/php-vs-node-js-vergleich>.
- [7] *Inertia.js Beschreibung*. Online in Internet: URL: <https://inertiajs.com/>.
- [8] Zelezny, Lukasz: *SSR vs CSR*, Juni 2022. Online in Internet: URL: <https://seo.london/de/server-seitig-vs-client-seitig-rendering/>.
- [9] *Next.js GitHub Repository*. Online in Internet: URL: <https://github.com/vercel/next.js/>.
- [10] *Laravel GitHub Repository*. Online in Internet: URL: <https://github.com/laravel/framework>.
- [11] *Svelte Beschreibung*. Online in Internet: URL: [https://de.wikipedia.org/wiki/Svelte_\(Framework\)](https://de.wikipedia.org/wiki/Svelte_(Framework)).
- [12] Konstantinidis, Antony: *JSX-Komponenten mit Java nutzen*, September 2020. Online in Internet: URL: <https://vuejs.de/artikel/vuejs-tutorial-deutsch-anfaenger/>.
- [13] Konstantinidis, Antony: *Inertia.js Beschreibung*, März 2023. Online in Internet: URL: <https://vuejs.de/artikel/vuejs-tutorial-deutsch-anfaenger/>.
- [14] *Entwickeln für IOS*. Online in Internet: URL: <https://www.impuls1.de/ios-app-programmieren/#voraussetzungen>.
- [15] *Java Beschreibung*. Online in Internet: URL: [https://de.wikipedia.org/wiki/Java_\(Programmiersprache\)](https://de.wikipedia.org/wiki/Java_(Programmiersprache)).
- [16] *Anforderungen Google Play Store*. Online in Internet: URL: <https://www.ionos.at/digitalguide/websites/web-entwicklung/die-eigene-app-entwickeln-android-app-veroeffentlichen/>.
- [17] *React Native Dokumentation*. Online in Internet: URL: <https://reactnative.dev/>.
- [18] *React Native Beschreibung*. Online in Internet: URL: <https://www.tenmedia.de/de/glossar/react-native>.
- [19] *Virtual Dom Beschreibung*. Online in Internet: URL: <https://legacy.reactjs.org/docs/faq-internals.html>.
- [20] *Flutter Beschreibung*. Online in Internet: URL: <https://dev.to/sudarasach/intro-to-flutter-2odk>.
- [21] *Skia Beschreibung*. Online in Internet: URL: <https://skia.org/docs/>.

- [22] *Xamarin Geschichte*. Online in Internet: URL: <https://www.infoq.com/articles/mobile-cross-platform-xamarin/>.
- [23] *Xamarin Funktionsweise*. Online in Internet: URL: <https://learn.microsoft.com/de-de/xamarin/get-started/what-is-xamarin>.
- [24] *Xamarin Beschreibung*. Online in Internet: URL: <https://visualstudio.microsoft.com/de/xamarin/>.
- [25] *Expo Beschreibung*. Online in Internet: URL: <https://expo.dev/>.
- [26] *Typescript Beschreibung*. Online in Internet: URL: <https://www.typescriptlang.org/>.
- [27] *XMLHttpRequest Beschreibung*. Online in Internet: URL: <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>.
- [28] *React HTTP-Routing*. Online in Internet: URL: <https://reactnative.dev/docs/network>.
- [29] *Axios Beschreibung*. Online in Internet: URL: <https://axios-http.com/>.
- [30] *Funktionsweise von Axios*. Online in Internet: URL: https://axios-http.com/docs/api_intro.
- [31] Martin: *Laravel Model erklärt*, Januar 2021. Online in Internet: URL: <https://www.seotheater.de/2021/01/26/laravel-model-kurz-mal-erklart/>.
- [32] *Laravel Schema Builder*. Online in Internet: URL: <https://laravel.com/docs/5.0/schema>.
- [33] *Laravel Routing: Übersicht*. Online in Internet: URL: <https://www.a-coding-project.de/ratgeber/laravel/routing>.
- [34] *Middleware - Laravel*. Online in Internet: URL: <https://runebook.dev/de/docs/laravel/docs/8.x/middleware>.
- [35] *Laravel Contracts*. Online in Internet: URL: <https://laravel.com/docs/9.x/contracts>.
- [36] Maxham, Tobias: *Laravel Mailables*, September 2016. Online in Internet: URL: <https://blog.maxham.de/2016/09/laravel-mailables/>.
- [37] *Laravel Blade*, September 2019. Online in Internet: URL: <https://www.wissensarchiv.org/doku.php?id=laravel:blade>.
- [38] *Laravel Authentication*. Online in Internet: URL: <https://laravel.com/docs/9.x/authentication>.
- [39] *OAuth*. Online in Internet: URL: <https://de.wikipedia.org/wiki/OAuth>.
- [40] *Laravel Passport*. Online in Internet: URL: <https://laravel.com/docs/9.x/passport>.
- [41] *Was ist tokenbasierte Authentifizierung?*, Februar 2023. Online in Internet: URL: <https://www.okta.com/de/identity-101/what-is-token-based-authentication/>.
- [42] Careaga, Andrew: *Eye-tracking studies: first impressions form quickly on the web*, Februar 2012. Online in Internet: URL: https://news.mst.edu/2012/02/eye-tracking-studies_show_firs/.

- [43] Golitschek, Nadja: *Die Website barrierefrei(er) gestalten*, September 2021. Online in Internet: URL: <https://medienkompass.de/barrierefreiheit-website-erstellen/>.
- [44] *React Native API*. Online in Internet: URL: <https://reactnative.dev/docs/components-and-apis>.
- [45] *React Native Navigation Beschreibung*. Online in Internet: URL: <https://reactnavigation.org/docs/bottom-tab-navigator/>.
- [46] *Funktionsweise von React Native Stack Navigator*. Online in Internet: URL: <https://reactnavigation.org/docs/stack-navigator/>.
- [47] *Swift Beschreibung*. Online in Internet: URL: <https://www.apple.com/de/swift/#:~:text=Swift%20ist%20eine%20robuste%20und,ist%20benutzerfreundlich%20und%20open%20Source>.
- [48] *DataGrip Beschreibung*. Online in Internet: URL: <https://www.capterra.at/software/1021524/datagrip>.
- [49] *Funktionsweise von React Native Async-Storage*. Online in Internet: URL: <https://react-native-async-storage.github.io/async-storage/docs/usage>.

II. Abbildungsverzeichnis

3.1. Funktionsweise React Native API	22
3.2. Funktionsweise Xamarin	24
4.1. ER Diagramm	35
4.2. Veranschaulichung der Bottom Navigation Bar	74
4.3. Veranschaulichung Stack Navigator	76

III. Tabellenverzeichnis

A.1. Arbeitstagebuch Kaufmann	93
A.2. Arbeitstagebuch Spindelberger	94

IV. Quellcodeverzeichnis

3.1. Beispiel eines JSX basierten Codes zum Festellen der Code Komplexität . . .	22
3.2. Beispiel eines Dart basierten Codes zum Festellen der Code Komplexität . . .	23
3.3. Beispiel eines Xamarin basierten Codes zum Festellen der Code Komplexität	25
3.4. Beispiel eines Javascript Codes	27
3.5. Beispiel eines Typescript Codes	27
3.6. Beispiel einer GET-Anfrage mit XMLHttpRequest	28
3.7. Beispiel einer POST-Anfrage mit XMLHttpRequest	29
3.8. Beispiel einer PUT-Anfrage mit XMLHttpRequest	29
3.9. Beispiel einer DELETE-Anfrage mit XMLHttpRequest	30
3.10. Beispiel einer GET-Anfrage mit FETCH	30
3.11. Beispiel einer POST-Anfrage mit FETCH	30
3.12. Beispiel einer PUT-Anfrage mit FETCH	31
3.13. Beispiel einer DELETE-Anfrage mit FETCH	31
3.14. Beispiel einer GET-Anfrage mit Axios	32
3.15. Beispiel einer POST-Anfrage mit Axios	32
3.16. Beispiel einer PUT-Anfrage mit Axios	33
3.17. Beispiel einer DELETE-Anfrage mit Axios	33
4.1. Model-Template	38
4.2. Facility.php	38
4.3. Facility migration	40
4.4. Aufruf der Create-Methode von School	41
4.5. Überschreiben der Funktionalität der Create-Methode von School	41
4.6. Route gibt Inertia-Seite mit Eingabeparametern zurück	44
4.7. Logout-Funktion im Account-Controller	45
4.8. Struktur eines Traits	46
4.9. Einbindung eines Traits in einen Controller	46
4.10. Aufruf von Trait-Funktionalität innerhalb eines Controllers	46
4.11. \$middleware in app/Http/Kernel.php	47
4.12. \$middlewareGroups in app/Http/Kernel.php	47
4.13. \$routeMiddleware in app/Http/Kernel.php	48
4.14. app/Http/Middleware/Web/BookmarkOwner.php	48
4.15. handle-Methode einer Middleware mit Argumenten	49
4.16. Zuweisung einer Middleware für mehrere Routes	49
4.17. Zuweisung einer Middleware für eine einzelne Route	49
4.18. Zuweisung einer Middleware mit Argumenten	49
4.19. Implementation von MustVerifyEmail in Account-Model	52
4.20. Implementation der benötigten Klassen von MustVerifyEmail im Account-Model	53
4.21. Implementation der benötigten Klassen von CanResetPassword im Account-Model	54
4.22. Nutzung der CanResetPassword-Funktionalität in einem Controller	55
4.23. toMail-Methode von ResetPasswordNotification.php	56
4.24. resetUrl-Methode von ResetPasswordNotification.php	56
4.25. Konstruktor von ResetPasswordNotification.php	56
4.26. envelope-Funktion von app/Mail/ResetPassword.php	57
4.27. content-Funktion von app/Mail/ResetPassword.php	57

4.28. Script-Komponente von Error.svelte	63
4.29. svelte:head-Komponente von Error.svelte	64
4.30. svelte:Body von Error.svelte	64
4.31. share-Funktion von HandleInertiaRequets.php	66
4.32. Beispiel einer JSX Komponente	67
4.33. Beispiel von Import Statements	67
4.34. Beispiel einer Funktion in React Native	67
4.35. Beispiel eines Export Statements	68
4.36. Beispiel einer React Native Seite	68
4.37. Beispiel für impotieren einer anderen Seite	69
4.38. Beispiel Verwendung impotierter Seiten	69
4.39. Beispiel für Importieren wie es im Projekt verwendet wurde	71
4.40. Beispiel NavigationContainer	71
4.41. Beispiel Navigator	71
4.42. Beispiel Screen	72
4.43. Beispiel einer Navigationsleiste wie sie im Projekt angewendet wurde	72
4.44. Beispiel für Importieren wie es im Projekt verwendet wurde	74
4.45. Beispiel Navigator	75
4.46. Beispiel einer Komponente welche eine Navigation auslöst	75
4.47. Beispiel Installation AsyncStorage	77
4.48. Beispiel Impotieren AsyncStorage	77
4.49. Beispiel speichern mit AsyncStorage	77
4.50. Beispiel lesen mit AsyncStorage	78

V. Abkürzungsverzeichnis

Abkürzung	Bedeutung
API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
CSS	Cascadian Style Sheet
DDL	Data Definition Language
DOM	Document Object Model
DX	Developer Experience
E-Mail	Electronic Mail
ER-Diagramm	Entity-Relationship-Diagramm
FastCGI	Fast Common Gateway Interface
FK	Foreign Key
FPM	FastCGI Process Manager
HTTP	Hypertext Markup Protocol
HTTPd	Apache HTTP Server
HTML	Hypertext Markup Language
HTL	Höhere Technische Lehranstalt
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
ID	Identifikator
JSX	JavaScript XML
MVC	Model View Controller
NPM	Node Package Manager
OAuth	Open Authorization
ORM	Object Relational Mapping
PDF	Portable Document Format
PK	Primary Key
PHP	Hypertext Preprocessor
REST	Representational State Transfer
RTE	Runtime Environment
SMS	Short Message Service
SPA	Single Page Application
SQL	Structured Query Language
SSH	Secure Shell
SSPL	Server Side Public License
SSR	Server Side Rendering
UI	User Interface
URL	Unified Resource Locator
UX	User Experience
XML	Extensible Markup Language
NPM	Node Package Manager
JSX	JavaScript Extensible Markup Language

A. Anhang

A.1. Projektstagebücher

A.1.1. Projektstagebuch David Kaufmann

Kalenderwoche	Zeit	kumulativ	Fortschritt
KW 39 - Sep./Okt.	5h	5h	Themenstellung
KW 40 - Okt.	4h	9h	Themenstellung
KW 41 - Okt.	5h	14h	Pflichtenheft
KW 42 - Okt.	4h	18h	Pflichtenheft
KW 44 - Okt./Nov.	5h	23h	Pflichtenheft
KW 46 - Nov.	5h	28h	Pflichtenheft
KW 47 - Nov.	7h	35h	Pflichtenheft, Koordination mit Stadt
KW 48 - Nov./Dez.	2h	37h	Hardware
KW 49 - Dez.	5h	42h	Hardware
KW 50 - Dez.	8h	50h	Pflichtenheft, Koordination mit Stadt
KW 2 - Jan.	6h	56h	Technologie Einarbeitung
KW 3 - Jan.	6h	62h	Technologie Einarbeitung
KW 4 - Jan.	7h	69h	Datenbank, Technologie Einarbeitung
KW 5 - Jan./Feb.	15h	84h	Datenbank, Programmierung
KW 6 - Feb.	30h	114h	Programmierung
KW 7 - Feb.	17h	131h	Programmierung
KW 8 - Feb.	9h	140h	Bugfixes
KW 9 - Feb./Mrz.	12h	152h	Bugfixes, Koordination mit Stadt, Hardware
KW 10 - Mrz.	7h	159h	Bugfixes, Hardware
KW 11 - Mrz.	10h	169h	Diplomarbeit schreiben
KW 12 - Mrz.	14h	183h	Diplomarbeit schreiben
KW 13 - Mrz.	9h	192h	Diplomarbeit schreiben

Tabelle A.1.: Arbeitstagebuch Kaufmann

A.1.2. Projekttagbuch Johannes Spindelberger

Kalenderwoche	Zeit	kumulativ	Fortschritt
KW 39 - Sep./Okt.	5h	5h	Themenstellung
KW 40 - Okt.	4h	9h	Themenstellung
KW 41 - Okt.	5h	14h	Plichtenheft
KW 42 - Okt.	4h	18h	Plichtenheft
KW 44 - Okt./Nov.	6h	24h	Plichtenheft
KW 46 - Nov.	5h	29h	Plichtenheft
KW 47 - Nov.	7h	36h	Plichtenheft, Koordination mit Stadt
KW 48 - Nov./Dez.	6h	42h	Hardware
KW 50 - Dez.	6h	48h	Plichtenheft, Koordination mit Stadt
KW 1 - Jan.	9h	57h	Einarbeitung
KW 2 - Jan.	10h	67h	Programmierung
KW 3 - Jan.	11h	78h	Programmierung
KW 5 - Jan./Feb.	10h	88h	Programmierung
KW 6 - Feb.	12h	100h	Programmierung
KW 7 - Feb.	13h	113h	Programmierung
KW 8 - Feb.	8h	121h	Programmierung
KW 9 - Feb./Mrz.	13h	134h	Programmierung
KW 10 - Mrz.	11h	145h	Bugfixes
KW 11 - Mrz.	15h	160h	Diplomarbeit schreiben
KW 12 - Mrz.	14h	174h	Diplomarbeit schreiben, Bugfixes
KW 13 - Mrz./Apr.	10h	184h	Bugfixes, Fertigstellung Diplomarbeit, Code Übergabe

Tabelle A.2.: Arbeitstagebuch Spindelberger

A.2. Datenträgerbeschreibung

- /jugendliches-krems.pdf: Schriftliche Diplomarbeit in PDF-Format
- /pflichtenheft.pdf: Das Pflichtenheft in PDF-Format
- /latex/: Verzeichnis vom Latex-Projekt der schriftlichen Diplomarbeit
- /api-web-app/: Verzeichnis vom Software-Projekt der Laravel-Applikation (KAUF)
- /mobile-app/: Verzeichnis vom Software-Projekt der Mobilapplikation (SPIN)
- /api-web-app-installation.pdf: Installationsanleitung für die Laravel-Applikation in PDF-Format (KAUF)
- /mobile-app-installation.pdf: Installationsanleitung für die Mobilapplikation in PDF-Format (SPIN)
- /dashboard-handbuch.pdf: Handbuch für das Dashboard in PDF-Format (KAUF)
- /webseite-handbuch.pdf: Handbuch für die Webseite in PDF-Format (KAUF)
- /mobilapp-handbuch.pdf: Handbuch für die Mobilapplikation in PDF-Format (SPIN)