

DIPLOMARBEIT

Gesamtprojekt

Timetracking

iOS App

Philipp Wudernitz 5AHEL

Android App

David Ensbacher 5AHEL

Web App

Marcel Hacker 5AHEL

Backend

Thomas Zeitlberger 5AHEL

Betreuer

Dipl.-Ing. Franz Geischläger

Schuljahr 2020/21

Abgabevermerk:

Datum: 20.04.2021

übernommen von:

Höhere Technische Bundeslehranstalt Hollabrunn

Höhere Lehranstalt für Elektronik und Technische Informatik

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig 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.

Philipp Wudernitz

David Ensbacher

Marcel Hacker

Thomas Zeitlberger

HINWEISE

Die vorliegende Diplomarbeit wurde in Zusammenarbeit mit der Firma Tailored Apps ausgeführt.

Die in dieser Diplomarbeit entwickelten Prototypen und Software-Produkte dürfen ganz oder auch in Teilen von Privatpersonen oder Firmen nur dann in Verkehr gebracht werden, wenn sie diese selbst geprüft und für den vorgesehenen Verwendungszweck für geeignet befunden haben.

Es wird keinerlei Haftung übernommen für irgendwelche Schäden, die aus der Nutzung der hier entwickelten oder beschriebenen Bestandteile des Projekts resultieren.

Für alle Entwicklungen gilt die GNU General Public License [<http://www.gnu.org/licenses/gpl.html>] der Free Software Foundation, Boston, USA in der Version 3.

Die Diplomarbeit erfüllt die "Standards für Ingenieur- und Technikerprojekte" entsprechend dem Rundschreiben Nr. 60 aus 1999 des BMBWK (GZ.17.600/101-II/2b/99).

[https://www.bmb.gv.at/ministerium/rs/1999_60.html]

SCHLÜSSELBEGRIFFE

- NFC-Tags
 - Auf NFC-Tags kann ein beliebiger Text (Format ist zu beachten) geschrieben werden, welcher von einem NFC Reader oder den Mobile-Apps gelesen werden kann.
- Bluetooth-Beacons
 - Ein Gerät (oder auch Smartphone mit Simulations-App), welches ein Bluetooth LE (Low Energy) Signal aussendet und beispielsweise von den Mobile-Apps gelesen werden kann.
- Geofencing
 - Ein virtueller Zaun wird mithilfe von Koordinaten (Längen- und Breitengraden) abgesteckt. Wird dieser Bereich betreten bekommen die Mobile-Apps dies mit.
- Mobile-App
 - Die Smartphone-Apps, geschrieben in Swift für iOS und in Kotlin für Android.
- Web-App
 - Eine App, welche mithilfe eines Browsers erreicht werden kann und zusammen mit dem Backend am eigenen Web Server gehostet wird.
- Backend
 - Das Backend, welches Node.js als Framework verwendet, verarbeitet und speichert die gesammelten Daten.
- Datenbank
 - Die Datenbank, welche mithilfe von MySQL-Workbench entwickelt wurde, dient der Datenspeicherung und Datenverwaltung.
- Redmine
 - Eine flexible Projektverwaltungsapplikation im Web, in der die Verwaltung der Mitarbeiterdaten von Tailored Apps durchgeführt wird.

DANKSAGUNGEN

Wir möchten uns recht herzlich bei Tailored Apps, insbesondere bei Martin Zehetner MSc, Alexander Grafl MSc, Mario Hahn und Nadine Christ BSc für die fachliche Unterstützung bei der Entwicklung der Apps bedanken.

Nicht zuletzt möchten wir uns bei unserem Betreuer Dipl.-Ing. Franz Geischläger, der uns tatkräftig während der gesamten Diplomarbeit unterstützt hat, organisatorisch, wie auch fachlich, bedanken.

DIPLOMARBEIT

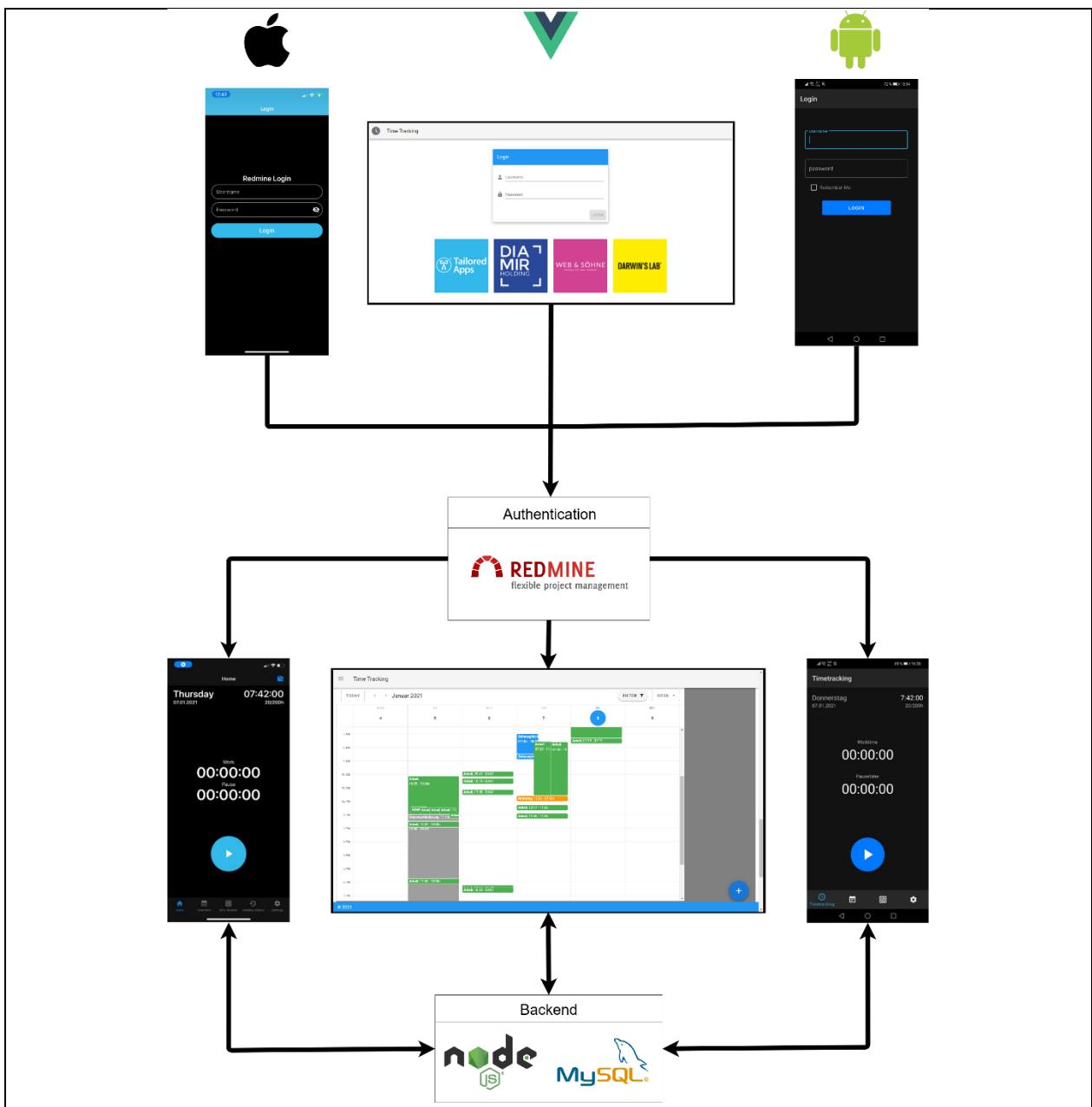
DOKUMENTATION

Namen der Verfasser/innen	David Ensbacher Marcel Hacker Philipp Wudernitz Thomas Zeitlberger
Jahrgang Schuljahr	5AHEL 2020/21
Thema der Diplomarbeit	Timetracking
Kooperationspartner	Tailored Apps

Aufgabenstellung	Die Aufgabe ist Smartphone-Apps und eine Web-App für die Mitarbeiter von Tailored Apps zu entwickeln, welche mithilfe eines gemeinsamen Backends verbunden werden. Die Aufgabenverteilung besteht darin, dass je ein Schüler für die Entwicklung der Android-App, iOS-App, Web-App und des Backends verantwortlich ist. Die Timetracking-App soll die Arbeitszeiten der Mitarbeiter mithilfe von NFC-Tags, Bluetooth-Beacons oder Geofencing tracken und diese aufbereitet in der Web-App und den Smartphone-Apps darstellen. Die Web-App soll ebenfalls eine Adminoberfläche bieten, welche Datensätze erstellen und manipulieren kann.
------------------	--

Realisierung	Mittels Mockups und Wireframes wurden die User-Interfaces anfänglich festgelegt. Nachdem die User-Interfaces in der Web-App und den Smartphone-Apps ausprogrammiert wurden, ist die Verknüpfung mit der Datenbank implementiert worden. Zur Authentifizierung wird Redmine verwendet. In den Smartphone-Apps wurden darauffolgend die Tracking-Methoden eingebaut. Das Backend kümmert sich um die Aufbereitung von Daten und sendet diese an die Apps.
--------------	---

Ergebnisse	Es stehen native und intuitive Apps für Android, iOS und dem Web zur Verfügung. Damit ist es möglich, die Arbeitszeiten der Mitarbeiter mithilfe der Tracking-Methoden zu erfassen und Abwesenheiten einzutragen. Weiters können diese in der Web-App verwaltet werden.
------------	---



Teilnahme an Wettbewerben,
Auszeichnungen

Möglichkeiten der
Einsichtnahme in die Arbeit

HTL Hollabrunn
Anton Ehrenfriedstraße 10
2020 Hollabrunn

Approbation
(Datum / Unterschrift)

Prüfer/Prüferin

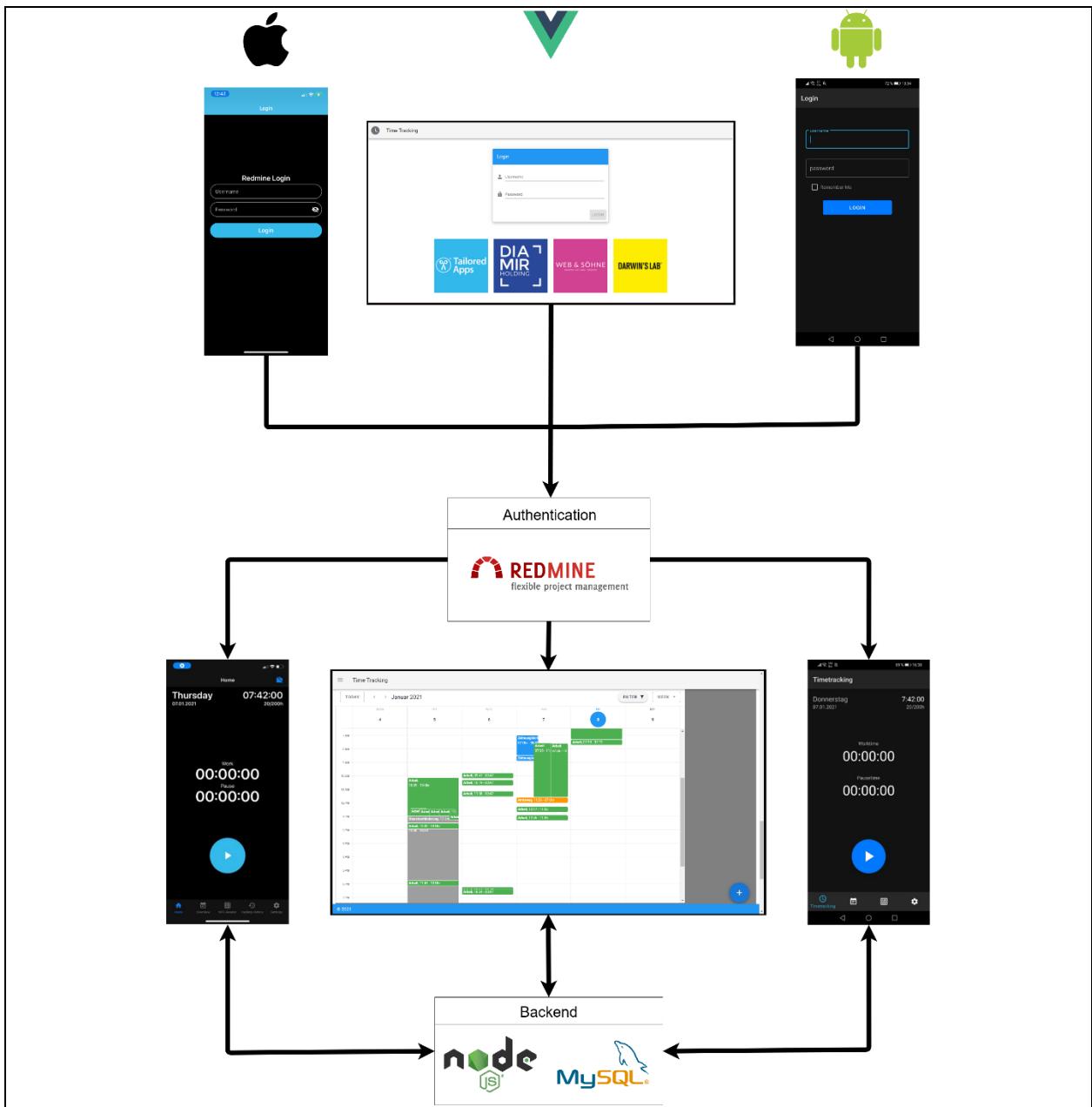
Direktor/Direktorin
Abteilungsvorstand/Abteilungsvorständin

DIPLOMA THESIS

Documentation

Author(s)	David Ensbacher Marcel Hacker Philipp Wudernitz Thomas Zeitlberger
Form Academic year	5AHEL 2020/21
Topic	Timetracking
Co-operation Partners	Tailored Apps

Assignment of Tasks	The task is to develop smartphone apps and a web app for employees of Tailored Apps, which are connected via a shared backend. Every student is responsible for the development of either the Android app, iOS app, web app or backend. The Timetracking app tracks working hours of the employees, using NFC-tags, bluetooth beacons and geofencing. The tracking data is processed in the backend and displayed in the web app and the smartphone apps. Additionally, the web app will provide an admin interface, which is used to create and manipulate data records.
Realisation	By creating mockups and wireframes the user interfaces were established. After coding the user interfaces in the web app and the smartphone apps, the connection to the database was implemented. Redmine was used for authentication. Subsequently, the tracking methods were integrated in the smartphone apps. The backend handles the data and sends them to the apps.
Results	The results are native and intuitive apps for Android, iOS and the web. The apps allow employers to track working hours, enter absences and manage these entries. These entries can be managed in the web app.



Participation in Competitions
Awards

Accessibility of
Diploma Thesis

HTL Hollabrunn
Anton Ehrenfriedstraße 10
2020 Hollabrunn

Approval
(Date / Sign)

Examiner

Head of College / Department

Timetracking

Verlauf

- 21.09.2020 um 18:33 Die Themenstellung "Timetracking" (David Ensbacher, Marcel Hacker, Philipp Wudernitz, Thomas Zeitlberger) wurde eingereicht.
- 24.09.2020 um 09:06 Die Themenstellung "Timetracking" (David Ensbacher, Marcel Hacker, Philipp Wudernitz, Thomas Zeitlberger) wurde vom Betreuer / von der Betreuerin abgelehnt. Dazu wurde folgender Kommentar verfasst: In der Web-App....
- 24.09.2020 um 09:41 Die Themenstellung "Timetracking" (David Ensbacher, Marcel Hacker, Philipp Wudernitz, Thomas Zeitlberger) wurde eingereicht.
- 24.09.2020 um 13:05 Die Themenstellung "Timetracking" (Philipp Wudernitz) wurde vom Betreuer / von der Betreuerin akzeptiert.
- 24.09.2020 um 19:15 Die Themenstellung "Timetracking" (Philipp Wudernitz) wurde vom zuständigen Abteilungsvorstand akzeptiert.
- 06.10.2020 um 16:00 Die Themenstellung "Timetracking" (Philipp Wudernitz) wurde vom Direktor / von der Direktorin genehmigt.

Schule

Höhere technische Bundeslehranstalt HOLLABRUNN

Abteilung(en)

Hauptverantwortlich: Elektronik und Technische Informatik

AV

Hauptverantwortlich: Wilfried Trollmann

Abschließende Prüfung

2021

Betreuer/innen

Hauptverantwortlich: Franz Geischläger

Ausgangslage

Bei Tailored Apps wird zurzeit die Arbeitszeit der Mitarbeiter händisch in einer Tabelle erfasst. Um den Verwaltungsaufwand zu reduzieren und künftig die Auswertung zu vereinfachen ist eine Weblösung mit Appunterstützung angedacht.

Projektteam (Arbeitsaufwand)

Name	Individuelle Themenstellungen	Klasse	Arbeitsaufwand
Philipp Wudernitz	Timetracking iOS App	5AHEL_21	180 Stunden
David Ensbacher	Timetracking Android App	5AHEL_21	180 Stunden
Marcel Hacker	Timetracking Web App	5AHEL_21	180 Stunden
Thomas Zeitlberger	Timetracking Backend	5AHEL_21	180 Stunden

Projektpartner

Tailored Apps (E-Mail: office@tailored-apps.com, www.tailored-apps.com) Heiligenstädterstraße 31/1 1190 Wien, Martin Zehetner, +43 18902845

Untersuchungsanliegen der individuellen Themenstellungen

Zur Zeiterfassung sollen die iOS-App und Android-App eingesetzt werden, wobei David Ensbacher die Android-App unter Verwendung von Android Studio entwickelt und Philipp Wudernitz für die iOS-App Xcode einsetzt. In der Web-App sollen die Arbeitszeiten verwaltet und grafisch dargestellt werden. Marcel Hacker und Thomas Zeitlberger sind für die Web-App verantwortlich, wobei Herr Hacker das Frontend mithilfe von VueJS und Herr Zeitlberger das Backend mithilfe von Node.JS entwickeln wird.

Zielsetzung

Die Mitarbeiter von Tailored Apps können mit ihrem Smartphone NFC-Tags scannen, in Bluetooth Beacon oder Geofence Bereiche eintreten und somit ihre Arbeitszeit, sowie die Pausen, tracken. In der Web-App ist Darstellung und Verwaltung der erfassten Zeiten auf einfache Weise und übersichtlicher Form möglich.

Geplantes Ergebnis der Prüfungskandidatin/des Prüfungskandidaten

Als Ergebnis dieses Projekts stehen den Mitarbeitern von Tailored Apps und Diamir Holding eine iOS- und eine Android-App für die Zeiterfassung zur Verfügung. Die Web-App unterstützt die Mitarbeiter beim Verwalten ihrer Arbeitszeiten und bietet ihnen die Möglichkeit der grafischen Darstellung.

Meilensteine

02.10.2020 Fertigstellung Grunddesign
27.11.2020 Fertigstellung Zeiterfassung
22.01.2021 Fertigstellung Auswertung
19.02.2021 Fertigstellung App, Web-App und Backend
12.03.2021 Fertigstellung Dokumentation

Rechtliche Regelung (mit dem/den Projektpartner/n erfolgt durch)

Dokumente

- [Erklärung_Timetracking.pdf](#)

Erklärung

Die unterfertigten Kandidaten/Kandidatinnen haben gemäß § 34 Abs. 3 Z 1 und § 37 Abs. 2 Z 2 des Schulunterrichtsgesetzes in Verbindung mit den Bestimmungen der „Prüfungsordnung BMHS, Bildungsanstalten“, BGBl. II Nr. 177/2012 i.d.g.F. die Ausarbeitung einer Diplomarbeit/Abschlussarbeit mit folgender Aufgabenstellung gewählt:

Timetracking (Gesamtprojekt)

Individuelle Aufgabenstellungen im Rahmen des Gesamtprojektes:

- David Ensbacher (5AHEL_21): **Timetracking Android-App**
- Marcel Hacker (5AHEL_21): **Timetracking Web-App**
- Philipp Wudernitz (5AHEL_21): **Timetracking iOS-App**
- Thomas Zeitlberger (5AHEL_21): **Timetracking Backend**

Die Kandidaten/Kandidatinnen nehmen zur Kenntnis, dass die Diplomarbeit/Abschlussarbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können, die jedenfalls als solche entsprechend kenntlich zu machen sind.

Die Abgabe der vollständigen Diplomarbeit/Abschlussarbeit hat in digitaler und in zweifach ausgedruckter Form bis spätestens **02.04.2021** beim zuständigen Betreuer/der zuständigen Betreuerin zu erfolgen.

Die Kandidaten/Kandidatinnen nehmen weiters zur Kenntnis, dass ein Abbruch der Diplomarbeit/Abschlussarbeit nicht möglich ist.

Kandidaten/Kandidatinnen:

David Ensbacher (5AHEL_21)

Marcel Hacker (5AHEL_21)

Philipp Wudernitz (5AHEL_21)

Thomas Zeitlberger (5AHEL_21)

Datum und Unterschrift bzw. Handysignatur:15.09.2020 Ensbacher David15.09.2020 Marcel Hacker15.9.2020 P. Zeitlberger15.9.2020 Thomas Zeitlberger

Timetracking Dokumentation

David Ensbacher, Marcel Hacker, Philipp Wudernitz, Thomas Zeitlberger



Inhaltsverzeichnis

1	Lastenheft.....	25
1.1	Zielbestimmung	25
1.2	Projekteinsatz.....	25
1.3	Produktfunktionen.....	25
1.3.1	Allgemeine Funktionen.....	25
1.3.2	Funktionen (Apps).....	25
1.3.3	Funktionen (Web).....	26
1.3.4	Admin-Funktionen (Web)	26
1.4	Produktdaten	26
1.5	Qualitätsanforderungen	26
1.6	Realisierung	26
2	Pflichtenheft.....	27
2.1	Zielbestimmung	27
2.1.1	Musskriterien.....	27
2.1.2	Wunschkriterien	27
2.2	Produkteinsatz.....	28
2.2.1	Anwendungsbereiche.....	28
2.2.2	Zielgruppen	28
2.3	Produktfunktionen.....	28
2.3.1	Allgemeine Funktionen.....	28
2.3.2	Funktionen (Apps).....	28
2.3.3	Funktionen (Web).....	28
2.3.4	Admin-Funktionen (Web)	28
2.4	Produktdaten	29
2.5	Qualitätsbestimmungen	30
3	Verwendete Programme	31
3.1	Mockups.....	31
3.1.1	Balsamiq	31
3.1.2	Xd	31
3.1.3	Sketch.....	31
3.1.4	Overflow.....	31
3.2	IDE	31
3.2.1	Visual Studio Code.....	31
3.2.2	Visual Studio Community	31
3.2.3	Xcode.....	32
3.2.4	Android Studio	32
3.3	Version Control.....	32
3.3.1	Git	32
3.3.2	Github Desktop	32
3.3.3	Tortoise Git	32
3.4	Runtime.....	33
3.4.1	Node.js.....	33
3.5	Storage.....	33



3.5.1	Google Drive	33
3.5.2	Google Backup & Sync	33
3.6	Bildbearbeitung	33
3.6.1	Gimp	33
3.6.2	App Icon Resizer.....	33
3.7	Browser.....	34
3.7.1	Google Chrome.....	34
3.7.2	Firefox.....	34
3.8	Office.....	34
3.8.1	Excel.....	34
3.8.2	Word	34
3.8.3	PowerPoint.....	34
4	Projektplan.....	35
5	Entwicklung der iOS App	36
5.1	Vorbereitung.....	36
5.1.1	Apple ID	36
5.1.2	Developer Account.....	36
5.1.3	Installation Xcode.....	36
5.2	Theoretische Grundlagen	36
5.2.1	Lebenszyklus einer iOS-App	36
5.2.1.1	AppDelegate.....	37
5.2.1.2	SceneDelegate	37
5.2.1.3	ViewController	38
5.2.2	UI-Entwicklung	38
5.2.2.1	Storyboards vs. Programmatically	38
5.2.2.2	UI-Elemente.....	38
5.2.2.3	Constraints	38
5.2.2.4	Human Interface Guidelines	39
5.3	Layout	39
5.3.1	Auto Layout.....	39
5.4	Programmiersprachen	39
5.4.1	Objective-C	40
5.4.2	Swift	40
5.5	Swift	40
5.5.1	Variablen.....	40
5.5.2	Konstanten.....	41
5.5.3	If-Bedingung.....	41
5.5.4	Schleifen	41
5.5.5	Strukturen und Klassen	42
5.5.6	Funktionen	43
5.5.7	Optionals.....	43
5.6	UI-Frameworks	44
5.6.1	UIKit.....	44
5.6.2	SwiftUI	44
5.7	Entwicklungsumgebung.....	45
5.7.1	Erstellen eines Projekts.....	45
5.7.2	Entwicklungsumgebung Xcode	48

5.7.3	Erstellen eines UI's (Programmatically).....	51
5.7.4	App auf iPhone deployen	52
5.8	API-Calls	53
5.8.1	Grundlagen	53
5.8.2	Daten lesen.....	54
5.8.3	Daten schreiben.....	55
5.9	Tracking Methoden.....	56
5.9.1	Grundlagen	56
5.9.2	NFC	56
5.9.3	Bluetooth Beacons (Bluetooth LE)	58
5.9.4	Geofencing.....	61
5.10	User Defaults	62
5.11	CoreData.....	63
5.12	Keychain	68
5.13	User Interface.....	69
5.13.1	Launch Screen.....	70
5.13.2	Login.....	70
5.13.3	Home Screen.....	71
5.13.3.1	Abwesenheiten	71
5.13.4	Überblick.....	72
5.13.5	NFC Reader.....	73
5.13.6	Tracking Verlauf.....	74
5.13.7	Einstellungen	74
5.14	Funktionen	75
5.14.1	Libraries	75
5.14.1.1	Pod-File.....	77
5.14.2	AppDelegate	78
5.14.3	SceneDelegate	79
5.14.4	TabBar.....	80
5.14.5	Login.....	81
5.14.6	Home Screen.....	84
5.14.6.1	Abwesenheiten	88
5.14.6.2	Neue Abwesenheit.....	90
5.14.7	Überblick.....	92
5.14.8	NFC Reader.....	94
5.14.9	Tracking Verlauf.....	95
5.14.10	Einstellungen	97
5.14.11	Tracking Methoden	98
5.14.11.1	Bluetooth Beacons	99
5.14.11.2	Geofencing.....	100
5.14.12	Extensions	102
6	Entwicklung der Android App	105
6.1	Vorbereitung.....	105
6.1.1	Installation Android Studio.....	105
6.2	Theoretische Grundlagen	105
6.2.1	Android	105
6.2.2	Activities.....	105



6.2.2.1	Lebenszyklus	105
6.2.3	Services	106
6.2.4	Broadcast Receivers	106
6.2.5	Content Providers	107
6.2.6	UI-Entwicklung	107
6.2.6.1	Layout.....	107
6.2.6.2	Layoutarten.....	107
6.2.6.3	Linear Layout.....	107
6.2.6.4	Constraint Layout	108
6.2.6.5	Material Design Guidelines	109
6.3	Berechtigungen	110
6.4	Programmiersprache	110
6.5	Kotlin	110
6.5.1	Variablen.....	110
6.5.2	Konstanten.....	111
6.6	Android Studio	111
6.6.1	Erstellen eines Projekts.....	111
6.6.2	Navigation in Android Studio	115
6.6.3	Toolbar.....	115
6.6.4	SDK Manager	116
6.6.5	AVD Manager	117
6.6.6	Erstellen eines UI's	118
6.7	App Files	118
6.7.1	Android Manifest	118
6.7.2	Kotlin Klassen Files	118
6.7.3	Resources.....	118
6.7.3.1	anim	119
6.7.3.2	color	119
6.7.3.3	drawable.....	119
6.7.3.4	layout.....	119
6.7.3.5	menu	119
6.7.3.6	mipmap.....	119
6.7.3.7	navigation	119
6.7.3.8	values	122
6.7.3.9	xml.....	123
6.7.4	Assets	123
6.7.5	Gradle	123
6.7.5.1	build.Gradle	123
6.7.5.2	Projekt Gradle File	123
6.7.5.3	Module Gradle File	124
6.7.5.4	Build Prozess.....	126
6.7.5.5	Kompilieren.....	126
6.7.5.6	Packagen	126
6.8	App auf Android deployen	127
6.8.1	Emulator	127
6.8.2	USB-Debugging	128
6.9	API-Calls	129

6.9.1	Grundlagen	129
6.9.2	API-Calls mit Retrofit2 und OkHttp	130
6.9.2.1	Synchrone Calls	130
6.9.2.2	Asynchrone Calls.....	131
6.9.3	HTTPS und SSL Zertifikate	132
6.10	Tracking Methoden.....	134
6.10.1	NFC	134
6.10.1.1	Foreground Dispatch	134
6.11	User Interface.....	137
6.11.1	Login.....	137
6.11.2	Home Screen.....	137
6.11.3	Abwesenheit erstellen Dialog	138
6.11.4	NFC Reader.....	138
6.11.5	Einstellungen	139
6.12	App Funktionen	139
6.12.1	Login Activity.....	139
6.12.2	Main Activity.....	144
6.12.3	Timetracking	145
6.12.4	Abwesenheiten	154
7	Entwicklung der Web App	157
7.1	Vorbereitung.....	157
7.1.1	Node.js.....	157
7.1.2	Installation von Vue CLI	157
7.1.3	Installation und Einrichtung von Visual Studio Code.....	158
7.1.3.1	Erweiterungen	159
7.1.3.2	Vetur.....	161
7.1.3.3	Vue Snippets	161
7.1.3.4	Prettier.....	162
7.1.3.5	Highlight Line	162
7.1.3.6	Bracket Pair Colorizer 2.....	163
7.1.3.7	Better Comments.....	163
7.1.3.8	IntelliSense for CSS.....	164
7.1.3.9	JavaScript (ES6) Snippets	164
7.1.3.10	Vscode-icons	165
7.1.3.11	Flat UI.....	165
7.2	Theoretische Kenntnisse	166
7.2.1	Lifecycle einer Vue.js Web App.....	166
7.3	JavaScript.....	167
7.3.1	Wichtige Konstrukte	167
7.3.1.1	Async & Await.....	167
7.3.1.2	Callbacks	168
7.3.1.3	Ausführen zeitbedingter Funktionen	168
7.3.1.4	This-Zeiger	168
7.4	Vue.js	169
7.4.1	Instanz	169
7.4.2	Komponenten.....	169
7.4.3	Double Curly Syntax	169



7.4.4 Iterationen.....	170
7.4.5 Single File Komponenten	170
7.4.6 Navigation Guards	171
7.4.7 Vuetify.....	171
7.5 Vuex.....	172
7.5.1 Store	173
7.6 Internationalisierung	173
7.6.1 Vue I18n	173
7.7 Entwicklungsumgebung.....	174
7.7.1 Projekterstellung	174
7.7.2 Projektinitialisierung über Terminal in VS Code.....	177
7.7.3 Testen der Web App	178
7.7.3.1 CORS	179
7.8 Navigation und Layout der Web App	181
7.8.1 Navigationsbar	181
7.8.1.1 Sidebar – Admin	182
7.8.1.2 Sidebar – User.....	182
7.8.2 Login.....	182
7.8.2.1 Laden	183
7.8.2.2 Fehler	183
7.8.3 Übersicht.....	184
7.8.3.1 Übersicht - Admin	184
7.8.3.2 Übersicht – User.....	185
7.8.3.3 Laden	185
7.8.3.4 Fehler	186
7.8.4 Zeit.....	186
7.8.4.1 Laden	187
7.8.4.2 Fehler	187
7.8.5 Mitarbeiter	188
7.8.5.1 Bearbeiten	188
7.8.5.2 Erfolg – Mitarbeiter bearbeiten.....	189
7.8.5.3 Laden	190
7.8.5.4 Fehler	190
7.8.6 Einstellungen	191
7.8.6.1 Einstellungen – Admin	191
7.8.6.2 Einstellungen – User.....	191
7.8.6.3 Sprache	192
7.8.6.4 Report.....	192
7.8.6.5 Fehler	193
7.9 Funktionsbeschreibung der App-Komponenten	194
7.9.1 index.html	195
7.9.2 Mainskript – main.js	196
7.9.3 Style-Konfiguration – vuetify.js	196
7.9.4 Router – router / index.js	197
7.9.5 Maintemplate – App.vue.....	200
7.9.6 Vuex Store – store.js.....	200
7.9.7 Authentifizierungs-Skript - authentication/actions.js.....	201

7.9.8	Kalender-Skript – AppCalendar.vue	205
7.10	Views	208
7.10.1	Login.....	208
7.10.2	Dashboard	208
7.10.3	Kalender	209
7.10.4	Mitarbeiter (Ausschließlich für Admins).....	213
7.10.5	Settings.....	214
7.10.6	Logout.....	214
7.11	Deployen der Vue-App	215
8	Entwicklung des Backends	216
8.1	MySQL	216
8.2	Entwicklungsumgebung und Programme	216
8.2.1	Visual Studio Community 2019	216
8.2.2	MySQL Workbench.....	217
8.2.3	XAMPP	219
8.2.4	Postman.....	219
8.2.5	FileZilla	220
8.2.6	PuTTY.....	221
8.3	Programmiersprachen und Plattformen	221
8.3.1	Node.js.....	221
8.3.2	JSON	222
8.4	Server.....	223
8.4.1	Apache Web-Server.....	223
8.4.2	MySQL Server	224
8.4.3	Server mit Node.js.....	225
8.5	API	226
8.5.1	RESTful API.....	226
8.5.2	API-Keys.....	226
8.5.2.1	Authentifizierung jedes Calls.....	226
8.5.2.2	Auslesen der Timestamps	227
8.5.2.3	Auslesen der Mitarbeiterinformation	230
8.5.2.4	Auslesen der Abwesenheiten	231
8.5.2.5	Auslesen von Sollzeiten.....	233
8.5.2.6	Bearbeiten von Sollzeiten	236
8.5.2.7	Bearbeiten von Timestamps	238
8.5.2.8	Bearbeiten von Userdaten	239
8.5.2.9	Berechnung diverser Zeiten.....	240
8.5.2.10	Erzeugen des Reports	242
8.5.2.11	Löschen von Einträgen	242
8.5.2.12	Schreiben von Timestamps	243
8.5.2.13	Registrierung neuer User.....	246
8.5.2.14	Lesen der Tracking-Methoden	247
9	Zeiterfassung	249
9.1	Wudernitz	249
9.2	Ensbacher	250
9.3	Hacker.....	251
9.4	Zeitlberger	252



Timetracking

10	Literaturverzeichnis.....	253
11	Abbildungsverzeichnis	258
12	Programm-Listing	262

1 Lastenheft

1.1 Zielbestimmung

Das Ziel dieser Diplomarbeit ist es, den Mitarbeitern von Tailored Apps 2 Mobile-Apps und eine Web-App zur Verfügung zu stellen, welche Arbeitszeiten der Mitarbeiter aufzeichnen ihnen die Eintragung von Abwesenheiten ermöglichen. Mittels NFC-Tags, Bluetooth-Beacons und Geofencing sollen die Arbeitszeit bestimmt werden. Die NFC-Tags und Bluetooth-Beacons können nach Belieben im Büro verteilt werden. Der Geofencing-Bereich soll von Tailored Apps frei gewählt werden können. Die Mobile-Apps sollen für Android und iOS verfügbar sein.

Folgende Informationen sollen ebenfalls angezeigt werden:

- Arbeitszeiten, Arbeitsstunden, Pausen und Abwesenheiten des letzten Monats
- Aktuelle Abwesenheiten
- Gearbeitete Wochenstunden vs. Sollwochenarbeitsstunden

Die Benutzerdaten sollen von Redmine mit einem speziellem Testaccount geholt werden.

Persönliche Daten des Benutzers sind beispielsweise:

- Vor- und Nachname
- E-Mail-Adresse
- Telefonnummer

Die iOS-App soll in Swift mithilfe des Frameworks UIKit entwickelt werden. Die Android-App soll in Kotlin mithilfe des Frameworks MaterialIO programmiert werden.

1.2 Projekteinsatz

Diese Apps sollen die Mitarbeiter von Tailored Apps beim Eintragen der Arbeitszeiten unterstützen. Dabei soll das vorhandene Google Sheet abgelöst werden und vollständig durch die Mobile-Apps und Web-App ersetzt werden. Ausschließlich die Angestellten von Tailored Apps haben Zugriff auf die Apps. Durch Verwenden von zusätzlichen NFC-Tags, Bluetooth-Beacons und Geofencing ist es möglich auch außerhalb des Bürokomplexes die Arbeitszeiten zu tracken.

1.3 Produktfunktionen

1.3.1 Allgemeine Funktionen

- Die Benutzer sollen sich mit Redmine einloggen und authentifizieren können.
- Die Benutzer sollen ihre eigenen Einträge einsehen können und gegebenenfalls auch bearbeiten können.
- Abwesenheiten sollen eingetragen werden können.

1.3.2 Funktionen (Apps)

- Die Benutzer sollen auch beim Schließen der App angemeldet bleiben.
- Die Arbeitszeiten der Benutzer sollen mittels NFC-Tags, Bluetooth-Beacons und Geofencing erfasst werden.



Timetracking

- Die Tracking-Methoden können gewählt werden und gegeben falls auch deaktiviert werden.

1.3.3 Funktionen (Web)

- Die Benutzer können ihre eigenen Arbeitszeiten einsehen und editieren.
- Die Benutzer können Abwesenheiten aller Mitarbeiter sehen.
- Die Benutzer können ihre Abwesenheiten editieren.

1.3.4 Admin-Funktionen (Web)

- Der Admin soll Einträge von allen Mitarbeitern einsehen können.
- Der Admin soll Einträge hinzufügen, editieren und löschen können.
- Der Admin soll einen Report der rechtlich relevanten Arbeitszeiten erstellen können.
- Die Benutzer-Daten sollen vom Admin verwaltet werden können.

1.4 Produktdaten

Folgende Daten sollen in der eigenen Datenbank gespeichert werden:

- Benutzer-Daten
 - Vorname
 - Nachname
 - API-Key (Redmine)
 - E-Mail
 - Solararbeitszeiten
- Timestamp-Daten
 - Datum und Uhrzeit
 - Typ (Arbeitszeit, Pause, Abwesenheit)
- Tracking-Methoden
 - Geofencing
 - Bluetooth-Beacons

1.5 Qualitätsanforderungen

Auf die Stabilität, Zuverlässigkeit, Benutzerfreundlichkeit und Performance der Apps wird Wert gelegt. Die Apps sollen den plattformspezifischen Design-Guidelines entsprechend gestaltet sein. Das allgemeine Layout der Oberflächen soll plattformübergreifend wiedererkennbar sein.

1.6 Realisierung

Die Daten sollen aus der Datenbank mittels API-Calls abgefragt werden. Diese werden im JSON-Format an die Apps übergeben oder von den Apps ans Backend übermittelt.

2 Pflichtenheft

2.1 Zielbestimmung

Das Ziel ist es, den Mitarbeitern von Tailored Apps die Aufzeichnung der Arbeitszeiten und Abwesenheiten zu erleichtern. Neben einem Tracking Button in den Mobile-Apps, welcher das manuelle Timetracking ermöglichen, stehen dafür folgende Technologien zur Verfügung:

- NFC (NFC-Tags)
- Bluetooth-LE (Bluetooth-Beacons)
- Geofencing

Die Apps können nur von den Angestellten von Tailored Apps verwendet und eingesetzt werden.

2.1.1 Musskriterien

- Die Apps sollen nativ für Android, iOS und dem Web verfügbar sein.
- Die Mobile-Apps sollen ab der Android-Version 7 (Nougat) und der iOS-Version iOS13 funktionieren.
- Die Apps sind vollfunktionsfähig, wenn das Gerät NFC, Bluetooth und GPS unterstützt sowie eine Internetverbindung besitzt.
- Die Mitarbeiterdaten kommen aus der Redmine Datenbank, welche von Tailored Apps verwaltet wird.
- Die Kommunikation zwischen den Apps und dem Backend erfolgt über eine REST-API.
- Die Web-App soll über eine Benutzer-Ansicht und eine Admin-Ansicht verfügen.
- Abwesenheiten sind in den Apps einzutragen und in der Web-App zu editieren.
- Aus Sicherheitsgründen wird das Passwort und der Username der Mitarbeiter nicht im Backend gespeichert, sondern lediglich aus der Redmine Datenbank bei Bedarf geholt.
- Die erfassten Tracking-Daten können in den Apps angesehen werden. In den Mobile-Apps ist lediglich eine Darstellung der letzten 30 Tage möglich.
- Der Admin kann einen Report der Mitarbeiter erstellen, in welchem rechtlich relevante Arbeitszeiten aufgelistet werden sollen.
- In den Mobile-Apps können die Tracking-Methoden sowie die Benachrichtigungen aktiviert oder deaktiviert werden.
- Bei Zugriffen auf die eigene Datenbank wird POST verwendet, da immer Authentifizierungsdaten mitgesendet werden.

2.1.2 Wunschkriterien

- Die Apps weisen gute Stabilität und Performance auf.
- Die Apps sind benutzerfreundlich und intuitiv aufgebaut.

2.2 Produkteinsatz

2.2.1 Anwendungsbereiche

Die Apps werden ausschließlich für den firmeninternen Gebrauch verwendet. Den Mitarbeitern soll das Tracken von Arbeitszeiten und Abwesenheit im Arbeitsalltag erleichtert werden.

2.2.2 Zielgruppen

Die Zielgruppe sind die Mitarbeiter von Tailored Apps, welche ihre Arbeitszeiten im Büro oder von zu Hause aus tracken möchten. Die korrekte Benutzung der App, sowie das Verständnis von rechtlichen Aspekten werden vorausgesetzt.

2.3 Produktfunktionen

2.3.1 Allgemeine Funktionen

- Die Benutzer sollen sich mit Redmine einloggen können.
- Die Benutzer sollen ihre eigenen Einträge einsehen können.
- Abwesenheiten sollen eingetragen werden können.
- Folgende Abwesenheitsgründe gibt es:
 - Arztbesuch
 - Amtsweg
 - Dienstverhinderung (nur ganztags)
 - Krankenstand
 - Urlaub (nur ganztags)
 - Pflegefreistellung
 - Zeitausgleich

2.3.2 Funktionen (Apps)

- Die Benutzer sollen auch beim Schließen der App angemeldet bleiben.
- Die Arbeitszeiten der Benutzer sollen erfasst werden.
- Die Tracking-Methoden können gewählt werden.

2.3.3 Funktionen (Web)

- Die Benutzer sollen alle Abwesenheiten aller Mitarbeiter sehen.

2.3.4 Admin-Funktionen (Web)

- Der Admin soll Einträge von allen Mitarbeitern einsehen können.
- Der Admin soll Einträge hinzufügen, editieren und löschen können.
- Der Admin soll einen Report der rechtlich relevanten Arbeitszeiten erstellen können.

2.4 Produktdaten

Userdaten:

- User ID (eindeutig)
- Vorname
- Nachname
- E-Mail
- Erstellungsdatum des Accounts
- API-Key von Redmine
- Sollzeiten
 - Uhrzeiten für jeden Wochentag
 - Datum

Timestamps:

- Timestamp ID (eindeutig)
- Uhrzeit
- Datum
- Kommentar
- User ID
- Typen (Start, Stop, Pause, Abwesenheiten)

Bluetooth-Beacons:

- ID (eindeutig)
- UUID
- Minor
- Major
- Radius

Geofencing:

- ID (eindeutig)
- Breitgrad
- Längengrad
- Radius



2.5 Qualitätsbestimmungen

	sehr wichtig	wichtig	weniger wichtig	unwichtig
Robustheit	x			
Zuverlässigkeit		x		
Korrektheit		x		
Benutzerfreundlichkeit	x			
Effizienz			x	
Portierbarkeit				x
Kompatibilität		x		

3 Verwendete Programme

3.1 Mockups

3.1.1 Balsamiq



Abbildung 1: Balsamiq [301]

3.1.2 XD



Abbildung 2: Adobe XD [302]

3.1.3 Sketch



Abbildung 3: Sketch [303]

3.1.4 Overflow



Abbildung 4: Overflow [304]

3.2 IDE

3.2.1 Visual Studio Code



Abbildung 5: Visual Studio Code [305]

3.2.2 Visual Studio Community



Abbildung 6: Visual Studio Community [306]

3.2.3 Xcode



Abbildung 7: Xcode [307]

3.2.4 Android Studio



Abbildung 8: Android Studio [308]

3.3 Version Control

3.3.1 Git



Abbildung 9: Git [309]

3.3.2 Github Desktop



Abbildung 10: Github Desktop [310]

3.3.3 Tortoise Git



Abbildung 11: Tortoise Git [311]

3.4 Runtime

3.4.1 Node.js



Abbildung 12: node.js [312]

3.5 Storage

3.5.1 Google Drive



Abbildung 13: Google Drive [313]

3.5.2 Google Backup & Sync

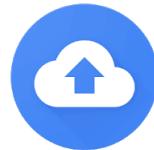


Abbildung 14: Google Backup & Sync [314]

3.6 Bildbearbeitung

3.6.1 Gimp



Abbildung 15: Gimp [315]

3.6.2 App Icon Resizer



Abbildung 16: App Icon Resizer [316]

3.7 Browser

3.7.1 Google Chrome



Abbildung 17: Google Chrome [317]

3.7.2 Firefox



Abbildung 18: Firefox [318]

3.8 Office

3.8.1 Excel



Abbildung 19: Microsoft Excel [319]

3.8.2 Word



Abbildung 20: Microsoft Word [319]

3.8.3 PowerPoint



Abbildung 21: Microsoft PowerPoint [319]

4 Projektplan

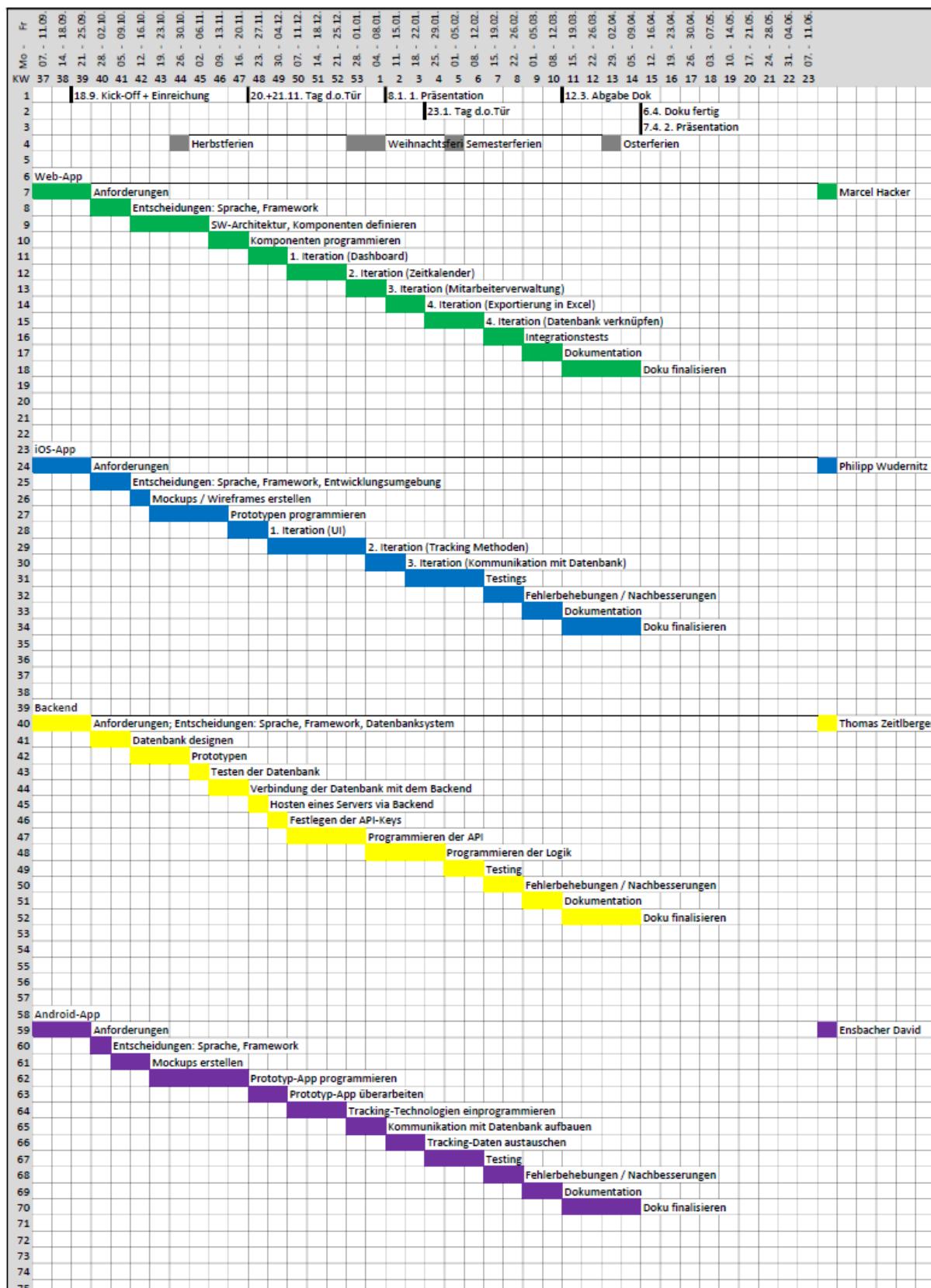


Abbildung 22: Projektplan

5 Entwicklung der iOS App

5.1 Vorbereitung

5.1.1 Apple ID

Ohne einer Apple ID kann keine Verbindung mit dem Apple App Store entstehen und somit können auch keine Apps für iOS, MacOS oder tvOS in Xcode entwickelt werden. Auf der offiziellen Apple Website kann eine Apple ID eingerichtet werden [1]. Mithilfe dieser Apple ID können Apps in Xcode entwickelt werden und im Simulator am Mac oder auf einem echten Gerät getestet werden.

5.1.2 Developer Account

Um Apps im App Store anbieten und einige spezielle Features von Apple (Capabilities) [2] benutzen zu können, wird ein Apple Developer Account [3] benötigt. Diese Mitgliedschaft kostet jährlich 99,99€ und wurde aus folgenden Gründen erworben:

- CoreNFC: CoreNFC ist das Framework, welches zum Scannen von NFC-Tags verwendet wird.
- Background Tasks: Ermöglicht das Erkennen von Bluetooth-Beacons (CoreBluetooth) und Geofencing-Bereichen (CoreLocation) wenn die App im Hintergrund läuft.

Diese und viele weitere Funktionen stehen nur Entwicklern mit bezahltem Apple Developer Account zur Verfügung.

5.1.3 Installation Xcode

Xcode [4] kann kostenlos aus dem Mac App Store heruntergeladen werden und auf allen Geräten mit einem MacOS Betriebssystem ab der Version 10.13.6 installiert werden. Das Entwickeln auf anderen Betriebssystemen, wie Linux oder Windows, ist nicht möglich.

5.2 Theoretische Grundlagen

5.2.1 Lebenszyklus einer iOS-App

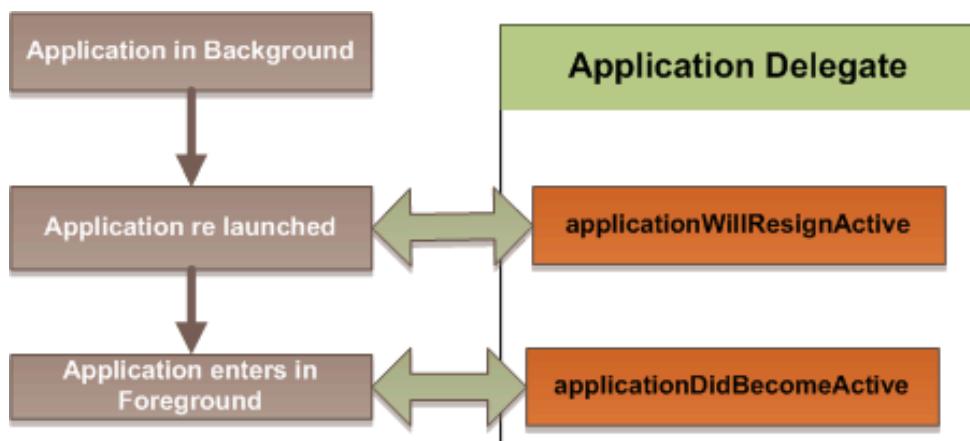


Abbildung 23: Lebenszyklus einer iOS-App [22]

Nach dem Einschalten des iPhones laufen zunächst nur Programme, welche Bestandteil des Betriebssystems (iOS) sind. Erst nachdem die App geöffnet und in den Arbeitsspeicher geladen wurde, werden die „shared libraries“ ausgeführt. „SpringBoard“ [44] kümmert sich um die Animation des Launch Screens. Kurz danach wird die App komplett ausgeführt und der „application delegate“ erhält Benachrichtigungen und Anweisungen [5], wie der Verlauf und die nächsten Schritte der App aussehen sollen.

Es gibt verschiedene Zustände einer iOS-App:

- Not Running
- Inactive
- Active
- Background
- Suspended

5.2.1.1 **AppDelegate**

Nachdem ein neues Xcode-Project (iOS-App) angelegt wurde, wird eine Klasse namens *AppDelegate* mit einigen leeren und hilfreichen Methoden angelegt. Diese Klasse ist für „lifecycle events“ vorgesehen, beispielsweise wenn die App gelauncht wird, die App in den Hintergrund oder Vordergrund rückt oder Daten von einem Web Server geholt werden. [7] Die *AppDelegate* Klasse wird für folgenden Funktionen benutzt: [36]

- Login oder Cloud Service
- Konfiguration von Push Notifications Handlers
- Das Verwalten von speziellen Events, wie in den Background gehen oder die iOS-App schließen

5.2.1.2 **SceneDelegate**

Nach dem ein neues Xcode-Project (iOS-App) angelegt wurde, wird ebenfalls eine Klasse namens *SceneDelegate* mit einigen leeren und hilfreichen Methoden angelegt. Diese Klasse wird vorrangig für die Darstellung der einzelnen Views der App verwendet. Zum Beispiel kann der *RootViewController* [39] ausgewählt werden. [7]

Viele Funktionen, welche von der *AppDelegate* zur Verfügung gestellt wurden, wurden mit iOS13 in die *SceneDelegate* verlegt. Das Konzept eines Windows wurde von einer Scene abgelöst. Die *SceneDelegate* erlaubt das Erstellen von Multi-Window-Apps bzw. Multi-Scene-Apps, da eine App mehrere Scenes besitzen kann. Die *SceneDelegate* Klasse kümmert sich um Lifecycle-Events (active, resign, disconnect), welche früher in der *AppDelegate* Klasse gehandhabt wurden. [38]



5.2.1.3 ViewController

ViewController werden für die konkrete Darstellung, sowie für die Erstellung des User-Interfaces, verwendet. *ViewController* beinhalten alle UI-Elemente eines Views und weisen Daten / Aktionen (z.B. Text eines Labels, Funktion eines Buttons) zu den einzelnen UI-Elementen zu (dies wird auch Model-View-Controller (MVC) genannt [9]). [10]

5.2.2 UI-Entwicklung

5.2.2.1 Storyboards vs. Programmatically

Xcode erlaubt Entwicklern die Auswahl zwischen 2 Arten von UI-Entwicklung:

- Storyboards: Erstellung des User-Interfaces mithilfe des sogenannten „Interface-Builder“
- Programmatically: Traditionelle Erstellung des User-Interfaces mit Code

Mit Storyboards können UI-Elemente grafisch in den View eingefügt werden, welche dort auch angepasst werden können (Größe, Farbe, „Constraints“, ...). Trotzdem kann nicht alles in Storyboards gemacht werden, da einige Funktionen der UI-Elemente in Code geschrieben werden müssen. Für komplexe / spezielle Applikationen muss ebenfalls auf Code zurückgegriffen werden. [11]

5.2.2.2 UI-Elemente

iOS-Apps bestehen aus verschiedenen UI-Elementen mit verschiedenen Funktionen.

Manche UI-Elemente reagieren auch auf Benutzerinteraktionen. Wichtige UI-Elemente sind beispielsweise:

- Buttons (*UIButton*)
- Text Fields (*UITextField*)
- Alerts / Action Sheets (*UIAlertController*)
- Table Views (*UITableView*)
- Tab Bars (*UITabBarController*)

Alle UI-Elemente können entweder in Storyboards oder in Code eingefügt, konfiguriert und angepasst werden. [12]

5.2.2.3 Constraints

Mithilfe von „Constraints“ können UI-Elemente an beliebigen Stellen (Koordinaten) und in beliebigen Abständen zueinander in der App fixiert werden. Constraints können im Interface-Builder und im Code konfiguriert werden. Dabei ist die Sinnhaftigkeit der gewählten Constraints der einzelnen UI-Elemente besonders zu beachten. Es ist empfehlenswert, die App auf verschiedenen iPhones im Simulator zu testen, da verschiedene Displaygrößen einen Effekt auf das User-Interface haben können.

Beispiel: Einen Button (Höhe: 50, Breite: 75) in die Mitte des Screens fixieren.

```
NSLayoutConstraint.activate([
    button.heightAnchor.constraint(equalToConstant: 50),
    button.widthAnchor.constraint(equalToConstant: 75),
    button.centerXAnchor.constraint(equalTo:
        view.centerXAnchor),
    button.centerYAnchor.constraint(equalTo:
        view.centerYAnchor),
])
```

Listing 1: Button Constraints

5.2.2.4 Human Interface Guidelines

Apple stellt allen Entwicklern sogenannte „Human Interface Guidelines“ [46] zur Verfügung, welche als Richtlinien zur Erstellung von User-Interfaces dienen sollen. Als Anfänger in iOS-App-Entwicklung sind diese durchzuarbeiten. Diese Guidelines geben beispielsweise an, wie:

- ein Fetch-Zugriff im User-Interface ausschauen soll.
- die Navigation durch die App ausschauen soll.
- Titel / Überschriften heißen sollen.

5.3 Layout

5.3.1 Auto Layout

„Auto Layout“ berechnet dynamisch die Größen und Positionen von Views, basierend auf deren gesetzten Constraints. Auto Layout ermöglicht Entwicklern User-interfaces zu erstellen, welche dynamisch auf interne und externe Änderungen reagieren können. [15]

Externe Änderungen:

- Der Benutzer verändert die Größe von Fenstern (MacOS)
- Das Gerät rotiert (iOS)
- Aktiv Views zeigen / verstecken

Interne Änderungen:

- Inhalte der App werden angezeigt
- Internationalisieren
- Dynamic Type (iOS)

5.4 Programmiersprachen

Traditionell können iOS-Apps in zwei Programmiersprachen geschrieben werden:

- Objective-C
- Swift



Da Objective-C die ältere der beiden Programmiersprachen ist, kann das neueste Framework, namens SwiftUI, nur in Swift verwendet werden. Im Gegensatz zu SwiftUI wird UIKit auch in Objective-C weiterhin unterstützt. UIKit wurde als Framework für die iOS-App gewählt, da es etabliert ist. SwiftUI ist ein sehr neues Framework, welches noch mit einigen Bugs kämpft.

5.4.1 Objective-C

Im Gegensatz zu C ist Objective-C (auch ObjC genannt) eine objektorientierte Programmiersprache. Der Compiler von Objective-C erlaubt es auch normale C-Programme zu kompilieren.

Eigenschaften von Objective-C [5]:

- Dynamisches Binden
- Dynamisches Typisieren
- Nachrichten
- Kategorien
- Protokolle
- RTTI / Reflexion
- Klassenobjekte

5.4.2 Swift

Swift ist ebenfalls eine objektorientierte Programmiersprache und wird ausschließlich zur Entwicklung von Apple Apps verwendet. Swift verwendet Konzepte von Objective-C, Ruby und anderen Programmiersprachen. [13] Vorteile von Swift im Vergleich zu Objective-C [14]:

- Schneller
- Sicherer
- Lesbarer
- Weniger Code
- Besseres Speicher Management
- Open Source

5.5 Swift

5.5.1 Variablen

Variablen in Swift werden mit `var` deklariert. Grundsätzlich können Variablen Inhalte aller Datentypen speichern, außer die Variable ist durch „Type Annotation“ [50] einem Datentyp zugeordnet.

Beispiel:

```
var myVar = "myString"  
var thisString: String = "myString"
```

Listing 2: Swift Variablen

5.5.2 Konstanten

Konstanten werden mit `let` deklariert und sind nach der Initialisierung nicht mehr änderbar. Durch „Type Annotation“ [51] kann Konstanten ebenfalls ein Datentyp zugeordnet werden.

Beispiel:

```
let myConstant = 123
let thisConstant: Int = 123
```

Listing 3: Swift Konstanten

5.5.3 If-Bedingung

Wie in fast allen Programmiersprachen gibt es auch in Swift die *If-Bedingung*, welche nach einer wahren Bedingung in einen Code-Block wechselt und diesen Code ausführt. Falls die Bedingung falsch ist, wird der Code im `else` Segment ausgeführt, oder, falls das `else` Segment nicht vorhanden ist, übersprungen.

Beispiel:

```
if myVar == thisString {
    print(thisString)
} else {
    print(myConstant)
}
```

Listing 4: Swift If-Bedingung

5.5.4 Schleifen

Folgende Schleifen gibt es in Swift:

- `for`
- `while`
- `repeat` (wie `do-Schleife` in C / C++)

Beispiel for-Schleife:

```
for i in 0...10 {
    print(i)
}
```

Listing 5: Swift for-Schleife

**Beispiel while-Schleife:**

```
var i = 0

while i < 10 {
    i += 1
    print(i)
}
```

*Listing 6: Swift while-Schleife***Beispiel repeat-Schleife:**

```
var i = 0

repeat {
    i += 1
    print(i)
} while i < 10
```

Listing 7: Swift repeat-Schleife

5.5.5 Strukturen und Klassen

Strukturen und Klassen können Daten / Informationen von Objekten speichern und diese mithilfe von Methoden (Funktionen) aufbereiten, ändern oder zur Verfügung stellen.

Beispiel Struktur:

```
struct myStruct {
    var name: String
    var age: Int

    init() {
        name = ""
        age = 0
    }
}

// Object
var newObjectOfStruct = myStruct()
```

*Listing 8: Swift Struktur***Beispiel Klasse:**

```
class MyClass {
    var name: String
    var age: Int

    init() {
        name = ""
        age = 0
    }
}
```

```

}

// Method
func setData(name: String, age: Int) {
    self.name = name
    self.age = age
}
}

// Object
var newObjectOfClass = myClass()

```

Listing 9: Swift Klasse

5.5.6 Funktionen

Funktionen sind Code-Segmente, welche beim Aufruf ausgeführt werden. Funktionen benötigen einen Namen, welcher die Funktion eindeutig identifizieren sollte. Beim Aufruf der Funktion muss der Name, sowie die Übergabeparameter der Funktion, angegeben werden.

Beispiel Funktion:

```

func myFunction(name: String, age: Int) -> Bool {
    if name == "Tom" && age >= 18 {
        return true
    } else {
        return false
    }
}

print(myFunction(name: "Tom", age: 18))

```

Listing 10: Swift Funktion

5.5.7 Optionals

Optionals erlauben Variablen auch den Wert *nil* zu speichern, wobei der Versuch, *nil* bei normalen Variablen zu speichern, einen Fehler verursachen würde.

Beispiel Optionals:

```

var data: String?
data = nil
data = "Name"

```

Listing 11: Swift Optionals



Will man auf den Inhalt einer *Optional-Variablen* zugreifen, muss diese „geunrappt“ werden. Dies ist nur möglich, wenn der Inhalt ungleich *nil* ist, ansonst kommt es zu einem Fehler. Unrappen bedeutet, dass der gespeicherte Wert (ein wirklicher Wert oder *nil*) abgefragt wird.

Beispiel:

```
var newData = data!
```

Listing 12: Swift Optionals unwrappen

Das Rufzeichen (!) signalisiert, das *data* geunrappt wird. Wäre diese Variable *nil*, würde das Programm einen Fehler ausgeben. Deshalb können *Optional-Variablen* auch sicher geunrappt werden.

Beispiel:

```
if let newData = data {  
    print(newData)  
} else {  
    print("data is nil")  
}
```

Listing 13: Swift Optionals sicher unwrappen

5.6 UI-Frameworks

5.6.1 UIKit

UIKit stellt den Entwicklern die benötigte Infrastruktur für iOS-Apps und tvOS-Apps zur Verfügung. UIKit stellt ebenfalls die Fenster- und View-Architektur bereit, sowie eine Event-Handling Architektur und die Hauptschleife, welche Interaktionen vom Benutzer, dem System und der App steuert. Weitere Unterstützungen sind [16]:

- Animationen
- Zeichnungen
- Dokumente
- Textverwaltung
- Display

5.6.2 SwiftUI

SwiftUI ist ein innovatives Framework, mit welchem auf jeder Apple-Plattform (iOS, MacOS, tvOS) User-Interfaces entwickelt werden kann. Vorteile von SwiftUI gegenüber UIKit: [17]

- Weniger Code
- Lesbarer

SwiftUI wurde 2019 veröffentlicht und ist somit ein neues Framework, welches noch einige Jahre benötigen wird, um Industriestandard in der Apple-App-Entwicklung zu werden.

5.7 Entwicklungsumgebung

Zur Entwicklung von IOS-Apps, aber auch MacOS-Apps und tvOS-Apps, wird Xcode verwendet. Xcode ist eine IDE, welche Objective-C, Swift, C und C++ unterstützt, wobei Apps nur in Swift und Objective-C entwickelt werden können. Xcode bietet auch einen „Playground“ an, in welchem Programmcode ausprobiert werden kann.

5.7.1 Erstellen eines Projekts

Nachdem Xcode geöffnet wurde, wird folgendes Bild gezeigt.



Abbildung 24: Xcode Startscreen

Auf der rechten Seite des Bildes können zuletzt verwendete Projekte geöffnet werden. Soll ein neues Projekt erstellt werden, muss auf **Create a new Xcode project** geklickt werden. Danach ist auszuwählen, welche Art von Projekt erstellt werden soll. In diesem Fall soll eine iOS-App erstellt werden (**iOS->App**). Nach der Auswahl der Art des Projekts mit **Next** bestätigen.

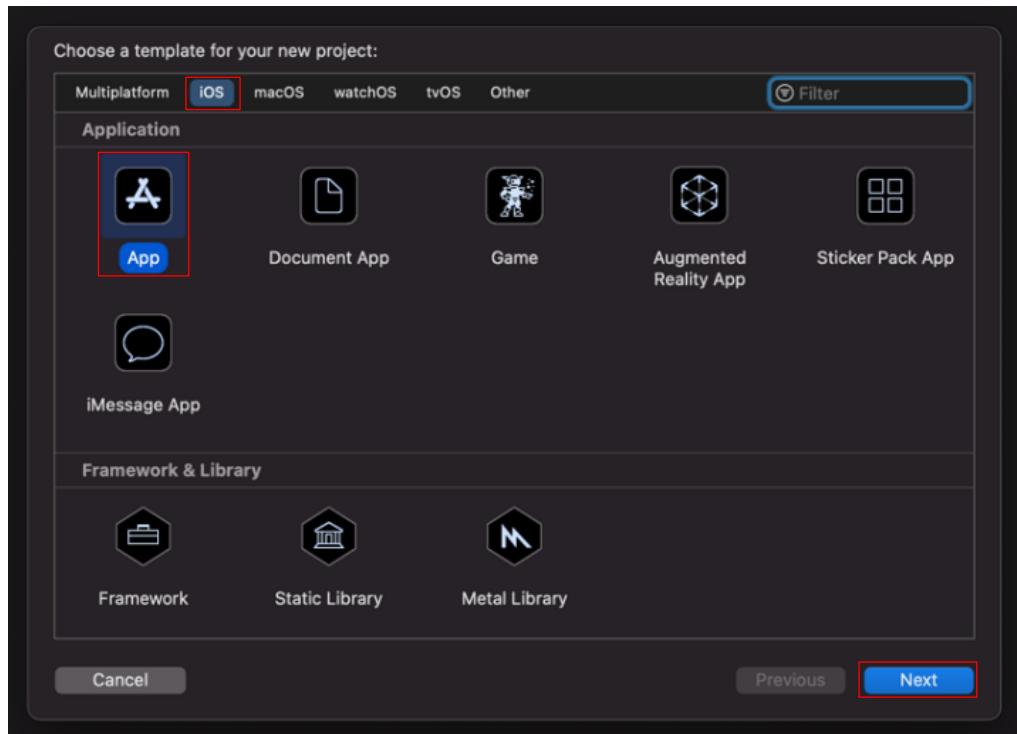


Abbildung 25: Xcode Auswahl iOS-App

Darauffolgend sind wichtige Parameter, wie beispielsweise Projektnamen, anzugeben. Hier ebenfalls mit **Next** bestätigen.

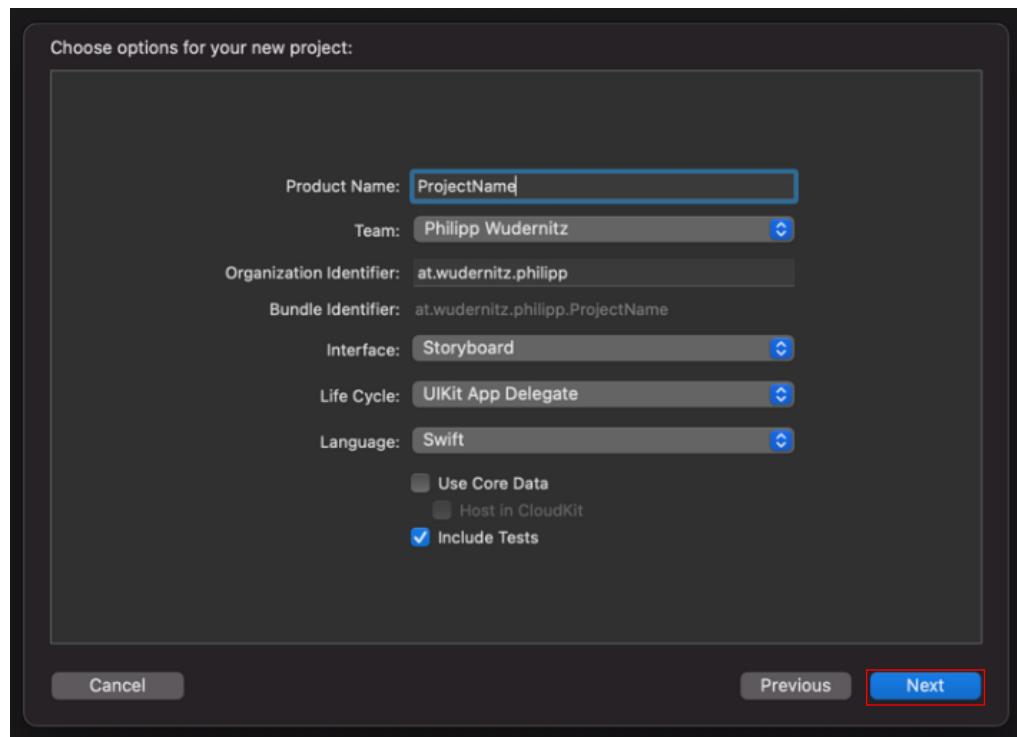


Abbildung 26: Xcode Projektoptionen

Bemerkung: Falls CoreData ebenfalls verwendet werden soll, muss der Haken bei **Use CoreData** gesetzt werden.

Nachdem die zuvor erwähnten Parameter eingetragen wurden, muss das Projekt abgespeichert werden. Es ist ebenfalls möglich, ein „Git Repository“ mitzuerstellen.

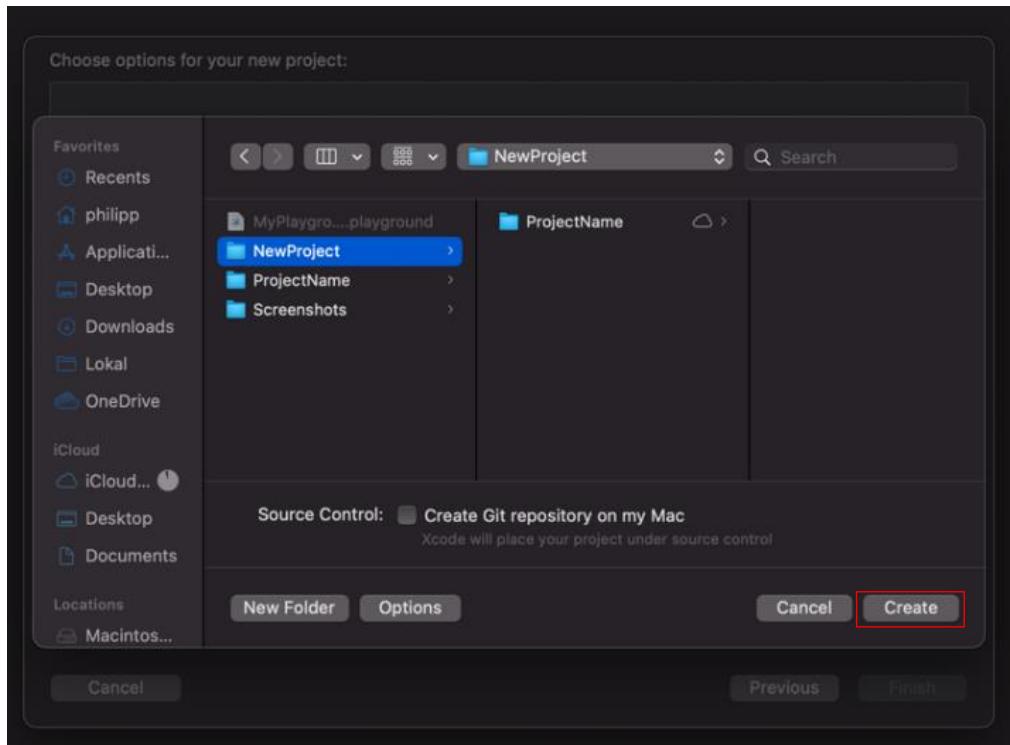


Abbildung 27: Xcode Projekt speichern

Nachdem auf **Create** geklickt wurde, wird das Projekt angelegt und die Projekterstellung ist somit abgeschlossen.

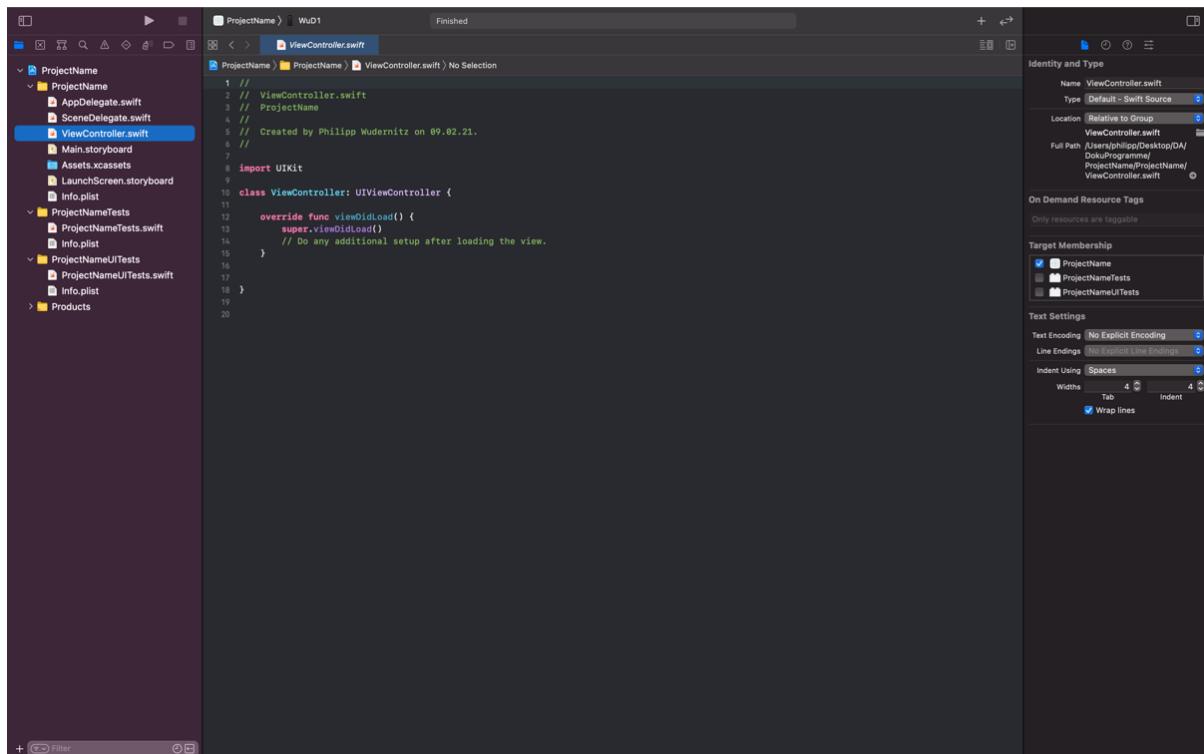


Abbildung 28: Xcode neues Projekt fertig



5.7.2 Entwicklungsumgebung Xcode

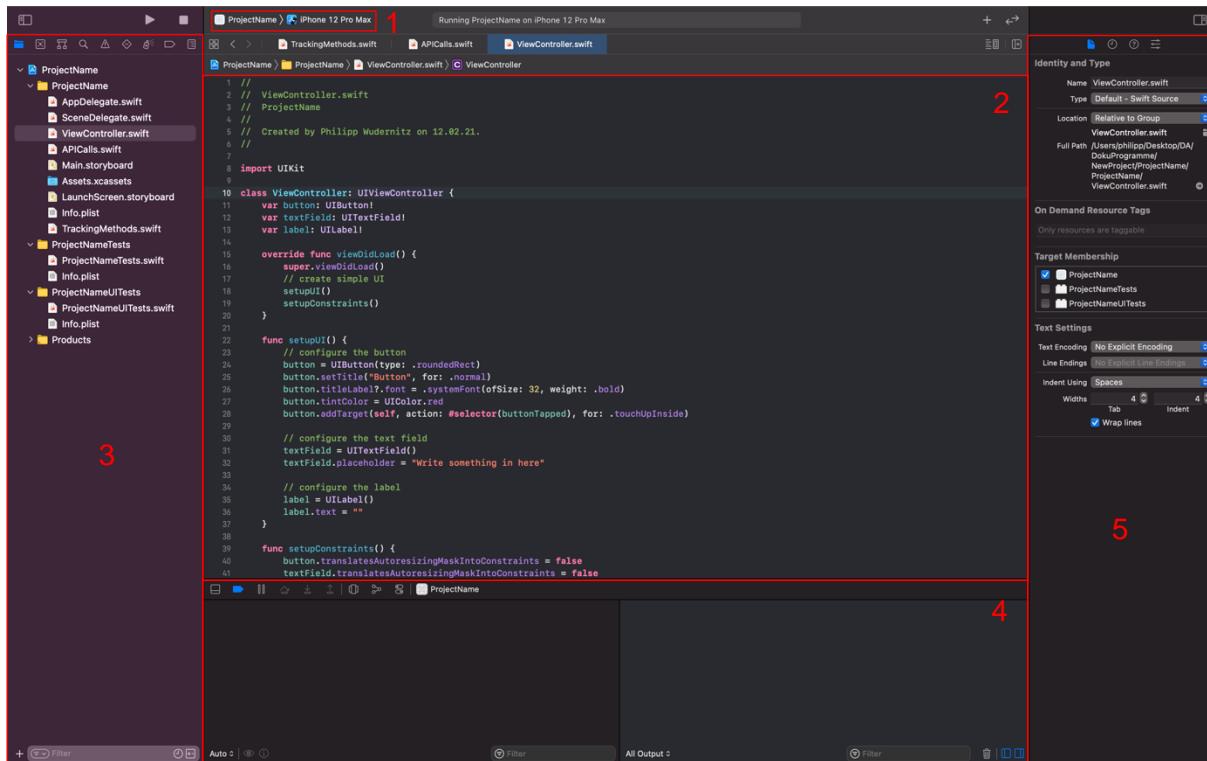


Abbildung 29: Xcode Hauptansicht

- 1) Auswahl, wo die App gestartet werden soll (Simulator oder echtes iPhone)
- 2) Editor, in welchem UI-Elemente, API-Calls oder ähnliches programmiert werden können.
- 3) In der Navigator Area werden alle eingebundenen Files und Libraries des Projekts angezeigt.
- 4) In der Debug Area können Konsolenausgaben beim Kompilieren und Simulieren angesehen werden, welche auf Errors oder Warnings hinweisen können.
- 5) In der Utility Area können Properties / Eigenschaften der Elemente / Klassen geändert werden.

Es ist ebenfalls möglich folgende Parameter der App zu spezifizieren:

- Version und Name der App
- Bundle Identifier des Entwicklers
- iOS-Version
- Orientation (Quer- oder Hochformat)

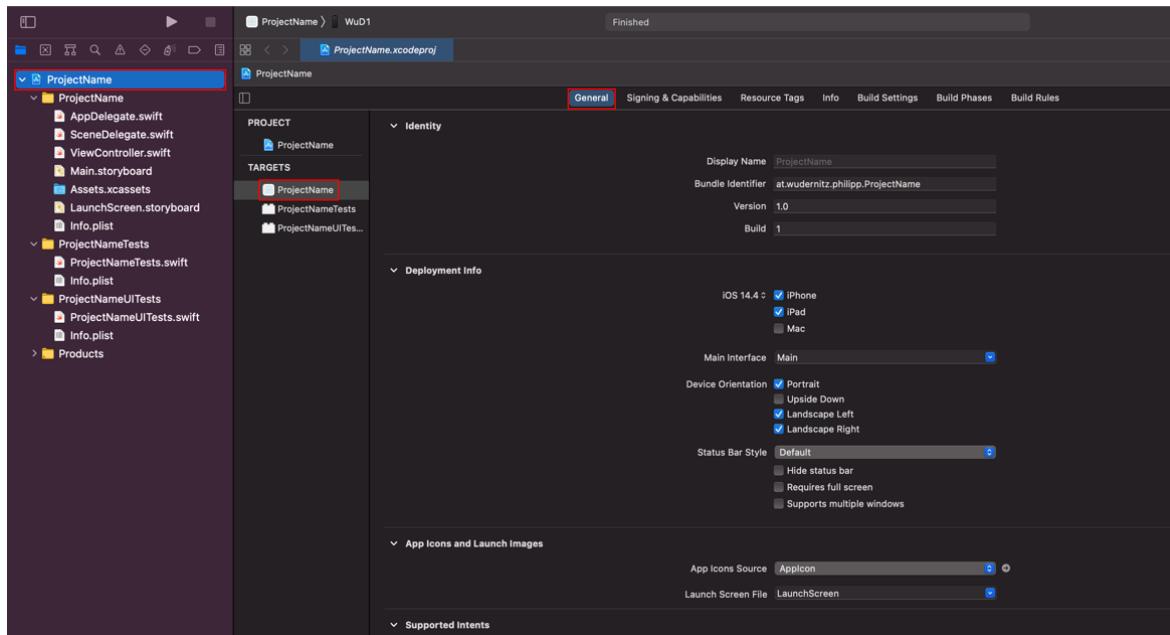


Abbildung 30: Xcode Projekteigenschaften

Mithilfe von *Capabilites* [2] ist es möglich, zusätzliche Funktionen von Apple zu nutzen. Voraussetzung ist der Besitz eines bezahlten Apple Developer Accounts.

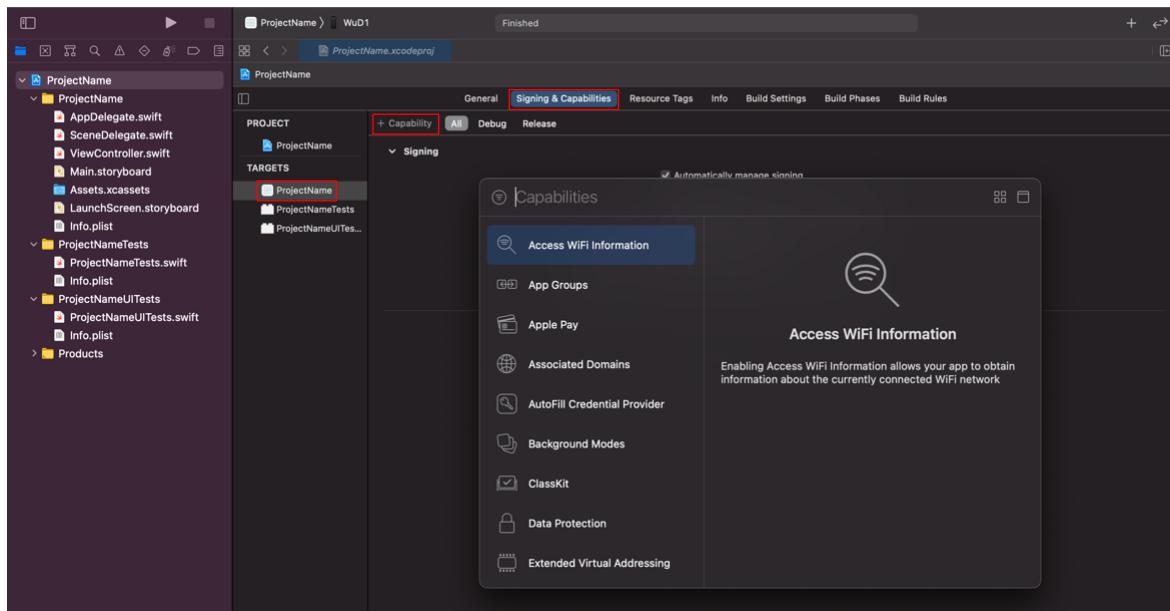


Abbildung 31: Xcode Capabilities



Weitere Information zur App sind unter *Info* zu finden, beispielsweise:

- Deployment Target (iOS-Version beim Testen im Simulator / am echten iPhone)
- Lokalisierungsdateien

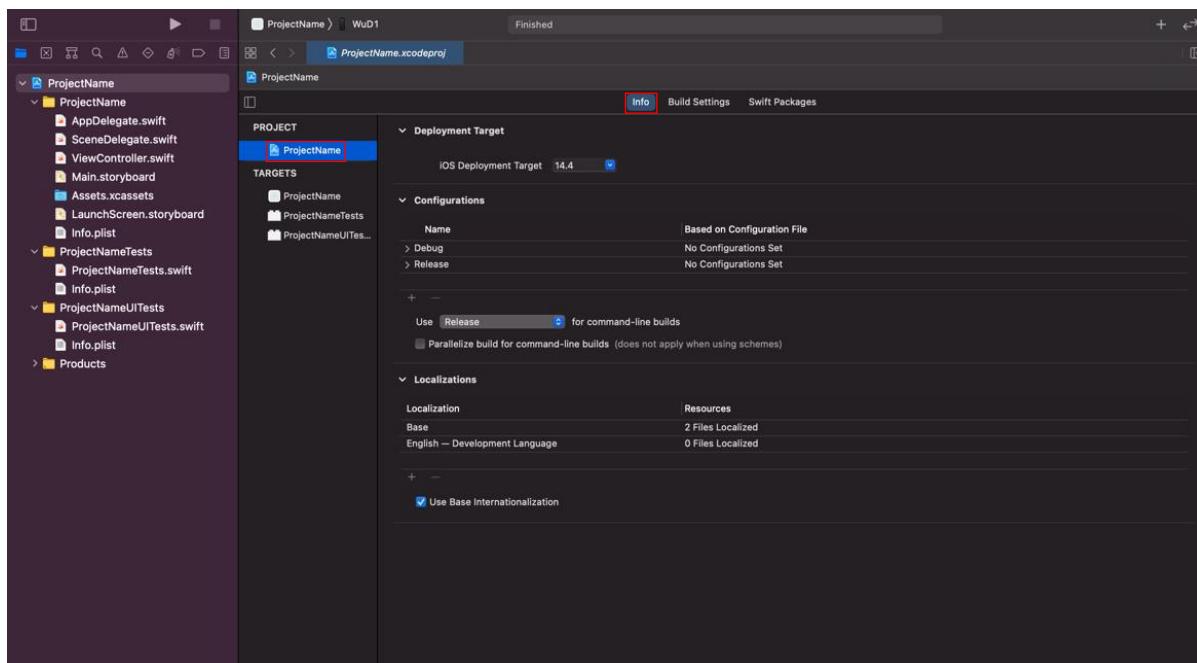


Abbildung 32: Xcode Project-Info

In der *Info.plist-Datei* können Properties / Eigenschaften gesetzt werden, welche gewisse Funktionen (z.B. NFC, Bluetooth, Location Tracking) freischalten. Ist eine Property / Eigenschaft hinzugefügt worden, wird der Benutzer beim Verwenden der App gefragt, ob die gewollte Funktion (z.B. Bluetooth verwenden) zugelassen werden soll.

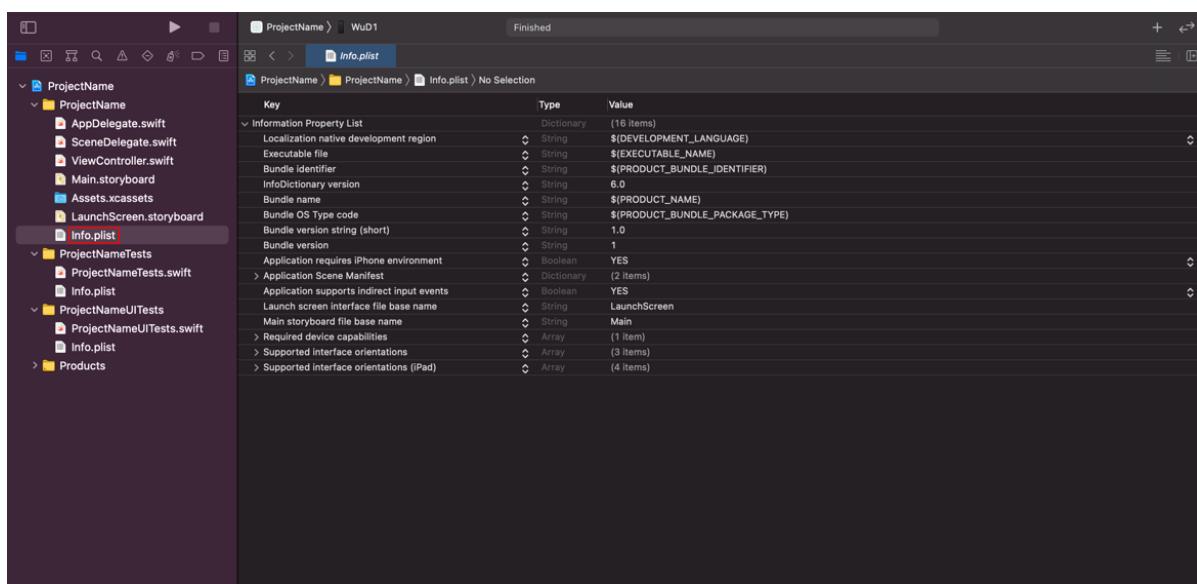


Abbildung 33: Xcode Info.plist

5.7.3 Erstellen eines UI's (Programmatically)

User-Interfaces werden in einer abgeleiteten Klasse der Oberklasse `UIViewController` erstellt. Beim Erstellen eines neuen Projekts wird das File `ViewController.swift` miterstellt, in dem schon eine Sub-Klasse der Oberklasse `UIViewController` namens `ViewController` vorhanden ist. In der Methode `viewDidLoad()` werden alle UI-Elemente initialisiert und in die View-Hierarchie eingefügt.

Beispiel:

```
class ViewController: UIViewController {
    var button: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        button = UIButton(type: .roundedRect)
        button.translatesAutoresizingMaskIntoConstraints = false
        button.setTitle("Press Me", for: .normal)
        button.backgroundColor = UIColor.blue
        button.tintColor = UIColor.white

        view.addSubview(button)

        NSLayoutConstraint.activate([
            button.heightAnchor.constraint(equalToConstant: 50),
            button.widthAnchor.constraint(equalToConstant: 75),
            button.centerXAnchor.constraint(equalTo:
                view.centerXAnchor),
            button.centerYAnchor.constraint(equalTo:
                view.centerYAnchor),
        ])
    }
}
```

Listing 14: Einfaches UI



Sobald dieser Code ausgeführt wird, wird ein iPhone im Simulator gestartet und die iOS-App kann getestet werden.

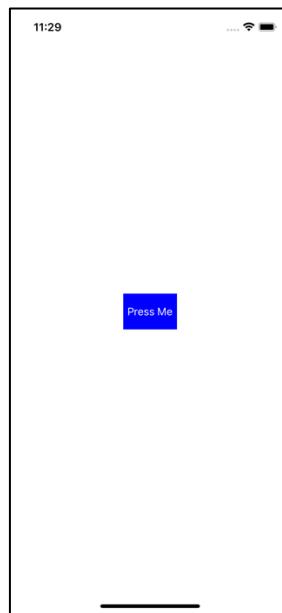


Abbildung 34: iOS-App mit einem Button in der Mitte des Views

5.7.4 App auf iPhone deployen

Es ist ebenfalls möglich, die iOS-App auf einem echten Gerät zu testen. Zunächst muss ausgewählt werden, auf welchem Gerät die App getestet werden soll.



Abbildung 35: App deployen

Hier können ebenfalls Simulatoren verschiedener iPhone-Modelle ausgewählt werden.



Abbildung 36: Auswahl Gerät zum Deployen

Das Gerät „WuD1“ entspricht dem echten iPhone des Entwicklers. Nach der Auswahl des Geräts kann **CMD+R** gedrückt werden und die App wird auf dem Gerät ausgeführt.

5.8 API-Calls

5.8.1 Grundlagen

APIs dienen als Schnittstellen zwischen Web-Server (und deren Datenbanken, welche relevante Informationen für Endgeräte speichern) und dem Endgerät. In Swift wird die Klasse *URLSession* verwendet, um mit API-Calls zu arbeiten. Mit dem Protokoll *Codable* können Strukturen in JSON geparsed (aus JSON ein Objekt erzeugen) und geserialized (aus einem Objekt JSON erzeugen) werden. [19]

Für API-Calls werden Model-Classes / Model-Structures benötigt, welche die Struktur der zu übertragenen / erhaltenen Informationen besitzen.

Beispiel einer Model-Structure mit folgenden Parametern:

- ID (Int)
- Name (String)
- Alter (Int)

In Swift:

```
struct Model: Codable {  
    var id: Int  
    var name: String  
    var age: Int  
}
```

Listing 15: Model Struktur



5.8.2 Daten lesen

GET:

```
func fetchData(completion: @escaping (Model?) -> Void) {
    let url = "https://myserver.com/api/fetchDataFromWebserver"
    let urlString = URL(string: url)!

    // making the request
    let task = URLSession.shared.dataTask(with: urlString) { data,
        response, error in
        if let error = error {
            print(error)
            return
        }

        if let response = response {
            if let httpStatus = response as? HTTPURLResponse {
                print(httpStatus.statusCode)
                return
            }
        }
    }

    if let data = data {
        let object = try? JSONDecoder().decode(Model.self, from:
            data)
        completion(object)
        return
    }
}
task.resume()
}
```

Listing 16: API-Call GET

5.8.3 Daten schreiben

POST:

```
func writeData(completion: @escaping (Int?) -> Void) {  
    let url = "https://myserver.com/api/writeDataToWebserver"  
    let urlString = URL(string: url)!  
  
    let model = Model(id: 0, name: "MyName", age: 18)  
    let jsonData = try? JSONEncoder().encode(model)  
  
    var request = URLRequest(url: urlString)  
    request.httpMethod = "POST"  
    request.httpBody = jsonData  
  
    // making the request  
    let task = URLSession.shared.dataTask(with: request) { data,  
        response, error in  
        if let error = error {  
            print(error)  
            return  
        }  
  
        if let response = response {  
            if let httpStatus = response as? HTTPURLResponse {  
                print(httpStatus.statusCode)  
                completion(httpStatus.statusCode)  
                return  
            }  
        }  
        task.resume()  
    }  
}
```

Listing 17: API-Call POST

5.9 Tracking Methoden

5.9.1 Grundlagen

Um die Position des Benutzers herauszufinden, werden in dieser Diplomarbeit 3 Tracking Methoden verwendet, welche der Benutzer in der iOS-App frei wählen kann:

- NFC
- Bluetooth Beacons
- Geofencing

Das Tracken von Bluetooth Beacons und die Feststellung der Position mittels Geofencing kann auch im Hintergrund erfolgen. Bei NFC muss ein sogenannter NFC-Tag vom Benutzer der iOS-App gescannt werden.

5.9.2 NFC

Um zu erkennen, welcher NFC-Tag mit welchem Inhalt gescannt wird, besitzen die Tags sogenannte *Payloads*. In diesen befindet sich ein Text, welcher von der iOS-App ausgelesen wird. Besonders auf das verwendete Format des Inhalts (z.B. utf-8) muss beim Lesen in der App und beim Schreiben des Tags geachtet werden. Dieser Text muss zuvor auf den NFC-Tag geschrieben (z.B. mit der iOS-App *NFC Tools* [20]).

Setup von NFC in Xcode:

Um NFC überhaupt in der App verwenden zu können, muss die Capability *Near Field Communication Tag Reading* zum Xcode-Projekt hinzugefügt werden.

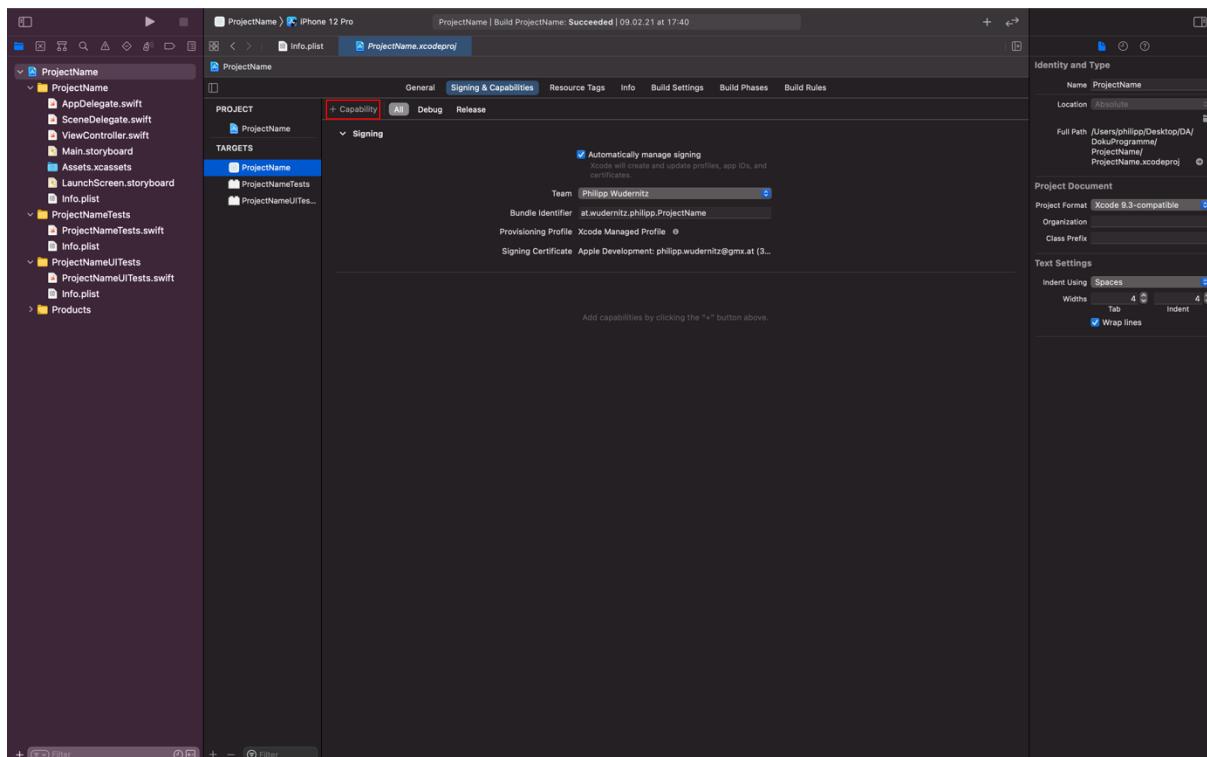


Abbildung 37: Xcode Capabilities

Dazu muss zunächst nach **Signing & Capabilities** in Xcode navigiert werden. Danach kann bei **+ Capability** eine neue Capability hinzugefügt werden, in diesem Fall also *Near Field Communication Tag Reading*.

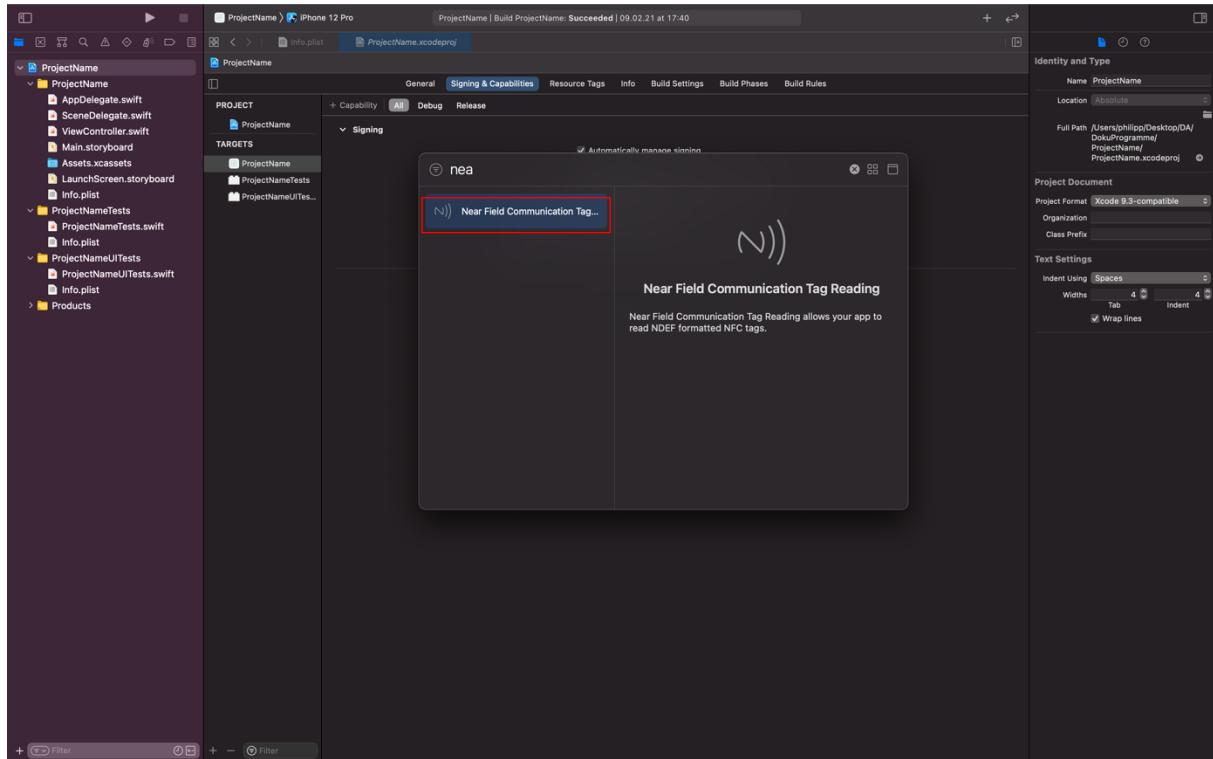


Abbildung 38: Xcode NFC Capability

Anschließend muss in der *Info.plist-Datei* (im Projektverzeichnis in Xcode zu finden) folgende Property (Eigenschaft) hinzugefügt werden: *Privacy – NFC Scan Usage Description*



Abbildung 39: NFC Property (Info.plist-File)

Im Value-Feld muss ein Grund angegeben werden, warum und wofür die NFC-Funktion verwendet wird. Mit folgendem Code wird die NFC-Session vorbereitet und alle wichtigen Funktionen werden bereitgestellt.

```
class MyViewController: UIViewController {
    // MARK: setup nfc scanner

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        print("mem warning")
    }

    func readerSessionDidBecomeActive(_ session: NFCNDEFReaderSession) {
        print("Became active!")
    }
}
```



```
}

func readerSession(_ session: NFCNDEFReaderSession,
    didInvalidateWithError error: Error) {
    print("The session was invalidated:
        \((error.localizedDescription))")
}

func readerSession(_ session: NFCNDEFReaderSession,
    didDetectNDEFs messages: [NFCNDEFMessage]) {
    print("reader session")
    var result = ""
    for payload in messages[0].records {
        result += String.init(data: payload.payload.advanced(by:
            3), encoding: .utf8) ?? "Format not supported"
        // Add additional code here
    }
}
}
```

Listing 18: NFC setup

5.9.3 Bluetooth Beacons (Bluetooth LE)

Bluetooth Beacons werden im Allgemeinen von folgendem Parametern bestimmt:

- UUID = Unterscheidung von Beacons im eigenen „Netzwerk“
- Major = Gruppierung von Beacons des „Netzwerks“ in kleinere Gruppen
- Minor = Gruppierung der Gruppen in noch kleinere Gruppen

Die UUID ist ein String in folgendem Format: "E2C56DB5-DFFB-48D2-B060-D0F5A71096E0", wobei die Major- und Minor-Values eine Zahl (Int) zwischen 1 und 65535 sein können. [43]

Folgende Grafik erklärt die Verwendung und Einsatz dieser Parameter anhand eines praktischen Beispiels.

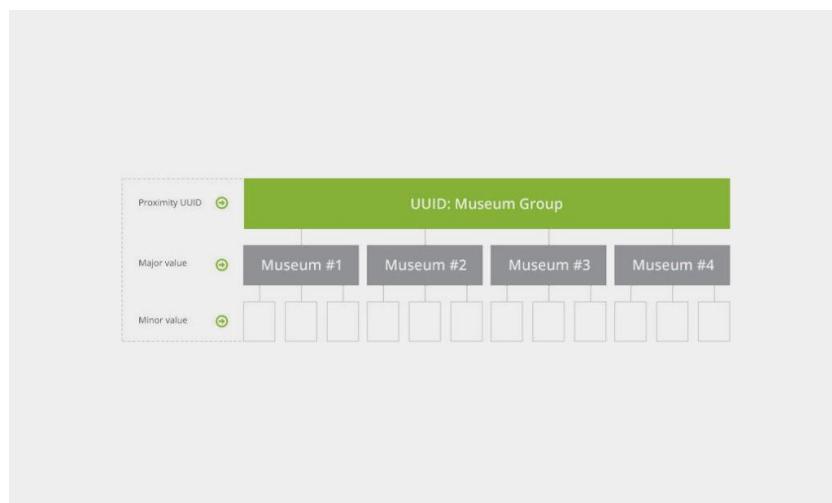


Abbildung 40: Bluetooth Beacons Parameter Beispiel [42]

Setup von Bluetooth Beacons in Xcode:

Um mit Bluetooth Beacons kommunizieren zu können, muss in der Info.plist-Datei (im Projektverzeichnis in Xcode zu finden) folgende Property (Eigenschaft) hinzugefügt werden: *Privacy – Bluetooth Always Usage Description*



Abbildung 41: Bluetooth-Beacons Properties (Info.plist-Datei)

Im Value-Feld muss ein Text hinzugefügt werden, welcher beschreibt, warum die Bluetooth-Funktion in der App verwendet werden soll. Mit folgendem Code wird das Erkennen von Bluetooth-Beacons vorbereitet und alle wichtigen Funktionen werden bereitgestellt.

```
class MyViewController: UIViewController, CLLocationManagerDelegate,
    CBCentralManagerDelegate {
    // Bluetooth-Beacons
    var bluetoothManager = CLLocationManager()
    // check if bluetooth is enabled
    var managerBLE = CBCentralManager()

    // Call in viewDidLoad()
    func configIBeacon() {
        print("Bluetooth-Beacons")
        bluetoothManager.delegate = self
        bluetoothManager.allowsBackgroundLocationUpdates = true
        bluetoothManager.requestAlwaysAuthorization()

        // Apple AirLocate Beacon on iPad
        let region = CLBeaconIdentityConstraint(uuid:
            UUID(uuidString:
                "E2C56DB5-DFFB-48D2-B060-D0F5A71096E0")!,
            major: CLBeaconMajorValue(1), minor:
            CLBeaconMinorValue(1))

        bluetoothManager.startRangingBeacons(satisfying: region)
    }

    func locationManager(_ manager: CLLocationManager, didRange
        beacons: [CLBeacon], satisfying beaconConstraint:
        CLBeaconIdentityConstraint) {
        switch beacons.first?.proximity {
            case .none:
                print("No Beacon in range")
            case .some(.immediate):
                print("A Beacon in immediate proximity")
            case .some(.near):
                print("A Beacon in near proximity")
        }
    }
}
```



```
        print("A Beacon in near proximity")
    case .some(.far):
        print("A Beacon in far proximity")
    case .some(.unknown):
        print("Could not find Beacon")
    default:
        print("No Beacon in range")
    }
}

// MARK: check if bluetooth is enabled

func bluetoothStatus() {
    managerBLE.delegate = self
}

func centralManagerDidUpdateState(_ central: CBCentralManager) {
    switch managerBLE.state {
    case CBManagerState.poweredOff:
        print("Powered Off")
        break
    case CBManagerState.poweredOn:
        print("Powered On")
        break
    case CBManagerState.unsupported:
        print("Unsupported")
        break
    case CBManagerState.resetting:
        print("Resetting")
        break
    case CBManagerState.unauthorized:
        print("Unauthorized")
        break
    case CBManagerState.unknown:
        print("Unknown")
    default:
        break
    }
}
}
```

Listing 19: Bluetooth-Beacons Setup

5.9.4 Geofencing

Geofencing erlaubt das Erkennen (betreten oder verlassen) von geografisch abgesteckten Bereichen („Zäune“). Dabei müssen folgende Parameter spezifiziert werden:

- Breitengrad
- Längengrad
- Radius

Der Breiten- und Längengrad sind in Grad angegeben und können auf beispielsweise Google Maps [45] für den gewünschten Geofencing-Bereich gefunden werden. Der Radius spezifiziert den Radius des Geofencing-Kreises, wobei der Breiten- und Längengrad den Mittelpunkt dieses Kreises bilden.

Setup von Geofencing in Xcode:

Um die Position des Benutzers tracken zu können, müssen in der Info.plist-Datei (im Projektverzeichnis in Xcode zu finden) folgende Properties (Eigenschaften) hinzugefügt werden:

- *Privacy – Location Always and When In Use Usage Description*
- *Privacy – Location Always Usage Description*
- *Privacy – Location Temporary Usage Description Dictionary*
- *Privacy – Location When In Use Usage Description*

Privacy - Location Always and When In Use Usage Des...	String	Timetracking always monitors you!
Privacy - Location Always Usage Description	String	We want your location!
> Privacy - Location Temporary Usage Description Dictio...	Dictionary	(0 items)
Privacy - Location When In Use Usage Description	String	Timetracking needs your location in order to use Geofencing and Bluetooth Beacons!

Abbildung 42: Geofencing Properties (Info.plist-Datei)

Hier muss ebenfalls im Value-Feld der Grund für die Verwendung von Geofencing angegeben werden. Mit folgendem Code kann Geofencing in einer iOS-App konfiguriert werden.

```
class MyViewController: UIViewController, CLLocationManagerDelegate
{
    // Geofence
    var locationManager = CLLocationManager()

    // Call in viewDidLoad()
    func confGeofence() {
        print("Geofence")
        locationManager.delegate = self
        locationManager.allowsBackgroundLocationUpdates = true
        locationManager.requestAlwaysAuthorization()
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        locationManager.startUpdatingLocation()

        let geofenceRegion = CLCircularRegion(center:

```



```
    CLLocationCoordinate2D(latitude: 48.52508886, longitude:  
        16.35610970), radius: 100, identifier: "MyLocation")  
    locationManager.startMonitoring(for: geofenceRegion)  
}  
  
func locationManager(_ manager: CLLocationManager,  
    didUpdateLocations locations: [CLLocation]) {  
    locationManager.stopUpdatingHeading()  
}  
  
func locationManager(_ manager: CLLocationManager,  
    didEnterRegion region: CLRegion) {  
    print("Entered Region")  
}  
  
func locationManager(_ manager: CLLocationManager, didExitRegion  
    region: CLRegion) {  
    print("Exited Region")  
}  
}
```

Listing 20: Geofencing Setup

5.10 User Defaults

Mit der Hilfe von User Defaults können Informationen / Einstellungen des Benutzers der App gespeichert und wieder abgerufen werden, auch wenn die App bereits geschlossen wurde. User Defaults dürfen nicht als Speicher für sicherheitsrelevante Informationen, wie z.B. Username, Passwort oder Abos verwendet werden, da auf die gespeicherten Informationen auch von außerhalb der App zugegriffen werden kann und diese daher auch geändert werden können. User Defaults sollten beispielsweise für folgende Dinge verwendet werden:

- Einstellungen, welche der Benutzer in der App ausgewählt hat (sofern nicht sicherheitskritisch)
- UI-Präferenzen (z.B. Dark Mode)

Apple stellt für das Speichern von User Defaults eine gleichnamige Klasse (*UserDefaults*) zur Verfügung, welche auch ohne Apple Developer Account verwendet werden kann. [40] Folgender Code zeigt das Erstellen einer *UserDefault*-Instanz, das Schreiben eines Werts auf User Defaults und das Lesen des zuvor geschriebenen Wertes.

```
let userDefaults = UserDefaults()  
userDefaults.setValue("Any Object (Number, String, ...)", forKey:  
    "myKey")  
let userValue = userDefaults.value(forKey: "myKey")
```

Listing 21: User Defaults erstellen

5.11 CoreData

CoreData ist ein lokales Datenbanksystem, welches das permanente Speichern von großen Datenmengen speziell für den offline-Betrieb ermöglicht.

Um CoreData im Xcode-Projekt benutzen zu können, sollte folgendes bei der Erstellung des Projekts gesetzt werden.

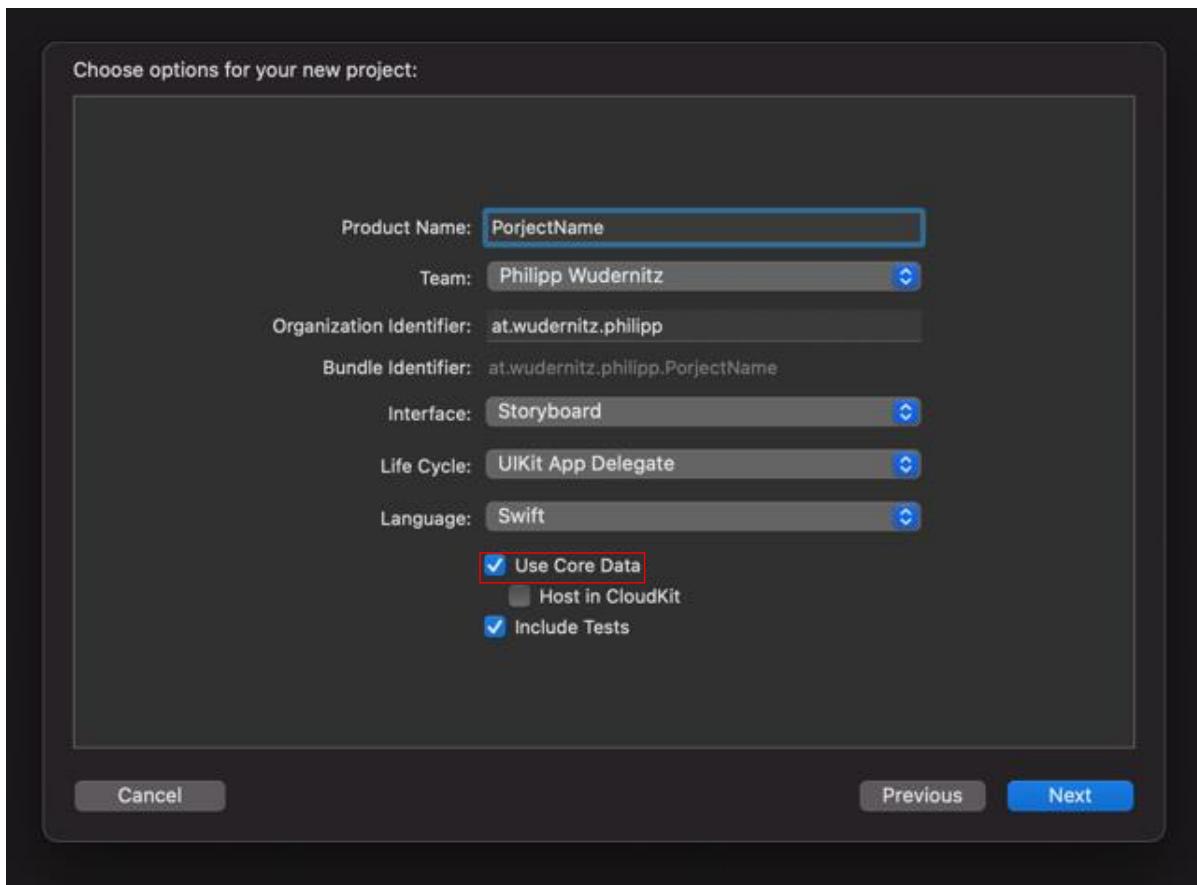


Abbildung 43: Xcode CoreData

Nach der Erstellung des Projekts sollte folgendes File im Projektverzeichnis auftauchen.

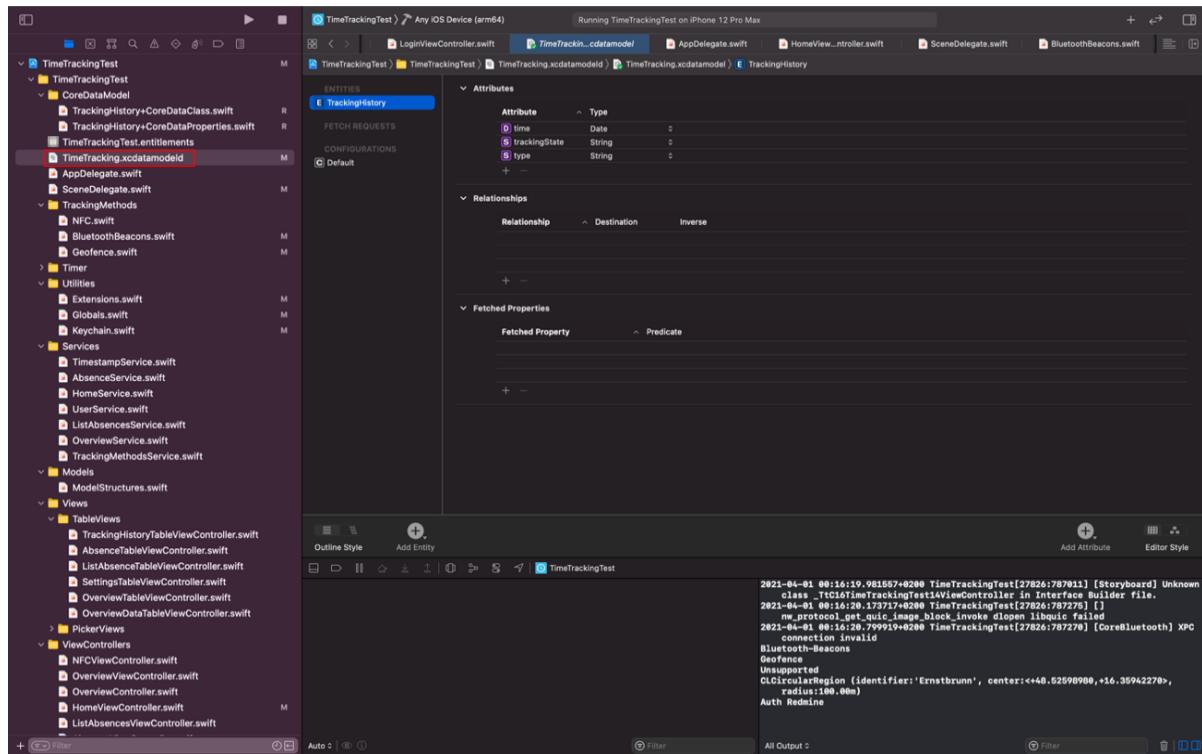


Abbildung 44: Xcode Core Data Model-File

Hier können *Entities* erstellt und diesen verschiedene Attributen mit verschiedenen Datentypen zugewiesen werden. Möchte man eine Entity in eine CoreData Klasse umwandeln, muss zuerst zwischen 3 Optionen gewählt werden:

- Manuel/None
- Class Definition
- Category/Extension

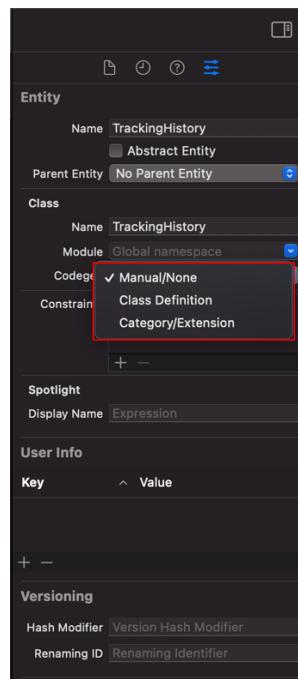


Abbildung 45: Core Data Codegen

Um den Code der CoreData Klasse zu erzeugen, muss folgendes getan werden.

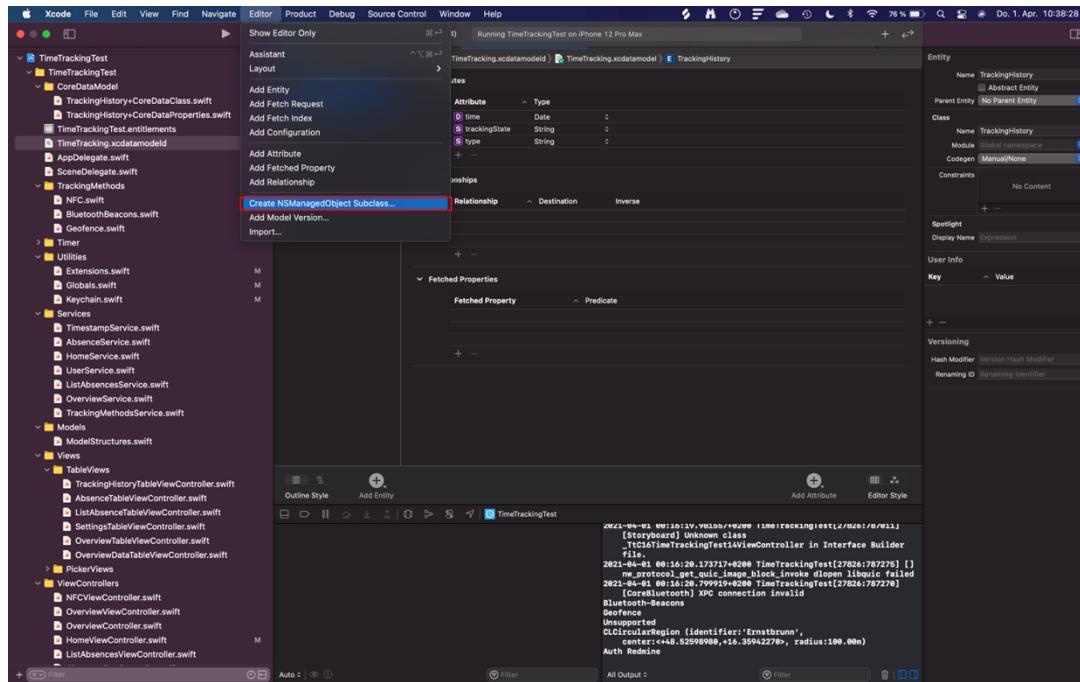


Abbildung 46: Xcode NSManagedObject erstellen

Danach ist das Model auszuwählen und mit **Next** den Speicherort im Projektverzeichnis zu wählen.

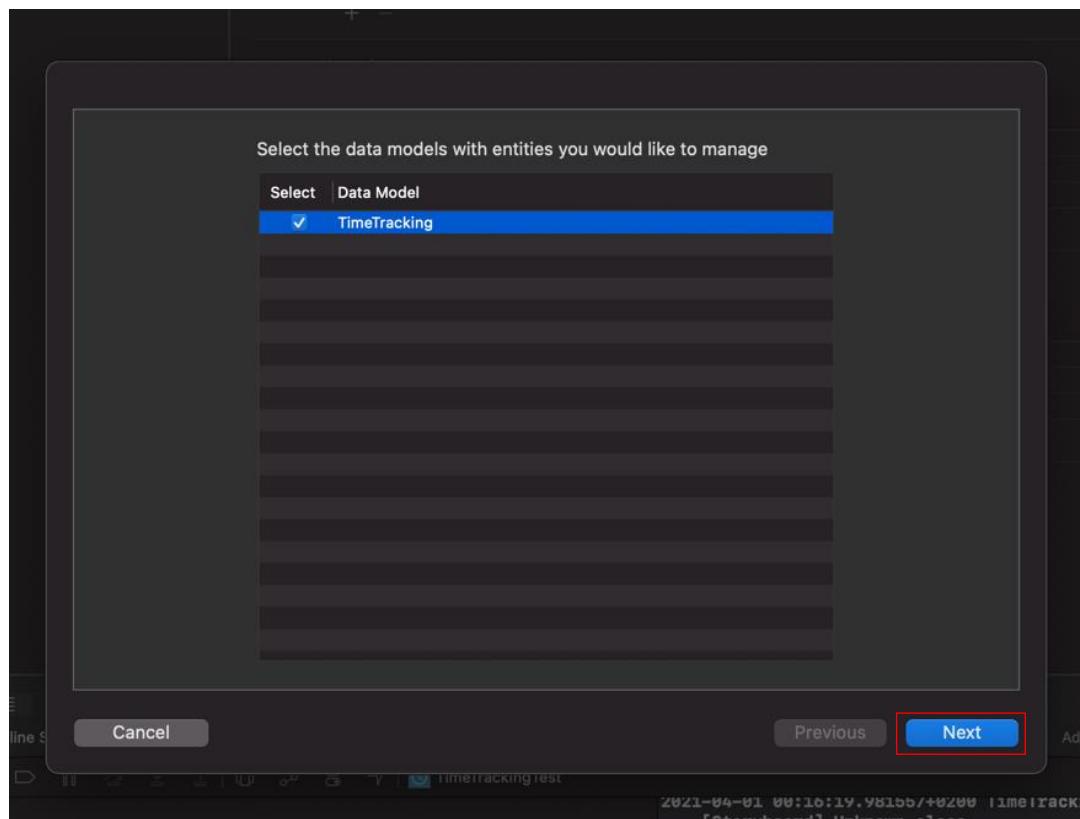


Abbildung 47: Xcode CoreData Model-Files speichern



Im Projektverzeichnis sollten 2 neue Files auftauchen.

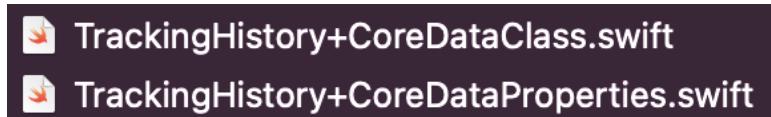


Abbildung 48: Xcode CoreData 2 neue Files

Der Code in diesen Files schaut wie folgt aus.

```
@objc(TrackingHistory)
public class TrackingHistory: NSManagedObject {

}
```

Listing 22: TrackingHistory+CoreDataClass.swift

```
extension TrackingHistory {

    @nonobjc public class func fetchRequest() ->
        NSFetchedResultsController<TrackingHistory> {
        return NSFetchedResultsController<TrackingHistory>(entityName:
            "TrackingHistory")
    }

    @NSManaged public var type: String?
    @NSManaged public var trackingState: String?
    @NSManaged public var time: Date?
}
```

Listing 23: TrackingHistory+CoreDataProperties.swift

Änderungen sollten lediglich im Extension-File (TrackingHistory+CoreDataProperties.swift) stattfinden. Im Class-Definition-File (TrackingHistory+CoreDataClass.swift) sollte nichts geändert werden. Bei Änderung der Entity müssen diese beiden Files neu generiert werden.

Die *AppDelegate* Klasse wird verwendet, um CoreData zu konfigurieren. Dabei wurden die Variable *persistentContainer* und die Funktion *saveContext()* automatisch bei der Projekterstellung hinzugefügt.

```
lazy var persistentContainer: NSPersistentContainer = {
    let container = NSPersistentContainer(name: "TimeTracking")
    container.loadPersistentStores(completionHandler: {
        (storeDescription, error) in
        if let error = error as NSError? {
            fatalError("Unresolved error \(error),
                \(error.userInfo)")
        }
    })
    return container
}()
```

Listing 24: AppDelegate.swift persistentContainer initialisieren

Die Variable `persistentContainer` ist mit dem Keyword `lazy` deklariert, welches `persistentContainer` nur initialisiert, wenn die Variable benötigt wird [37]. Der `persistentContainer` beinhaltet den CoreData Stack und vereinfacht die Erstellung und Verwaltung des Stacks. Mit der Funktion `saveContext()` können Änderungen des CoreData Stacks abgespeichert werden.

```
func saveContext() {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror),
                      \((nserror.userInfo))")
        }
    }
}
```

Listing 25: AppDelegate.swift saveContext()

Es wurde ebenfalls eine Funktion `fetchTrackingHistory()` hinzugefügt, welche alle Informationen aus dem CoreData Stack in dem Array `historyArray` speichert.

```
func fetchTrackingHistory() {
    do {
        historyArray = try
        context.fetch(TrackingHistory.fetchRequest())
    } catch {
        let nserror = error as NSError
        fatalError("Unresolved error \(nserror),
                  \((nserror.userInfo))")
    }
}
```

Listing 26: AppDelegate.swift fetchTrackingHistory()

Um Änderungen am CoreData Stack außerhalb der `AppDelegate` durchzuführen wird ein „Context“ [52] verwendet, auf welchen global zugegriffen werden kann.

```
let context = (UIApplication.shared.delegate as!
AppDelegate).persistentContainer.viewContext
```

Listing 27: Erstellen des Context (CoreData)



Soll beispielsweise ein Element aus dem CoreData Stack gelöscht werden, wird folgender Code verwendet.

```
context.delete(data)
```

Listing 28: Löschen eines Elements aus dem CoreData Stack

Da das Löschen des Elements einen Einfluss auf den CoreData Stacks hatte, werden folgende 2 Zeilen Code verwendet, um den Context zu speichern und das Array *historyArray* mit den aktuellen Informationen des CoreData Stacks zu erneuern.

```
(UIApplication.shared.delegate as? AppDelegate)?.saveContext()
```

Listing 29: Zugriff auf saveContext()

```
(UIApplication.shared.delegate as?
    AppDelegate)?.fetchTrackingHistory()
```

Listing 30: Zugriff auf fetchTrackingHistory()

5.12 Keychain

Apple's Keychain ermöglicht es sicherheitsrelevante Daten, wie z.B. Username oder Passwort, sicher zu speichern. Dabei werden diese Daten in einer verschlüsselten Datenbank namens „Keychain“ gespeichert. Da das Handhaben mit Keychain nicht sehr einfach ist, empfiehlt es sich, eine Library, wie z.B. Valet [32], zu benutzen. [41]

Mit folgendem Code kann ganz einfach ein Keychain Valet erstellt werden.

```
let myValet = Valet.valet(with: Identifier(nonEmpty:
    "myIdentifier")!, accessibility: .whenUnlocked)
```

Listing 31: Keychain Valet erstellen

Neben einem normalen Valet gibt es noch Shared Valets, iCloud Valets und Secure Enclave Valets. Das iCloud-Valet wurde bei der iOS-App verwendet.

```
let keychainIdentifier = Identifier(nonEmpty: "Login")!
let cloudValet = Valet.iCloudValet(with: keychainIdentifier,
    accessibility: .whenFirstUnlocked)
```

Listing 32: Keychain.swift iCloud Valet erstellen

Folgender Code dient zum Schreiben auf die Keychain.

```
do {  
    try myValet.setString("Value", forKey: "myKey")  
} catch {  
    print("Error Writing")  
}
```

Listing 33: Schreiben auf Valet

Im Gleichen Schema kann von dem Keychain Valet gelesen werden.

```
do {  
    try myValet.string(forKey: "myKey")  
} catch {  
    print("Error Reading")  
}
```

Listing 34: Lesen von Valet

5.13 User Interface

Zur Darstellung des User-Interfaces wurde der Simulator mit einem iPhone 12 Pro Max in Xcode verwendet. Das User-Interface ist ebenfalls im Dark Mode vorhanden. Dafür muss das iPhone in den Dark Mode gewechselt werden (Einstellung am iPhone). Laut den Apple Human Interface Guidelines [46] soll dies statt einem Toggle Button in der App gemacht werden. Das User-Interface der iOS-App ist lediglich in Deutsch vorhanden.

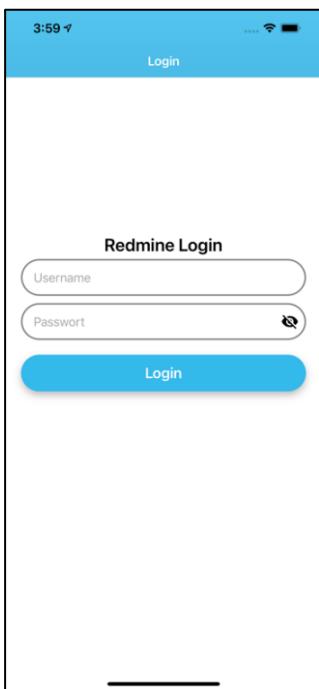
5.13.1 Launch Screen



Für die Farbe des Launch Screens wurde die Farbe des Tailored Apps Logos verwendet. Das Logo mit dem Text „Timetracking“ in der Mitte des Screens wurde selbst entworfen, wobei die Uhr im Logo von MaterialIO stammt.

Abbildung 49: Launch Screen

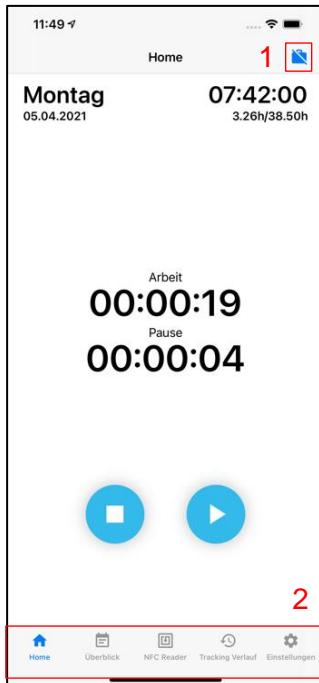
5.13.2 Login



Für die Redmine Authentifikation werden lediglich der Benutzername (Username) und das Passwort des Mitarbeiters benötigt. Im Passwort Feld hat man sich für einen Visibility Button entschieden, welcher das Passwort auch lesbar darstellen kann. Bei erfolgreicher Anmeldung wird der Benutzer auf den Home Screen weitergeleitet.

Abbildung 50: Login Screen

5.13.3 Home Screen



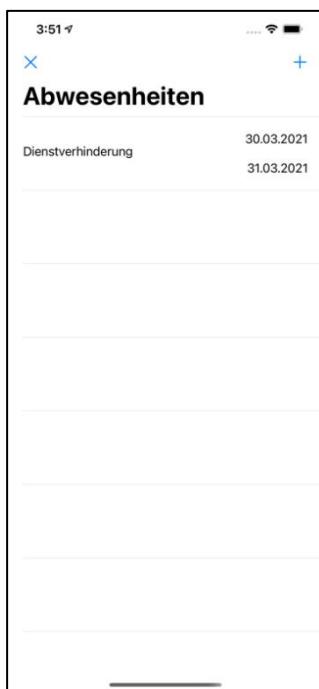
Der Home Screen zeigt folgende Informationen an:

- Heutigen Tag
- Heutiges Datum
- Solararbeitszeit
- Gearbeitete Stunden vs. Solararbeitsstunden der Woche
- Arbeitszeit (getrackt)
- Pausezeit (getrackt)
- Manuelle(r) Button(s) zum Timetracken

Es ist ebenfalls möglich, die eigenen Abwesenheiten zu managen. Dafür muss auf den durchgestrichenen Koffer (rot eingerahmt (1)) gedrückt werden. Für die Navigation in der App hat man sich, wie in den Apple Human Interface Guidelines [46] vorgeschrieben, für eine Tab Bar (rot eingerahmt (2)) entschieden.

Abbildung 51: Home Screen

5.13.3.1 Abwesenheiten



In diesem Screen ist es möglich, alle noch offenen Abwesenheiten einzusehen. Dafür wird ein Table View (Tabelle) eingesetzt. Mit dem Plus Button (rot eingerahmt (1)) kann eine neue Abwesenheit hinzugefügt werden. Mit dem Cancel Button (rot eingerahmt (2)) wird auf den Home Screen gewechselt.

Abbildung 52:
Abwesenheiten (Liste)



Abbildung 53: Neue Abwesenheit hinzufügen

Hier können neue Abwesenheiten hinzugefügt (rot eingehaumt (1)) werden. Folgende Informationen müssen angegeben werden:

- Abwesend bis zur nächsten Zeitaufzeichnung (wenn wieder aufgezeichnet wird ist die Abwesenheit vorbei)
- Ganztags (nur bei Urlaub und Dienstverhinderung möglich)
- Abwesenheitsgrund
- Beginn der Abwesenheit (wenn Ganztags: nur Datum, sonst Datum & Uhrzeit)
- Ende der Abwesenheit (wenn Ganztags: nur Datum, sonst Datum & Uhrzeit)
- Notizen (Relevante Zusatzinformationen)

5.13.4 Überblick

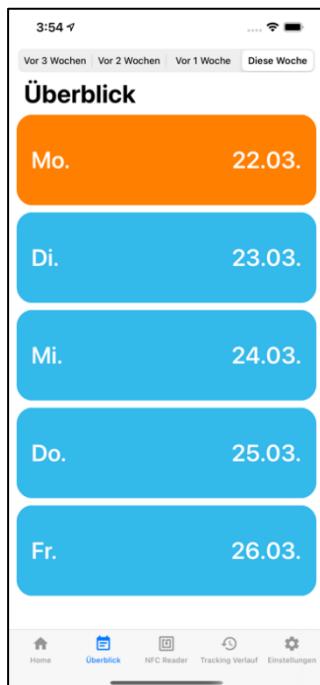


Abbildung 54:
Wochenüberblick

Im Überblick können wichtige Informationen über einen Arbeitstag eingesehen oder nachgesehen werden. Dabei können Arbeitstage über einen Monat in Wochenschritten verfolgt werden.

Für die Auswahl der Arbeitswoche wurde ein Segment Control (rot eingehaumt) verwendet.

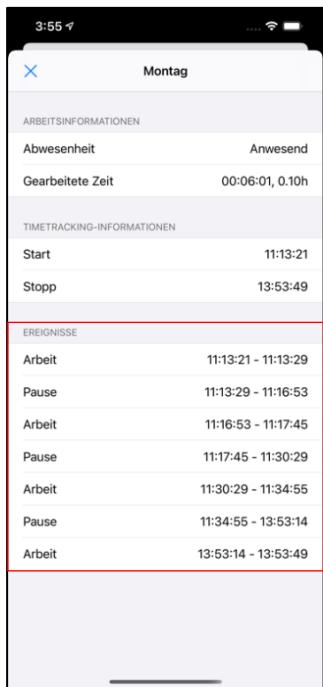


Abbildung 55:
Tagesüberblick

Nachdem ein Tag aus der Wochenübersicht ausgewählt wurde, kann dieser genauer betrachtet werden. Folgende Informationen sind diesem zu entnehmen:

- Abwesenheit (Abwesenheitsgrund)
- Gearbeitete Zeit
- Start- und Stopp-Zeit des Arbeitstages
- Ereignisse:
 - Arbeit
 - Pause
 - Abwesenheiten

Nur Informationen unter der Überschrift Ereignisse (rot eingerahmt) sind in der App lösbar.

5.13.5 NFC Reader

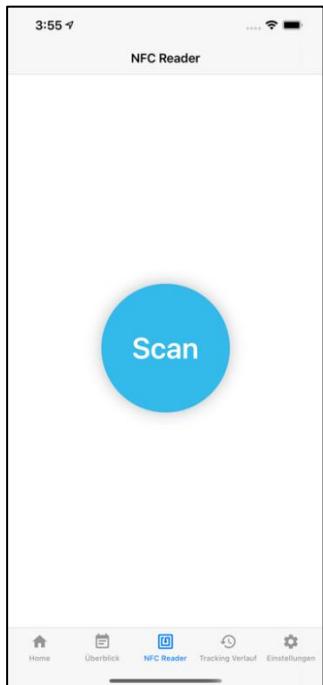
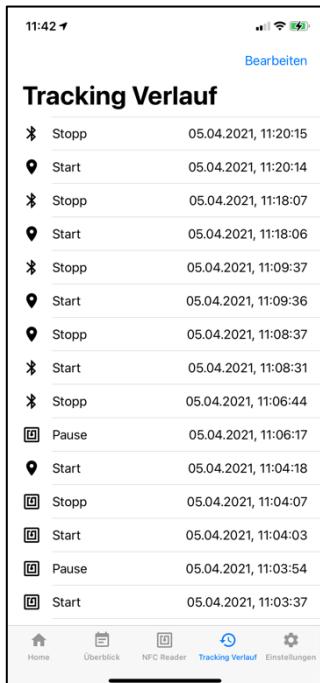


Abbildung 56: NFC Reader

Nach einem Druck auf den Scan Button wird ein Scanning des NFC Tags durchgeführt, wobei zunächst gewartet wird, bis ein NFC-Tag in der unmittelbaren Nähe ist (ca. 10cm). Nachdem ein passender NFC-Tag gefunden wurde, wird entweder das Timetracking gestartet, pausiert oder gestoppt.



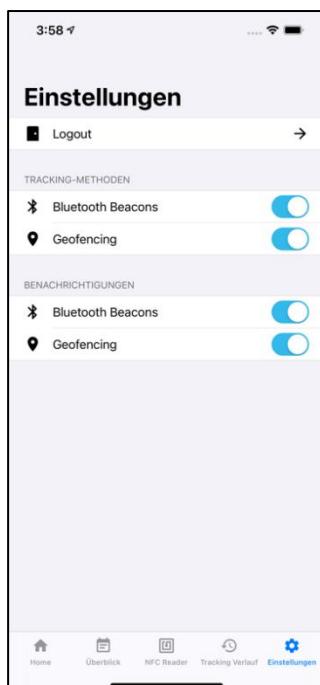
5.13.6 Tracking Verlauf



Hier werden die getrackten Ereignisse (NFC, Bluetooth Beacons, Geofencing; manuelle Tracks mit dem Tracking Button im Home Screen werden nicht gezählt) lokal am Gerät abgespeichert. Hierfür wurde CoreData verwendet, um die Daten lokal am iPhone zu halten.

Abbildung 57: Tracking Verlauf

5.13.7 Einstellungen



In den Einstellungen kann sich aus der App abgemeldet, die Tracking Methoden und die Notifications ein- oder ausgeschalten werden. Standardmäßig sind alle Switches ausgeschaltet.

Abbildung 58: Einstellungen

5.14 Funktionen

5.14.1 Libraries

Für die Entwicklung der iOS-App wurde lediglich eine Library verwendet:

- Valet (Version 4.1.1) [32]:
 - Vereinfacht die Verwendung von Apple's Keychain um sicherheitsrelevante Informationen (Username, Passwort) zu speichern.

2 andere Libraries wurden getestet, allerdings durch Änderungen des User-Interfaces nie eingesetzt:

- TinyConstraints [33]:
 - Vereinfacht das Erstellen von Constraints für UI-Elemente.
- Charts [34]:
 - Stellt Diagramme / Statistiken zur Verfügung, welche in der App verwendet werden können.

Valet kann mit dem Swift Package Manager installiert werden. Dazu muss der Github Clone Code kopiert werden:

- <https://github.com/square/Valet> im Browser öffnen
- Auf *Code* klicken
- <https://github.com/square/Valet.git> kopieren

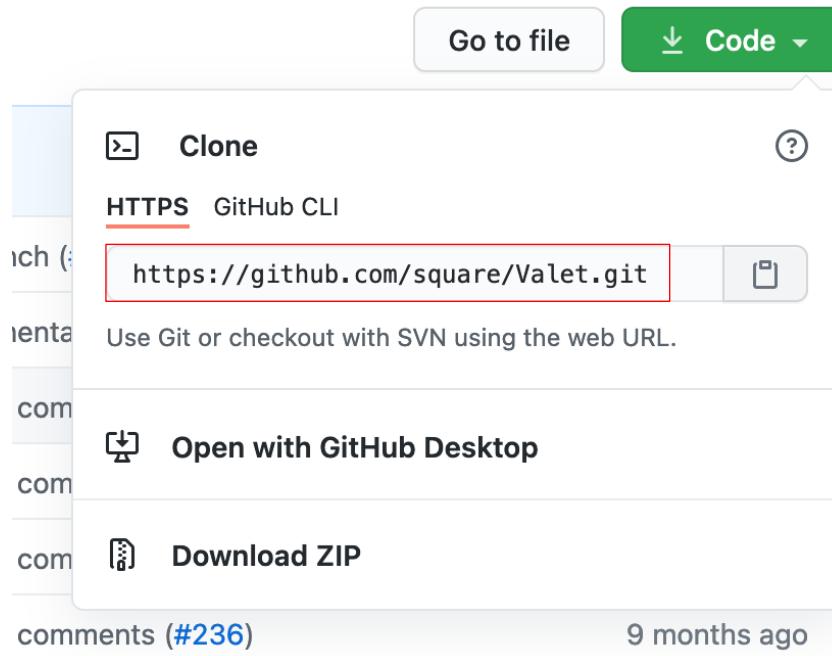


Abbildung 59: Valet Clone Code Link



Für den nächsten Schritt muss nach Xcode gewechselt werden, um die *Swift Package Dependency* zu installieren.

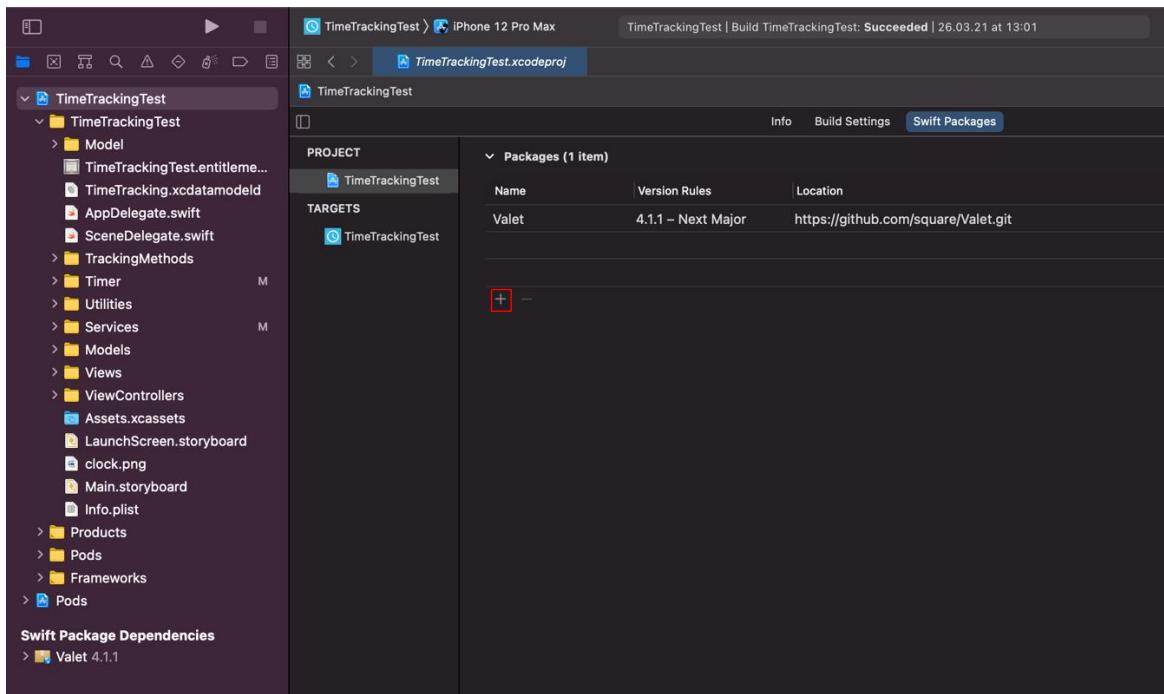


Abbildung 60: Swift Package Manager

Wenn aufs **Plus (+)** geklickt wird, kann eine neue *Swift Package Dependency* hinzugefügt werden. In diesem Fall *Valet*.

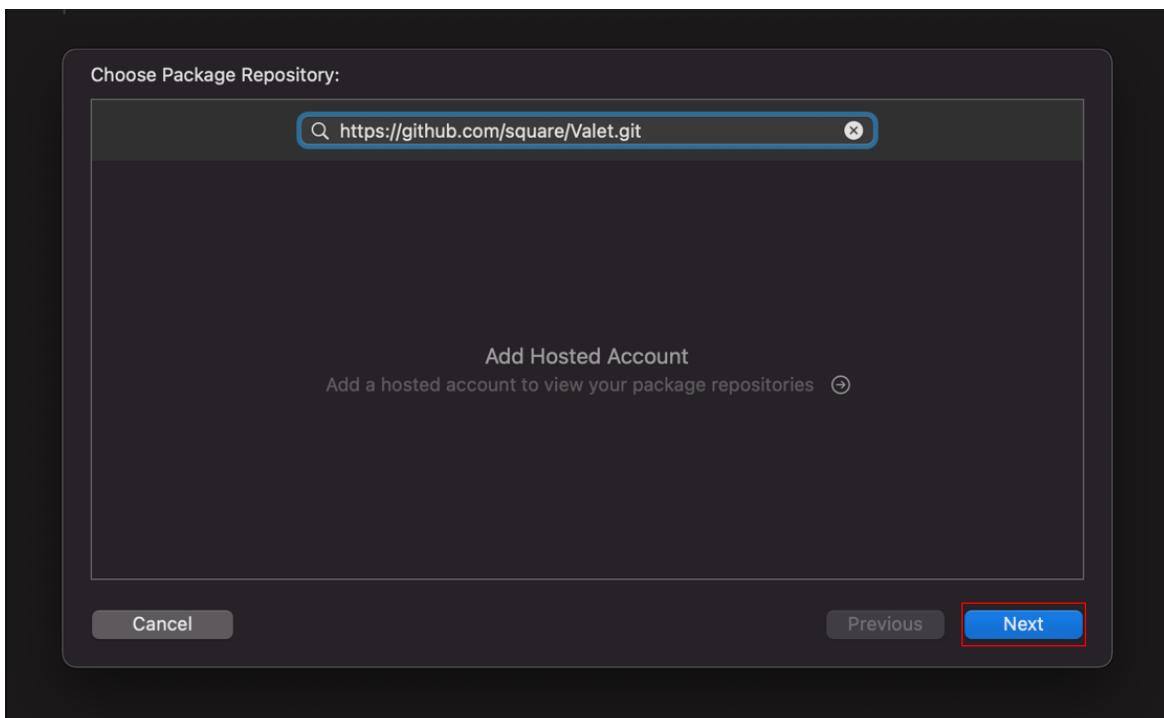


Abbildung 61: Swift Package Dependency hinzufügen

Falls es zu keinen Problemen kommt, sollte im Projektverzeichnis die *Swift Package Dependency Valet* angezeigt werden.

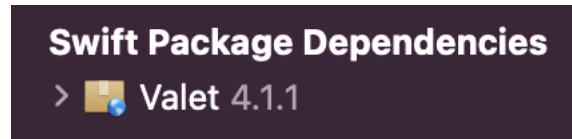


Abbildung 62: Valet als Swift Package Dependency

5.14.1.1 Pod-File

Pod-Files werden verwendet, um Cocoapods-Libraries [35] zum Projekt hinzuzufügen. Im Pod-File werden diese spezifiziert. Cocoapods ist ein *Dependency Manager* für Swift und Objective-C, welcher mittlerweile über 81000 Libraries zur Verfügung stellt.

Um ein Pod-File zu erstellen, muss folgendes getan werden:

- Im Terminal muss mit ***cd [Projektpfad]*** in den Projektordner navigiert werden.
- Danach kann mit ***pod init*** ein neues Pod-File erstellt werden (das File trägt den Namen ***Podfile***).
- Im Pod-File muss folgendes eingefügt werden: ***target '[Projektname]' do end***, wobei zwischen ***do*** und ***end*** einige Zeilen Platz sein sollte.

```
cd /Users/philipp/Desktop/DA/timetracking/IOS_TestAPPS/Apps
```

Abbildung 63: Terminal in den Projektordner navigieren

```
pod init
```

Abbildung 64: Terminal Pod-File erstellen

Name	Änderungsdatum
Podfile	21.09.20, 19:08
Podfile.lock	21.09.20, 19:09
> Pods	03.01.21, 15:10
> TimeTrackingTest	26.03.21, 13:01
TimeTrackingTest.xcodeproj	26.03.21, 10:37
TimeTrackingTest.xcworkspace	Heute, 09:33

Abbildung 65: erfolgreich erstelltes Pod-File im Projektverzeichnis



Danach sollte das Pod-File wie folgt aussehen. Hier wurden die Libraries TinyConstraints und Charts eingefügt, welche allerdings in der iOS-App nicht benutzt wurden.

```
# Uncomment the next line to define a global platform for your
# project
# platform :ios, '14.0'

target 'TimeTrackingTest' do
    # Comment the next line if you don't want to use dynamic
    frameworks
    use_frameworks!

    # Pods for TimeTrackingTest
    pod 'Charts'
    pod 'TinyConstraints'

end
```

Listing 35: Podfile

5.14.2 AppDelegate

Die *AppDelegate* Klasse ist der Startpunkt jeder iOS-App. Die Funktion *application(_:didFinishLaunchingWithOptions:)* ist deshalb auch die erste Funktion, die beim Starten der iOS-App ausgeführt wird. Die Methode

application(_:didFinishLaunchingWithOptions:) wurde lediglich für eine Funktion verwendet:

- Notification Center aktivieren

Authentifikation und Login wurden in eigenen Service Klassen realisiert und in den einzelnen *ViewController* Klassen aufgerufen.

```
func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    UNUserNotificationCenter.current().delegate = self
    return true
}
```

Listing 36: AppDelegate.swift Notifications aktivieren

5.14.3 SceneDelegate

Der folgende Code wird verwendet, um den *RootViewController* [39] auszuwählen, je nachdem ob der Benutzer bereits eingeloggt ist oder nicht. Falls der Benutzer eingeloggt ist, wird die Tab Bar konfiguriert, welche 5 Elemente in sich trägt und der erste *ViewController*, der in der Tab Bar angezeigt wird (Home Screen), wird spezifiziert. Darüber hinaus wird ebenfalls *timetracking* initialisiert. *timetracking* ist ein Objekt der Klasse *TimeTracking*, welche sich um die Timestamps der getrackten Arbeitszeit, Pausenzeit oder Abwesenheiten sowie die Darstellung der getrackten Arbeitszeit und Pausezeit im Home Screen kümmert.

```
func scene(_ scene: UIScene, willConnectTo session: UISceneSession,
options connectionOptions: UIScene.ConnectionOptions) {
    guard let winScene = (scene as? UIWindowScene) else { return
    }
    let nav = UINavigationController()
    nav.navigationBar.tintColor = UIColor.clear
    let win = UIWindow(windowScene: winScene)

    if try! cloudValet.containsObject(forKey:
        KeychainKeys.usernameKey) {
        userService.authWithRedmine(completion: { didAuth in
            if didAuth {
                DispatchQueue.main.async {
                    win.rootViewController = self.confTabBar()
                    win.makeKeyAndVisible()
                    timetracking.initialize()
                }
            }
        })
    } else {
        win.rootViewController =
        UINavigationController(rootViewController:
        LoginViewController())
        win.makeKeyAndVisible()
    }
    self.window = win
}
```

Listing 37: SceneDelegate.swift scene()



5.14.4 TabBar

Für die Navigation wird nach dem Login eine Tab Bar der Klasse *UITabBarController* verwendet. Diese beinhaltet 5 Elemente / Items:

- Home
- Überblick
- NFC Reader
- Tracking Verlauf
- Einstellungen

Mit folgendem Code können diese Items der Tab Bar hinzugefügt werden, sowie mit Symbolen / Icons leichter erkennbar gemacht werden.

```
func confTabBar() -> UITabBarController {
    let tabBarController = UITabBarController()

    let homeView = UINavigationController(rootViewController:
        HomeViewController())
    homeView.tabBarItem.image = assets.icons.home
    homeView.title = "Home"

    let overviewView = UINavigationController(rootViewController:
        OverviewViewController())
    overviewView.tabBarItem.image = assets.icons.overview
    overviewView.title = "Überblick"

    let nfcView = UINavigationController(rootViewController:
        NFCViewController())
    nfcView.tabBarItem.image = assets.icons.nfc
    nfcView.title = "NFC Reader"

    let historyView = UINavigationController(rootViewController:
        TrackingHistoryViewController())
    historyView.tabBarItem.image = assets.icons.history
    historyView.title = "Tracking Verlauf"

    let settingsView = UINavigationController(rootViewController:
        SettingsViewController())
    settingsView.tabBarItem.image = assets.icons.settings
    settingsView.title = "Einstellungen"

    tabBarController.setViewControllers([homeView, overviewView,
        nfcView, historyView, settingsView], animated: false)
    tabBarController.modalPresentationStyle = .fullScreen

    return tabBarController
}
```

Listing 38: Extensions.swift confTabBar()

Nach erfolgreichem Anmelden sieht die Tab Bar wie folgt aus.

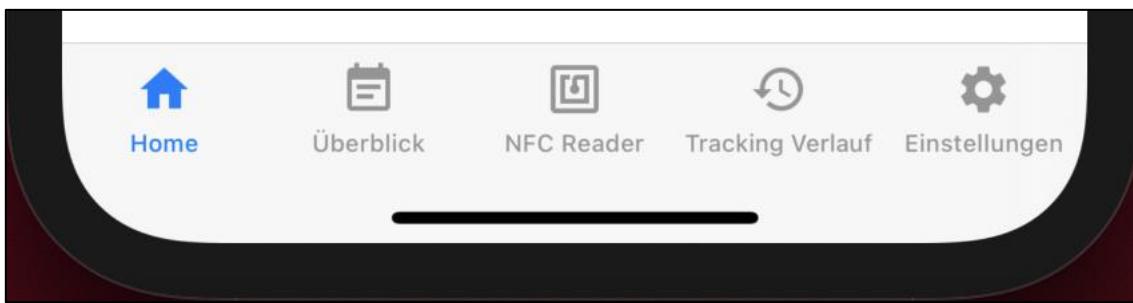


Abbildung 66: TabBar der iOS-App

5.14.5 Login

Um die Text Fields für Username und Passwort vom Design anpassen zu können, wurde eine Extension der Klasse `UITextField` geschrieben.

```
usernameTF.setLoginTextFieldStyle()
```

Listing 39: `LoginViewController.swift` Username Text Field anpassen in `setupLogin()`

Zusätzlich kann der Placeholder des Text Fields angepasst werden (Title: Username, Farbe: hellgrau). Hierfür kann folgender Code verwendet werden.

```
usernameTF.attributedPlaceholder = NSAttributedString(string:  
    "Username", attributes: [NSAttributedString.Key.foregroundColor:  
        UIColor.lightGray])
```

Listing 40: `LoginViewController.swift` Konfiguration des Username Text Fields in `setupLogin()`

Das Resultat ist ein Text Field mit hellgrauem Placeholder, einem grauen Rahmen mit abgerundeten Ecken und einem klaren Hintergrund, welcher die Farbe des Hintergrunds des Views (im Dark Mode oder Light Mode) durchlässt.



Abbildung 67: Username-Text Field angepasst

Nach demselben Schema kann auch das Passwort Text Field angepasst werden.



Der Activity Indicator View (Spinner), welcher auftaucht nachdem auf den Login Button gedrückt wurde, kann wie folgt realisiert werden.

```
spinner = UIActivityIndicatorView()
spinner.translatesAutoresizingMaskIntoConstraintsAutoresizingMaskIntoConstraints = false
spinner.backgroundColor = .clear
```

Listing 41: Spinner konfigurieren

Der Spinner der Klasse `UIActivityIndicatorView` soll dem Benutzer der App signalisieren, dass gerade Daten ausgetauscht werden und dass es womöglich eine gewisse Zeit dauern kann, bis diese Daten am iPhone ankommen.

Dieser Spinner kann mit folgenden 2 Zeilen Code gestartet und wieder gestoppt werden.

```
spinner.startAnimating()
spinner.stopAnimating()
```

Listing 42: Spinner starten und stoppen

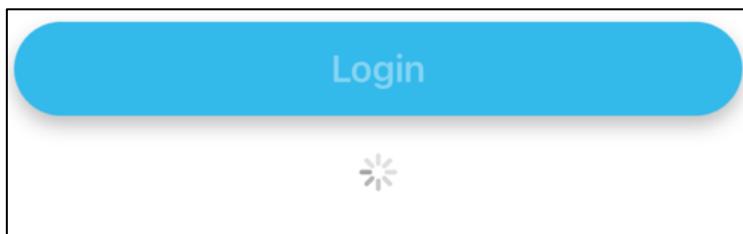


Abbildung 68: Login Spinner

Um das eingebende Passwort auf eine sicherheitsrelevante Mindestanforderung zu prüfen, wurde die Funktion `setPassword()` geschrieben, welche das Passwort nur akzeptiert, wenn es folgende Kriterien erfüllt:

- Mindestens 8 Zeichen lang
- Mindestens 1 Großbuchstabe
- Mindestens 1 Sonderzeichen

Folgender Code wird zur Realisierung hierfür verwendet.

```
func setPassword(password: String) -> Bool {
    // min 8 characters, min 1 capital letter, min 1 number, min
    1 special letter
    let passwordCheck = NSPredicate(format: "SELF MATCHES %@", "^(?=.*[a-z])(?=.*[$@#$!%*?&])[A-Za-z\\d$@#$!%*?&]{8,}")
    if passwordCheck.evaluate(with: password) {
        self.password = password
        return true
    } else {
        return false
    }
}
```

```
    }
}
```

Listing 43: UserService.swift setPassword()

Da Redmine für die Authentifikation verwendet wird, wurde ein API-Call erstellt, welcher sich mit dem Username und Passwort authentifiziert und die Antwort abspeichert.

```
func checkRedmineAuth(completion: @escaping ((UserRedmine?) ->
Void)) {
    // login information
    let url = "https://redmine.tailored
        apps.com/my/account.json/"
    let loginData = String(format: "%@:%@", username!,
        password!).data(using: String.Encoding.utf8)
    let base64LoginData = loginData?.base64EncodedString()

    // create request
    let urlString = URL(string: url)
    var request = URLRequest(url: urlString!)
    request.httpMethod = "GET"
    request.setValue("Basic \\"(base64LoginData!)\\\"", forHTTPHeaderField: "Authorization")

    // making the request
    URLSession.shared.dataTask(with: request) { data, response,
        error in
        guard let data = data else {
            print("Data is nil")
            completion(nil)
            return
        }

        if let error = error {
            print(error)
            completion(nil)
            return
        }

        do {
            let result = try
                JSONDecoder().decode(UserRedmine.self, from:
                    data)
            if let httpStatus = response as? HTTPURLResponse {
                if httpStatus.statusCode == 200 {
                    self.userRedmine = result
                    self.keychain()
                    completion(result)
                }
            }
        }
    }.resume()
}
```



```
        }
    }
} catch {
    print("failed to convert LoginData")
    completion(nil)
}
.resume()
}
```

Listing 44: UserService.swift checkRedmineAuth()

Dieser API-Call wird im *LoginViewController* zusammen mit einigen anderen API-Calls (Authentifizierung mit dem eigenen Backend, Verifizierung / Registrierung des Benutzers) aufgerufen.

In der Antwort von Redmine stehen relevante Informationen drinnen, wie z.B.:

- Vorname
- Nachname
- E-Mail
- API-Key (wichtig für Authentifikation mit dem eigenen Backend)

5.14.6 Home Screen

Folgende Services werden in der Methode *viewWillAppear()* des *HomeViewControllers* aufgerufen:

- *allocTime*: Stellt die Sollarbeitszeit des heutigen Tages zur Verfügung
- *effTime*: Stellt die gearbeitete Arbeitszeit vs. der Sollarbeitszeit der gesamten Woche zur Verfügung

```
homeService.allocTime(date: nil, completion: { statusCode in
    DispatchQueue.main.async {
        if statusCode == 200 {
            if !timetracking.getIsTimeTracking() &&
                timetracking.timeWork! == 0 {
                self.timeToWorkLB.text = homeService.allocTime?.diff
                ?? "00:00:00"
                timetracking.timeToWork =
                    homeService.allocTime?.diff_sec ?? 0
            }
        }
    }
})
homeService.effTime(completion: { statusCode in
    DispatchQueue.main.async {
        if statusCode == 200 {
            let effTime = Double((homeService.effTime?.effTime ??

```

```

        0))/60/60
        let allocTime = Double((homeService.effTime?.allocTime
            ?? 0))/60/60
        self.weekHours.text = String(format: "%.2f", effTime) +
            "h/" + String(format: "%.2f", allocTime) + "h"
    }
}
}
)

```

Listing 45: HomeViewController.swift AllocTime und EffTime Service in viewWillAppear()

An freien Tagen ist die Sollarbeitszeit 0 Stunden. In einer freien Woche ist die gesamte Wochensollarbeitszeit ebenfalls 0 Stunden. Dies ist in der unteren Abbildung zu sehen. Trotzdem kann aufgezeichnet werden. Dies ist erkennbar durch die 0.05 Stunden, welche der gearbeiteten Zeit in dieser Woche entsprechen.

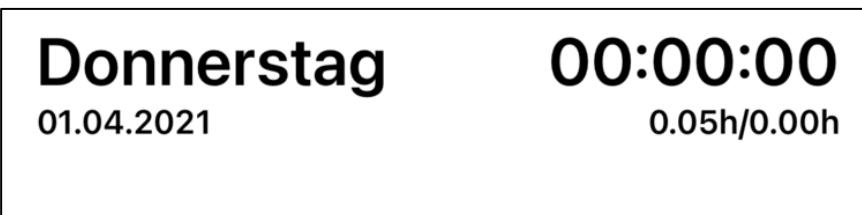


Abbildung 69: Sollarbeitszeit und Wochensollarbeitszeit

Um den Timer in der Mitte des Home Screens, sowie den Button darunter, zu koordinieren, wird das Objekt *timetracking* der Klasse *TimeTracking* verwendet, welches sich um das Timetracking kümmert. Falls Timetracking gestartet oder gestoppt werden soll, sind 2 Methoden bereits vorhanden.

```

func startTimeTracking() {
    timeStartDate = Date.init(timeIntervalSinceNow: 0)
    whichTimer = .Start
    isTimeTracking = true
    timestampService.writeTimestamp(reason: "Start", fulltime:
        false, completion: { statusCode in
            if statusCode == 200 {
                print("Success")
            }
        }
    )
}

```

Listing 46: Timer.swift startTimeTracking()

```

func stopTimeTracking() {
    isTimeTracking = false
    offset = timePause! + timeWork!
    timestampService.writeTimestamp(reason: "Stop", fulltime: false,

```



```
completion: { statusCode in
    if statusCode == 200 {
        print("Success")
    }
}
```

Listing 47: Timer.swift stopTimeTracking()

Diese 2 Methoden werden von NFC, Bluetooth Beacons, Geofencing und manuellen Tracking (Tracking Button(s) am Home Screen) verwendet, um Timetracking zu starten oder zu stoppen.

In der Timer Klasse *TimeTracking* stehen ebenfalls 2 Methoden zur Verfügung, welche die Arbeitszeit und Pausenzeit am Home Screen anzeigen. Dabei werden folgende 2 Methoden von einem Timer, welcher jede Sekunde einen Interrupt auslöst, aufgerufen.

```
func convWorkTime() -> String {
    calcTime()
    let work = timeWork!

    let hrs = work / 3600
    let mins = (work % 3600) / 60
    let secs = (work % 3600) % 60

    return String(format: "%02d", hrs) + ":" + String(format:
        "%02d", mins) + ":" + String(format: "%02d", secs)
}
```

Listing 48: Timer.swift convWorkTime()

```
func convPauseTime() -> String {
    calcTime()
    let work = timePause!

    let hrs = work / 3600
    let mins = (work % 3600) / 60
    let secs = (work % 3600) % 60

    return String(format: "%02d", hrs) + ":" + String(format:
        "%02d", mins) + ":" + String(format: "%02d", secs)
}
```

Listing 49: Timer.swift convPauseTime()

Dank dieser Methoden kann die Arbeits- und Pausenzeit von Sekunden auf ein lesbareres Format (HH:mm:ss) konvertiert und am Home Screen angezeigt werden.

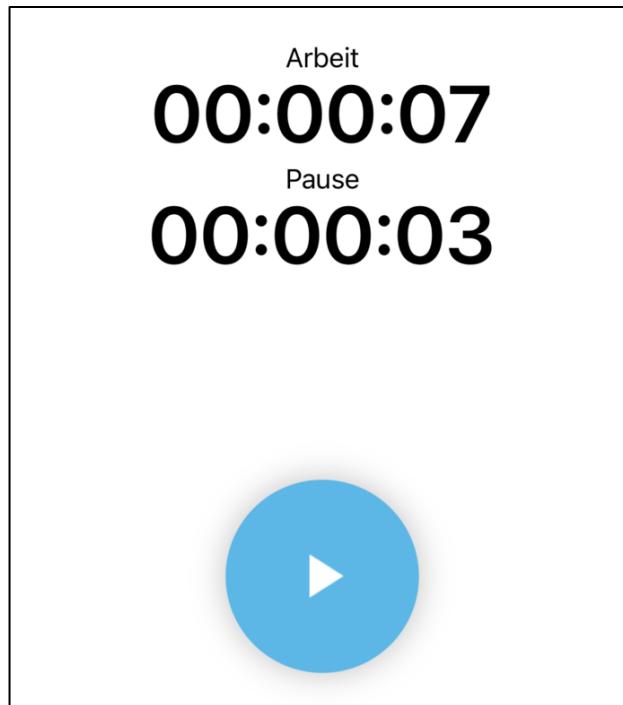


Abbildung 70: Arbeits- und Pausezeit

Um die Labels im Home Screen, welche die Arbeitszeit und die Pausenzeit anzeigen, im Sekundentakt zu updaten, kann ein Timer verwendet werden. Folgender Code zeigt das Initialisieren des Timers.

```
func confUpdateLabelsByTimer() {  
    timerUpdatingLabels = Timer.scheduledTimer(timeInterval: 0.1,  
                                                target: self, selector: #selector(chooseTimerModeToDisplay),  
                                                userInfo: nil, repeats: true)  
}
```

Listing 50: HomeViewController.swift confUpdateLabelsByTimer()



Nach jeder Sekunde wird die Funktion `chooseTimerModeToDisplay()` aufgerufen, welche entweder die Arbeitszeit oder die Pausenzeit updatet.

```
@objc func chooseTimerModeToDisplay() {
    if timetracking.getWhatTimer() == .Start {
        updateWorkTime()
    } else {
        updatePauseTime()
    }
}
```

Listing 51: HomeViewController.swift chooseTimerModeToDisplay()

5.14.6.1 Abwesenheiten

Die aktuellen Abwesenheiten werden in der Methode `viewWillAppear()` geladen und gespeichert. Für die Darstellung der Abwesenheiten wird ein Table View verwendet. Folgender API-Call dient zur Abfrage der Abwesenheit vom Backend.

```
listAbsencesService.readListAbsences(completion: { statusCode in
    DispatchQueue.main.async {
        print(listAbsencesService.absencesList)
        self.tableView.reloadData()
        self.removeLoadingScreen(spinner: self.spinner)
    }
})
```

Listing 52: ListAbsenceViewController.swift Abwesenheiten laden in viewWillAppear()

Die Cell des Table Views hat 3 Darstellungsmöglichkeiten:

- Bis Timetracking:

Arztbesuch Untersuchung	01.04.2021, 14:17
	Bis Timetracking

Abbildung 71: Abwesenheit (Bis Timetracking)

- Ganztägig:

Urlaub Kroatien	15.04.2021
	16.04.2021

Abbildung 72: Abwesenheit (Ganztägig)

- Geplant:

Arztbesuch	01.04.2021, 14:15
Untersuchung	01.04.2021, 16:00

Abbildung 73: Abwesenheit (Geplant)

Um zwischen diesen 3 Darstellungen zu entscheiden wurde eine Subclass der Oberklasse `UITableViewCell` namens `ListAbsencesTableViewCell` erstellt, welche die Methode `layoutSubviews()` besitzt. Diese Methode wird aufgerufen, wenn die Table View Cell gerendert werden soll.

```
override func layoutSubviews() {
    super.layoutSubviews()

    let formatterDate = DateFormatter()
    formatterDate.dateFormat = "yyyy-MM-dd HH:mm:ss"

    let formatterString = DateFormatter()
    formatterString.dateFormat = "dd.MM.yyyy, HH:mm"

    let formatter = DateFormatter()
    formatter.dateFormat = "HH:mm:ss"

    if reason == "Urlaub" || reason == "Dienstverhinderung" {
        formatterString.dateFormat = "dd.MM.yyyy"
    }

    if let begin = begin {
        guard let date = formatterDate.date(from: begin) else {
            beginLB.text = ""
            return
        }

        beginLB.text = formatterString.string(from: date)
    }

    if let fulltime = fulltime {
        if fulltime == 1 {
            endLB.text = "Bis Timetracking"
        } else {
            if let end = end {
                guard let date = formatterDate.date(from: end) else {
                    endLB.text = ""
                    return
                }
            }
        }
    }
}
```



```
        }
        endLB.text = formatterString.string(from: date)
    }
}
}
```

Listing 53: ListAbsenceTableViewController.swift layoutSubviews()

5.14.6.2 Neue Abwesenheit

Möchte man eine neue Abwesenheit hinzufügen, muss immer ein Abwesenheitsgrund spezifiziert werden.

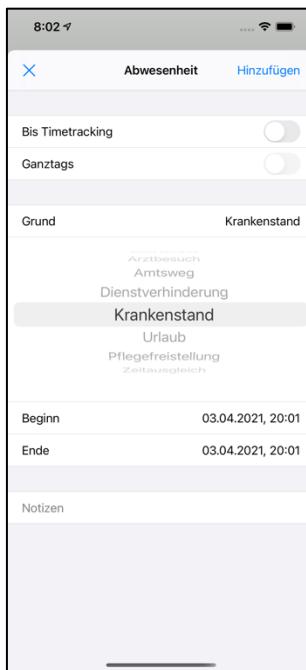


Abbildung 74: Abwesenheitsgrund auswählen

In der oberen Abbildung ist erkennbar, wie ein Abwesenheitsgrund ausgewählt werden kann. Hierfür wurde ein *UIPickerView* verwendet, welcher die Elemente (Abwesenheiten) aus einem Array namens *reasonArray* nimmt. Dieser Array besitzt folgende Items:

```
var reasonArray = ["Kein Grund", "Arztbesuch", "Amtsweg",
"Dienstverhinderung", "Krankenstand", "Urlaub",
"Pflegefreistellung", "Zeitausgleich"]
```

Listing 54: AbsenceViewController.swift Abwesenheitsgründe Array

Wird der *Hinzufügen* Button gedrückt, werden folgende API-Calls aufgerufen, je nachdem ob der Switch namens *Bis Timetracking* aktiviert wurde oder nicht. Die Funktion *reloadListAbsences()* bringt der Benutzer zum vorherigen View zurück und updatet den Table View und somit die Abwesenheiten.

```
@objc func saveAbsence() {
    if reasonLabel.text == "" || reasonLabel.text == "Kein Grund" {
        let alert = UIAlertController(title: "Kein
            Abwesenheitsgrund", message: "Sie müssen einen
            Abwesenheitsgrund auswählen", preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: .default,
            handler: { action in
                alert.dismiss(animated: true, completion: nil)
            }))
        self.present(alert, animated: true, completion: nil)
    } else {
        let formatterDate = DateFormatter()
        formatterDate.dateFormat = "yyyy-MM-dd"
        let formatterTime = DateFormatter()
        formatterTime.dateFormat = "HH:mm"

        var timeBegin: String? = formatterTime.string(from:
            datePickerBeginTime.date)
        var timeEnd: String? = formatterTime.string(from:
            datePickerEndTime.date)

        let dateBegin = formatterDate.string(from:
            datePickerBegin.date)
        let dateEnd = formatterDate.string(from: datePickerEnd.date)

        let reason = reasonLabel.text
        let comment = textFieldRemark.text

        let dateBeginPicker = datePickerBegin.date
        let dateEndPicker = datePickerEnd.date

        homeService.allocTime(date: dateBeginPicker, completion: {
            statusCode in
            print(statusCode as Any)
            timeBegin = homeService.allocTime?.start ?? "00:00:00"
            homeService.allocTime(date: dateEndPicker, completion: {
                statusCode in
                print(statusCode as Any)
                timeEnd = homeService.allocTime?.stop ?? "00:00:00"
                self.addAbsence(timeBegin: timeBegin, timeEnd:
                    timeEnd, dateBegin: dateBegin, dateEnd: dateEnd,
                    reason: reason!, comment: comment!)
            })
        })
    }
}
```

Listing 55: AbsenceViewController.swift saveAbsence()



```
func addAbsence(timeBegin: String?, timeEnd: String?, dateBegin: String, dateEnd: String, reason: String, comment: String) {
    DispatchQueue.main.async {
        if timeBegin == "00:00:00" || timeEnd == "00:00:00" {
            let alert = UIAlertController(title: "Keine Arbeit",
                message: "An den von Ihnen ausgewählten Daten ist keine Arbeit", preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "OK", style: .default, handler: { action in
                alert.dismiss(animated: true, completion: nil)
            }))
            self.present(alert, animated: true, completion: nil)
        } else {
            if self.swTimetracking.isOn {
                absenceService.writeAbsence(stamp: timeBegin!,
                    date: dateBegin, reason: reason, fulltime: true,
                    comment: comment, completion: { _ in
                        self.reloadListAbsences()
                })
            } else {
                absenceService.writeAbsence(stamp: timeBegin!, date:
                    dateBegin, reason: reason, fulltime: false,
                    comment: comment, completion: { _ in
                        absenceService.writeAbsence(stamp: timeEnd!,
                            date: dateEnd, reason: "Stop", fulltime:
                            false, comment: comment, completion: { _ in
                                self.reloadListAbsences()
                            })
                })
            }
        }
    }
}
```

Listing 56: AbsenceViewController.swift addAbsence()

Falls kein Abwesenheitsgrund ausgewählt wurde, wird ein Alert erstellt und dem Benutzer gezeigt. Dieser fordert den Benutzer auf, einen Abwesenheitsgrund auszuwählen, da sonst keine neue Abwesenheit erstellt werden kann. Falls an den Daten, an welchem der Benutzer eine Abwesenheit hinzufügen möchte, kein Arbeitstag ist (Wochenende, Feiertag, ...), wird ebenfalls ein Alert erstellt.

5.14.7 Überblick

Mit folgendem Code werden die Informationen von einem ausgewählten Tag / Datum abgefragt. Die Variable *dateString* entspricht dabei dem Datum im passenden Format als String.

```

overviewService.readOvData(dateString: dateString!, completion: {
    statusCode in
    DispatchQueue.main.async {
        if statusCode == 200 {
            self.removeLoadingScreen(spinner: self.spinner)
            self.setupOverview()
            self.view.sendSubviewToBack(self.tableView)
        } else {
            let alert = UIAlertController(title: "Keine Einträge vorhanden", message: "Es sind keine Einträgen für den ausgewählten Tag vorhanden", preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "Zurück", style: .default, handler: { action in
                alert.dismiss(animated: true, completion: {
                    self.dismissVC()
                })
            }))
            self.present(alert, animated: true, completion: nil)
        }
    }
})

```

Listing 57: OverviewController.swift Überblicksdaten des ausgewählten Tages abfragen in viewDidLoad()

Hier wird ebenfalls ein Alert erstellt, welcher ein Pop-Up zeigt, falls die Abfrage der Informationen des Tages nicht funktioniert hat. Dieses Pop-Up sieht wie folgt aus.

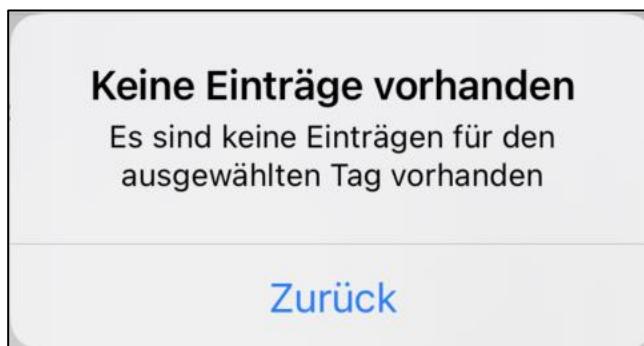


Abbildung 75: PopUp im Überblick

Es ist ebenfalls möglich, Einträge zu löschen. Dafür wird folgende Methode, welche bereits vom *UITableView* bereitgestellt wurde, erweitert.

```

func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
    if editingStyle == .delete {
        overviewService.overview?.events?.remove(at: indexPath.row)
        overviewService.overview?.type?.remove(at: indexPath.row)
        let idEntry = overviewService.overview?.id![indexPath.row]
    }
}

```



```
        overviewService.overview?.id?.remove(at: indexPath.row)
        tableView.deleteRows(at: [indexPath], with: .fade)

        print("delete Entry")

        overviewService.deleteEntry(id: idEntry!, completion: {
            statusCode in
            print(statusCode as Any)
        })
    }
}
```

Listing 58: OverviewDataTableViewController.swift Einträge aus Tagesüberblick löschen

Diese Methode ermöglicht den Benutzer mit einem Swipe nach links einen beliebigen Eintrag zu löschen.

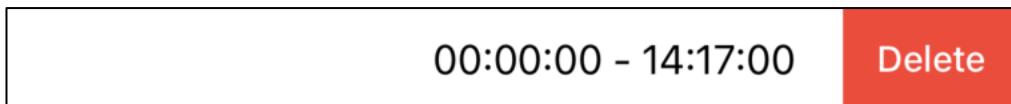


Abbildung 76: Löschen eines Events im Überblick

Bemerkung: Nur Einträge unter der Überschrift *Ereignisse* können in der iOS-App gelöscht werden. Falls mehr als nur Ereignisse gelöscht werden sollen, muss in die Web-App gewechselt werden.

5.14.8 NFC Reader

Gescannte NFC-Tags können 3 Werte beinhalten:

- „Start“ => Timetracking wird gestartet
- „Stop“ => Timetracking wird gestoppt
- „Pause“ => Timetracking wird pausiert

Diese werden in der Struktur *resultNFC* gespeichert und mit dem gescannten Wert verglichen.

```
struct resultNFC {
    static var content: String?
    static let startTimer = "Start"
    static let stopTimer = "Stop"
    static let pauseTimer = "Pause"
}
```

Listing 59: Globals.swift resultNFC Struktur

Ja nach dem welcher Wert vom NFC-Tag gelesen wurde, wird Timetracking gestartet, gestoppt oder pausiert. Nach dem Scan eines NFC-Tags werden Daten im CoreData Stack abgespeichert, damit diese im Tracking Verlauf angezeigt werden können. Dafür wird folgende Funktion verwendet.

```
func storeNFCScanInCoreData() {
    let object = TrackingHistory(context: context)
    object.type = "NFC"
    object.trackingState = getTrackingState()
    object.time = Date()
    (UIApplication.shared.delegate as? AppDelegate)?.saveContext()
}
```

Listing 60: NFC.swift storeNFCScanInCoreData()

Nach diesem Schema werden ebenfalls die Daten von Bluetooth Beacons und Geofencing gespeichert, allerdings wird die Funktion `getTrackingState()` nicht verwendet, da es bei Bluetooth-Beacons und Geofencing nur „Start“ und „Stop“ gibt.

5.14.9 Tracking Verlauf

Wird auf *Bearbeiten* und anschließend *Löschen* gedrückt, wird ein Action Sheet angezeigt, welches den Benutzer zur Sicherheit fragt, ob alle Einträge gelöscht werden sollen.

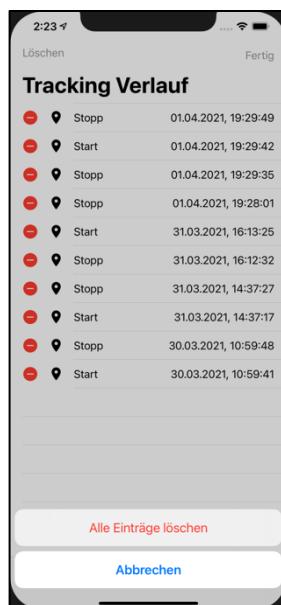


Abbildung 77: ActionSheet im Tracking Verlauf



Mit folgendem Code kann dies realisiert werden.

```
func showActionSheet() {
    let actionsheet = UIAlertController(title: nil, message: nil,
        preferredStyle: .actionSheet)
    actionsheet.addAction(UIAlertAction(title: "Abbrechen", style:
        .cancel, handler: { action in
            print("Dismiss Actionsheet")
        }))
    actionsheet.addAction(UIAlertAction(title: "Alle Einträge
        löschen", style: .destructive, handler: { action in
            print("Delete all entries in tracking history")
            for object in historyArray! {
                context.delete(object)
            }
            (UIApplication.shared.delegate as?
                AppDelegate)?.saveContext()
            (UIApplication.shared.delegate as?
                AppDelegate)?.fetchTrackingHistory()
            self.tableView.reloadData()
            self.editBarButton()
        })
    )
    present(actionsheet, animated: true)
}
```

Listing 61: TrackingHistoryViewController.swift showActionSheet()

Es ist ebenfalls möglich, Einträge zu löschen. Dafür wird folgende Methode, welche bereits vom UITableView bereitgestellt wurde, erweitert.

```
func tableView(_ tableView: UITableView, commit editingStyle:
    UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
    if editingStyle == .delete {
        let data = historyArray?[historyArray!.count - indexPath.row
            - 1]
        context.delete(data!)
        (UIApplication.shared.delegate as?
            AppDelegate)?.saveContext()
        (UIApplication.shared.delegate as?
            AppDelegate)?.fetchTrackingHistory()
        tableView.deleteRows(at: [indexPath], with: .fade)
    }
}
```

Listing 62: TrackingHistoryTableViewController.swift Element aus CoreData Stack löschen

Diese Methode ermöglicht dem Benutzer mit einem Swipe nach links einen Eintrag zu löschen.

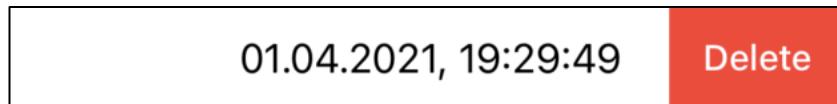


Abbildung 78: Löschen eines Eintrags im Tracking Verlauf

5.14.10 Einstellungen

Wenn auf *Logout* gedrückt wird, wird ein Action Sheet gezeigt, um den Benutzer noch einmal zu fragen, ob wirklich ausgeloggt werden soll. Dies wurde mit folgendem Code realisiert.

```
@objc func tappedLogout() {
    let actionsheet = UIAlertController(title: nil, message: nil,
        preferredStyle: .actionSheet)
    actionsheet.addAction(UIAlertAction(title: "Abbrechen", style:
        .cancel, handler: { action in
            print("Dismiss Actionsheet")
        }))
    actionsheet.addAction(UIAlertAction(title: "Logout", style:
        .destructive, handler: { action in
            self.logout()
        }))
    present(actionsheet, animated: true)
}
```

Listing 63: SettingsViewController.swift tappedLogout()

Die Funktion *logout()* löscht den Username und Passwort aus der Keychain-Valet und zeigt den Login Screen an ohne der Möglichkeit, zurückzuswipen.

```
func logout() {
    try? cloudValet.removeObject(forKey: KeychainKeys.usernameKey)
    try? cloudValet.removeObject(forKey: KeychainKeys.passwordKey)

    let loginView = UINavigationController(rootViewController:
        LoginViewController())
    loginView.modalPresentationStyle = .fullScreen
    present(loginView, animated: true)
}
```

Listing 64: SettingsViewController.swift logout()



Um die Einstellungen, welche Tracking Methoden verwendet werden soll und welche Notifications gezeigt werden dürfen, in User Defaults abzuspeichern, wurde die Struktur `SettingsKeys` verwendet, dessen Variablen als Schlüssel / Keys agieren.

```
struct SettingsKeys {
    static let TMBB = 0
    static let TMGF = 1
    static let NFBB = 2
    static let NFGF = 3
}
```

Listing 65: Globals.swift Schlüssel für User Defaults

Damit überprüft werden kann, welcher Switch gerade ausgewählt wurde, bekommt jeder Switch in den Einstellungen einen Tag. Dieser Tag entspricht den Werten aus der Struktur `SettingsKeys`.

```
sw.tag = SettingsKeys.TMBB
```

Listing 66: SettingsViewController.swift Switch Tag zuweisen

Wenn ein Switch geändert wird, wird folgende Funktion aufgerufen, welchen den Wert des Switches für den dazugehörigen Key (Switch Tag) in User Defaults speichert.

```
@objc func switchDidChange(_ sender: UISwitch) {
    defaults.set(sender.isOn, forKey: String(sender.tag))
}
```

Listing 67: SettingsViewController.swift User Defaults je nach Switch Tag setzen

Wenn der View Controller (`SettingsViewController`) geladen wird, erhält jeder Switch den Wert aus User Defaults in Abhängigkeit vom Switch Tag (Key).

```
sw.isOn = defaults.bool(forKey: String(sw.tag))
```

Listing 68: SettingsViewController.swift Zuweisung der Werte für die Switches in Einstellungen

5.14.11 Tracking Methoden

Mit folgendem Code kann der `locationManager` so konfiguriert werden, dass:

- der Benutzer jedes Mal gefragt wird, ob seine Position getrackt werden darf.
- die Position im Background getrackt werden kann.
- die Genauigkeit der Positionsbestimmung am höchsten ist.

```
func confTrackingMethods() {
    locationManager.delegate = self
    locationManager.allowsBackgroundLocationUpdates = true
    locationManager.requestAlwaysAuthorization()
    locationManager.desiredAccuracy = kCLLocationAccuracyBest
```

```

locationManager.startUpdatingLocation()

confBluetoothBeacons()
confGeofence()
bluetoothStatus()
}

```

Listing 69: HomeViewController.swift confTrackingMethods()

Die Funktion *confTrackingMethods()* wird in der Methode *viewDidLoad()* in der Klasse *HomeViewController* aufgerufen.

Die Variable *locationManager* wurde bereits in der Klasse *HomeViewController* initialisiert und wird für Bluetooth Beacons und Geofencing verwendet.

```

// Geofence & Bluetooth Beacons
var locationManager = CLLocationManager()

```

Listing 70: HomeViewController.swift locationManager initialisieren

5.14.11.1 Bluetooth Beacons

Falls der Benutzer eingeloggt ist oder sich eingeloggt hat, wird die Funktion *confBluetoothBeacons()* in der Methode *viewDidLoad()* der Klasse *HomeViewController* ausgeführt. Dies Funktion startet das Suchen nach einem Bluetooth Beacon mit der UUID: "E2C56DB5-DFFB-48D2-B060-D0F5A71096E0". Die Major und Minor Values können passend gewählt werden.

```

func confBluetoothBeacons() {
    print("Bluetooth-Beacons")
    let beacon = CLBeaconIdentityConstraint(uuid: UUID(uuidString:
        "E2C56DB5-DFFB-48D2-B060-D0F5A71096E0")!)
    locationManager.startRangingBeacons(satisfying: beacon)
}

```

Listing 71: BluetoothBeacons.swift confBluetoothBeacons()

Folgende Funktion wird von *CLLocationManagerDelegate* bereitgestellt und wird verwendet, um Bluetooth Beacons in der Nähe zu finden. Dabei wird zuerst überprüft, ob Bluetooth Beacons überhaupt verwendet werden dürfen. Dies ist in den Einstellungen der iOS-App auszuwählen. Das getrackte Ereignis wird ebenfalls im CoreData Stack für den Tracking Verlauf gespeichert. Dies folgt dem Schema wie bei NFC beschrieben.

```

func locationManager(_ manager: CLLocationManager, didRange beacons:
    [CLBeacon], satisfying beaconConstraint:
    CLBeaconIdentityConstraint) {
    if defaults.bool(forKey: String(SettingsKeys.TMBB)) {
        if beacons.first?.proximity == .some(.immediate) ||
            beacons.first?.proximity == .some(.near) {
                if startBBScan == true && timetracking.isTimeTracking ==

```



```
        false {
            startTimeTracking()
            storeStartBBScanInCoreData()
            if defaults.bool(forKey: String(SettingsKeys.NFBB)) {
                let title = "Bluetooth Beacon wurde gefunden"
                let message = "Timetracking wurde gestartet"
                self.showNotification(title: title, message:
                    message)
            }
            startBBScan = false
        } else if startBBScan == true &&
            timetracking.isTimeTracking == true {
            stopTimeTracking()
            storeStopBBScanInCoreData()
            if defaults.bool(forKey: String(SettingsKeys.NFBB)) {
                let title = "Bluetooth Beacon wurde gefunden"
                let message = "Timetracking wurde gestoppt"
                self.showNotification(title: title, message:
                    message)
            }
            startBBScan = false
        }
    } else {
        startBBScan = true
    }
}
}
```

Listing 72: BluetoothBeacons.swift Bluetooth Beacons finden

Wurde ein Bluetooth Beacon gefunden, wird entweder Timetracking gestartet oder gestoppt, je nachdem ob Timetracking bereits läuft oder nicht. Das Einsatzgebiet von diesen Bluetooth Beacons ist beispielsweise an Haupteingängen eines Bürokomplexes:

- Beim Betreten des Gebäudes würde das Timetracking starten.
- Beim Verlassen des Gebäudes würde das Timetracking stoppen.

5.14.11.2 Geofencing

Die Funktion `confGeofence()` wird ausgeführt, sobald sich der Benutzer angemeldet hat oder als eingeloggter Benutzer die App öffnet, da diese Funktion in der Methode `viewDidLoad()` der Klasse `HomeViewController` aufgerufen wird. Diese Funktion holt alle Geofences („Zäune“) vom Backend und startet das monitoring dieser Bereiche. Die Variable `locationManager` wurde bereits in der Klasse `HomeViewController` initialisiert und wird für Bluetooth Beacons und Geofencing verwendet.

```

func confGeofence() {
    trackingMethodsService.getGeofencing(completion: { statusCode in
        if statusCode == 200 {
            for region in trackingMethodsService.geofencing {
                let geofenceRegion = CLCircularRegion(center:
                    CLLocationCoordinate2D(latitude:
                        region.latitude!, longitude: region.longitude!),
                    radius: region.radius!, identifier:
                        region.identifier!)
                print(geofenceRegion)
                self.locationManager.startMonitoring(for:
                    geofenceRegion)
            }
        }
    })
}

```

Listing 73: Geofence.swift confGeofence()

Falls ein Geofencing Bereich betreten und verlassen wurde, werden die folgenden 2 Funktionen ausgeführt, welche ebenfalls von der Klasse `CLLocationManagerDelegate` zur Verfügung gestellt wurden. Dabei wird zuerst überprüft, ob Geofencing überhaupt verwendet werden darf. Dies ist in den Einstellungen der iOS-App auszuwählen. Falls Notifications in den Einstellungen der iOS-App eingeschalten wurde, wird eine Notification angezeigt. Das getrackte Ereignis wird ebenfalls im CoreData Stack für den Tracking Verlauf gespeichert. Dies folgt dem Schema wie bei NFC und Bluetooth Beacons beschrieben.

```

func locationManager(_ manager: CLLocationManager, didEnterRegion
    region: CLRegion) {
    if defaults.bool(forKey: String(SettingsKeys.TMGF)) &&
        timetracking.isTimeTracking == false {
        print("Entered")
        storeStartGFScanInCoreData()
        startTimeTracking()
        if defaults.bool(forKey: String(SettingsKeys.NFGF)) {
            let title = "Geofencing Bereich betreten"
            let message = "Timetracking wurde gestartet"
            self.showNotification(title: title, message: message)
        }
    }
}

```

Listing 74: Geofence.swift Geofencing Bereich betreten

```

func locationManager(_ manager: CLLocationManager, didExitRegion
    region: CLRegion) {
    if defaults.bool(forKey: String(SettingsKeys.TMGF)) &&

```



```
timetracking.isTimeTracking == true {
    print("Exited")
    storeStopGFScanInCoreData()
    stopTimeTracking()
    if defaults.bool(forKey: String(SettingsKeys.NFGF)) {
        let title = "Geofencing Bereich verlassen"
        let message = "Timetracking wurde gestoppt"
        self.showNotification(title: title, message: message)
    }
}
```

Listing 75: Geofence.swift Geofencing Bereich verlassen



Abbildung 79: Notification Beim Eintritt in den Geofencing Bereich

5.14.12 Extensions

Folgende Extension wird verwendet, um die Text Fields für Username und Passwort anzupassen.

```
func setLoginTextFieldStyle() {
    backgroundColor = UIColor.clear
    layer.cornerRadius = 25
    layer.borderWidth = 2
    layer.borderColor = UIColor.gray.cgColor
    borderStyle = .roundedRect
    //clearButtonMode = .whileEditing
    clipsToBounds = true

    let leftView = UIView(frame: CGRect(x: 0, y: 0, width: 10,
                                         height: frame.size.height))
    leftView.backgroundColor = backgroundColor
    self.leftView = leftView;
    leftViewMode = .always
    let rightView = UIView(frame: CGRect(x: 0, y: 0, width: -10,
                                         height: -frame.size.height))
    self.rightView = rightView
    rightViewMode = .always

    NSLayoutConstraint.activate([
        heightAnchor.constraint(equalToConstant: 50),
    ])
}
```

```
    ])
}
```

Listing 76: Extensions.swift setLoginTextFieldStyle()

Bevor Table Views geladen werden, werden Loading Spinner (`UIActivityIndicatorView`) verwendet. Dafür steht eine Extension der Klasse `UIViewController` bereit, um den Loading Spinner mit weißem oder schwarzem Hintergrund (bei Dark Mode) richtig zu konfigurieren.

```
func addLoadingScreen(spinner: UIActivityIndicatorView) {
    view.bringSubviewToFront(spinner)
    spinner.translatesAutoresizingMaskIntoConstraints = false
    spinner.backgroundColor = assets.colors.background

    view.addSubview(spinner)

    NSLayoutConstraint.activate([
        spinner.leadingAnchor.constraint(equalTo:
            view.leadingAnchor),
        spinner.trailingAnchor.constraint(equalTo:
            view.trailingAnchor),
        spinner.topAnchor.constraint(equalTo:
            view.safeAreaLayoutGuide.topAnchor),
        spinner.bottomAnchor.constraint(equalTo:
            view.safeAreaLayoutGuide.bottomAnchor),
    ])
    spinner.startAnimating()
}
```

Listing 77: Extensions.swift addLoadingScreen()

Mit folgendem Code kann ein Loading Spinner erstellt und zum View hinzugefügt und wieder gestoppt werden.

```
let spinner = UIActivityIndicatorView()
addLoadingScreen(spinner: spinner)
removeLoadingScreen(spinner: spinner)
```

Listing 78: Loading Spinner erstellen, starten und stoppen

Um Notifications am iPhone anzeigen zu können, steht eine Extension der Klasse `UIViewController` bereit, in welcher die Funktion `showNotification()` implementiert wurde.

```
func showNotification(title: String, message: String) {
    UNUserNotificationCenter.current().requestAuthorization(options:
        [.alert, .sound, .badge], completionHandler: { ( granted,
            error) in })
```



```
let content = UNMutableNotificationContent()
content.title = title
content.body = message
content.badge = 0

let request = UNNotificationRequest(identifier: "timetracking",
    content: content, trigger: nil)
UNUserNotificationCenter.current().add(request,
    withCompletionHandler: nil)
}
```

Listing 79: Extensions.swift showNotification()

Bei NFC, Bluetooth-Beacons und Geofencing wurde die Extension verwendet. Mit folgendem Code wird diese Funktion in den einzelnen ViewControllern aufgerufen.

```
showNotification(title: "Title", message: "Message")
```

Listing 80: Aufruf showNotification()

6 Entwicklung der Android App

6.1 Vorbereitung

6.1.1 Installation Android Studio

Android Studio kann kostenlos von der offiziellen Android developer website heruntergeladen werden und auf allen Geräten installiert werden, welche mindestens folgende Betriebssysteme besitzen:

- Windows 7/8/10 (64-bit)
- Mac OS X 10.10 (Yosemite) or higher, up to 10.14 (macOS Mojave)
- Linux
- Chrome OS

6.2 Theoretische Grundlagen

6.2.1 Android

In Android hat jede App eine eigene Linux ID und läuft in einer eigenen VM, der sogenannten ART Android Runtime. Früher wurde die DVM Dalvik Virtual Machine verwendet. Dadurch wird der Code jeder App isoliert ausgeführt. Die ID wird nur vom Betriebssystem verwendet und ist der App nicht bekannt. Die Files, welche mit der App in Verbindung gebracht werden, können nur über die ID der App abgerufen werden. Android Apps haben 4 Arten von App components, welche verwendet werden, um verschiedene Funktionen innerhalb der App auszuführen.

- Activities
- Services
- Broadcast receivers
- Content providers

6.2.2 Activities

Eine Activity wird verwendet, um die Verbindung zwischen App und User herzustellen. Sie stellt das GUI dar, verwaltet dieses und reagiert auf dessen Interaktion mit dem User und reagiert, wenn erwünscht oder benötigt mit Änderungen des Layouts.

6.2.2.1 Lebenszyklus

Activities haben 4 Zustände, in denen sie verweilen können.

- Running (Die Activity ist sichtbar und kann verwendet werden)
- Paused (Die Activity ist pausiert, wenn beispielsweise eine andere Activity gestartet wurde und die derzeitige Activity in den Hintergrund gelegt wurde)
- Stopped (Die Activity wird gestoppt, wenn die aktuelle Activity nicht mehr vom User sichtbar und beeinflussbar ist)
- Destroyed (Die Activity hat ihren Zweck erfüllt und wurde zerstört, um den RAM freizugeben)



Der genaue Ablauf des Lebenszyklus ist in der unteren Grafik zu sehen. In der Grafik sind die Lifecycle callbacks zu sehen, welche von der App automatisch ausgeführt werden, je nachdem in welchem Zustand die Activity sich befinden soll.

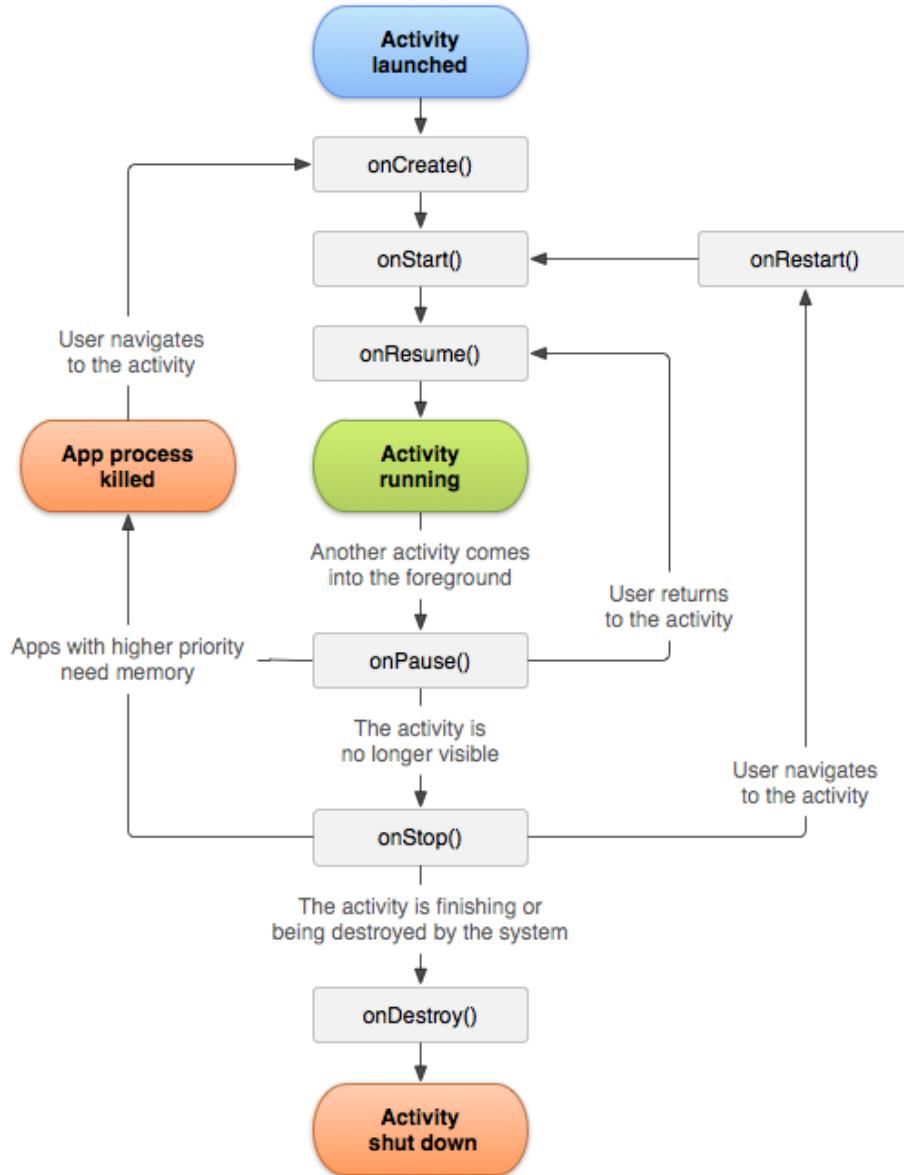


Abbildung 80: Activity Lifecycle [21]

Die anderen drei App Components haben ebenfalls einen Lifecycle, relativ ähnlich, zu dem der Activity, aber dennoch für jeden Component unterschiedlich und angepasst.

6.2.3 Services

Services wurden verwendet, um lange aufwendige Tasks im Hintergrund, vom User unbemerkt, durchzuführen. Das Service lässt die App im Hintergrund weiterarbeiten, selbst wenn diese nicht verwendet wird.

6.2.4 Broadcast Receivers

Broadcast Receivers werden verwendet, um systemweite Events abzufangen, diese in die App einzuschleusen und auf diese innerhalb der App zu reagieren.

6.2.5 Content Providers

Content Provider werden verwendet, um Daten von verschiedenen Speicherplätzen zu managen und auf diese zuzugreifen.

6.2.6 UI-Entwicklung

Um die App für den Benutzer verwendbar zu machen, wird ein User Interface, kurz UI, benötigt. In Android wird ein solches in zwei Teile aufgeteilt. Der erste Teil ist das Layout, in dem der grafische Aufbau, welcher für den User sichtbar ist, definiert ist. Im Layout wird mittels der Beschreibungssprache XML definiert, auf welche Art die UI Elemente wie Buttons, Textfelder und andere Elemente platziert sind. Der zweite Teil ist die zugehörige Kotlin-Klasse, welche die Funktion des Interfaces definiert. Die Funktionen und der Code der ausgeführt werden soll, je nachdem wie der User mit dem Layout interagiert, werden in besagter Klasse ausprogrammiert. Das Layout wird von der Kotlin-Klasse erzeugt und initialisiert.

6.2.6.1 Layout

Das User Interface kann aus mehreren Screens, Dialogen, und anderen GUI Elementen bestehen. Jeder einzelne Screen wird in einem eigenen XML File definiert.

6.2.6.2 Layoutarten

Es gibt zwei Layoutarten, die verwendet werden können, um ein Layout aufzubauen

- Linear Layouts
- Constraint Layouts

6.2.6.3 Linear Layout

Linear Layouts können entweder horizontal oder vertikal ausgerichtet sein. Die Anordnung der Elemente wird von der Reihenfolge der Elemente im Layout File bestimmt.



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="20dp">

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/message_read_tag"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_gravity="center"/>

    <ImageView
        android:id="@+id/logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:src="@drawable/ic_nfc_24"
        android:layout_gravity="center"
        android:contentDescription="@string/NFC_title" />

    <TextView
        android:id="@+id/tv_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_gravity="center"
        android:text="@string/waiting"
        android:textAppearance="?android:attr/textAppearanceMedium"/>

</LinearLayout>
```

Listing 81: Linear Layout XML File

6.2.6.4 Constraint Layout

Constraint Layouts positionieren die Elemente durch Constraints welche im Grunde Anker für die Element sind. Diese Constraints können mit Bias versehen werden, um den Platz der Elemente relativ zum Mittelpunkt zu verschieben. Margins werden verwendet, um einen gewissen Abstand zu gewissen Teilen des Layouts immer einzuhalten. Zu welchem Punkt die Constraints verankert werden sollen ist frei wählbar.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ui.nfcreader.NFCReaderFragment">

    <com.google.android.material.floatingactionbutton.ExtendedFloatingActionButton
        android:id="@+id/ScanBtn"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:clickable="true"
        android:focusable="true"
        app:fabCustomSize="100dp"
        app:maxImageSize="55dp"
        app:tint="@android:color/white"
        android:text="@string/scan"
        android:textAlignment="center"
        android:textColor="@android:color/white"
        android:textSize="35sp"
        android:textStyle="bold"
        android:textAppearance="@style/TextAppearance.AppCompat.Headline"
        app:elevation="6dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.55"
        android:contentDescription="@string/starts_time
tracking"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 82: Constraint Layout XML File

6.2.6.5 Material Design Guidelines

Die Material Design Guidelines sind eine Sammlung an Open-Source Code, welche verwendet werden, um qualitativ hochwertige UIs zu erstellen. Es gibt allgemeine Regeln und Code-Beispiele, die befolgt werden sollten, während die UI erstellt wird. Diese Richtlinien sind auf der Website <https://material.io> von Material Design nachlesbar. Weiters gibt es auf der Website <https://developer.android.com/guide> Guides und Regeln, nach denen die App entwickelt werden sollte.



6.3 Berechtigungen

Um die Privatsphäre der User zu gewährleisten, muss der User gefragt werden, ob die Standortdaten des Geräts verwendet werden dürfen. Früher wurden die benötigten Berechtigungen in den Informationen bei der Installation im Play Store angezeigt. Mittlerweile werden diese Berechtigungen, während die App läuft, abgefragt. Es kommt auf den Programmierer an, den richtigen Zeitpunkt zur Abfrage der Berechtigung zu wählen.

6.4 Programmiersprache

Android Apps können in drei verschiedenen Programmiersprachen programmiert werden.

Diese sind:

- Kotlin
- Java
- C++

Von diesen Sprachen ist C++ am wenigsten verbreitet. Java und Kotlin werden hauptsächlich zur Programmierung verwendet. Java war früher die Hauptprogrammiersprache für Android Apps. Seit kurzer Zeit ist die hauptsächlich verwendete Sprache Kotlin.

6.5 Kotlin

Kotlin wurde zur Programmierung dieser App verwendet. Kotlin ist eine Programmiersprache, welche objekt-orientierte und funktionale Programmierungselemente kombiniert verwendet. Klassen können mit Hilfe von extension functions erweitert werden ohne von anderen Klassen ableben zu müssen. Ebenfalls ist der Code sehr einfach zu lesen und lässt sich schneller schreiben. Lambda expression sind unterstützt.

6.5.1 Variablen

Variablen werden mit var angeschrieben. Jede Variable muss entweder einen Typ zugewiesen bekommen oder initialisiert werden. Bei Int Variablen zählt die Zuweisung eines Int Wertes als Initialisierung. Variablen können Inhalte aller Datentypen haben, wenn bei der Deklaration der Variable der Typ Any zugewiesen wird.

Beispiel:

```
var myVar = 1
var myInt: Int
var multitype: Any
multitype = 2
myInt = myVar - multitype
```

Listing 83: Kotlin Variablen Beispiel

6.5.2 Konstanten

Konstanten haben die Abkürzung val und verhalten sich ähnlich wie Variablen. Jedoch sobald ihnen ein Wert zugewiesen wird, kann dieser nicht mehr verändert werden. Wenn die Konstante ein Objekt ist, kann dieses sich zwar ändern, aber es kann nicht ausgetauscht werden. Weiters muss eine Konstante immer sofort initialisiert werden.

Beispiel:

```
val myVal: Long = 1  
val myString = "String"
```

Listing 84: Kotlin Konstanten Beispiel

6.6 Android Studio

Android Studio basiert auf der IntelliJ IDEA IDE und ist die Entwicklungsumgebung für die Programmierung von Android Apps. Android Studio kann von jedem gratis heruntergeladen werden.

6.6.1 Erstellen eines Projekts

Nach dem Android Studio installiert wurde, kommt folgendes Fenster auf:

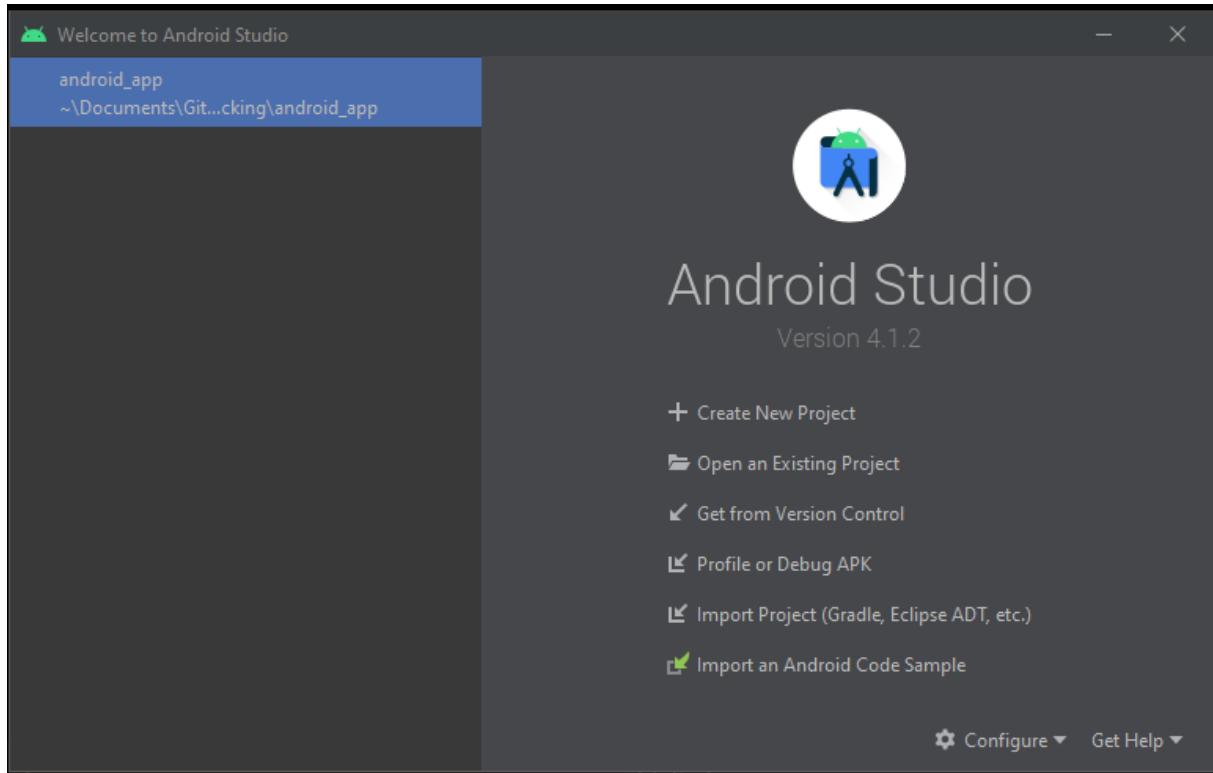


Abbildung 81: Android Studio Start Screen

Um ein neues Projekt anzulegen, muss „Create New Project“ gewählt werden. Falls bereits ein Projekt vorhanden ist, kann dieses unter „Open an Existing Project“ geöffnet werden. Bereits verwendete Projekte sind links in der Liste vermerkt und können durch einen Doppelklick geöffnet werden.

Nachdem man auf „Create New Project“ geklickt hat erscheint folgendes Fenster:

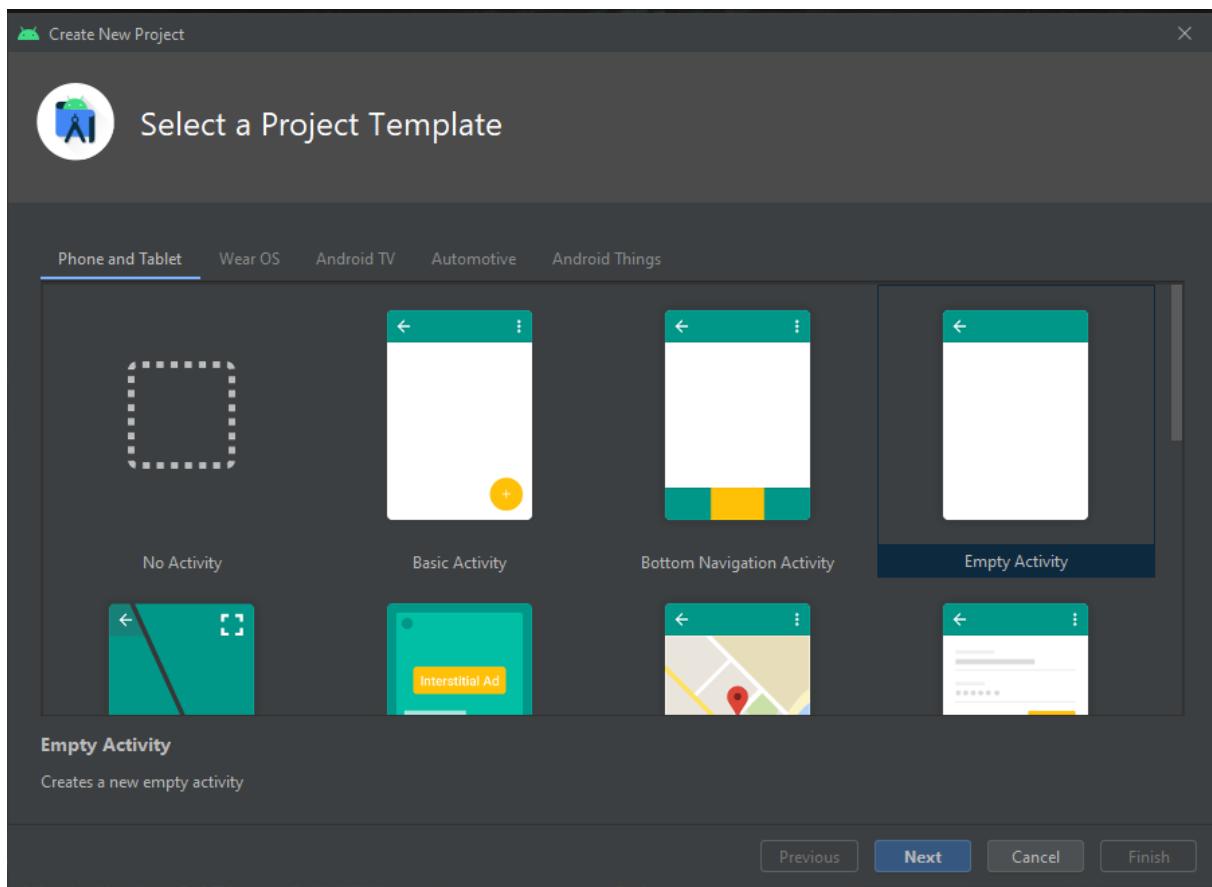


Abbildung 82: Activity Template Auswahl

Hier kann man aus verschiedenen Templates auswählen, welche Art von Activity erstellt werden soll.

Nachdem man eine für seine Wünsche passende Activity ausgewählt hat und auf „next“ geklickt hat, kommt folgendes Fenster:

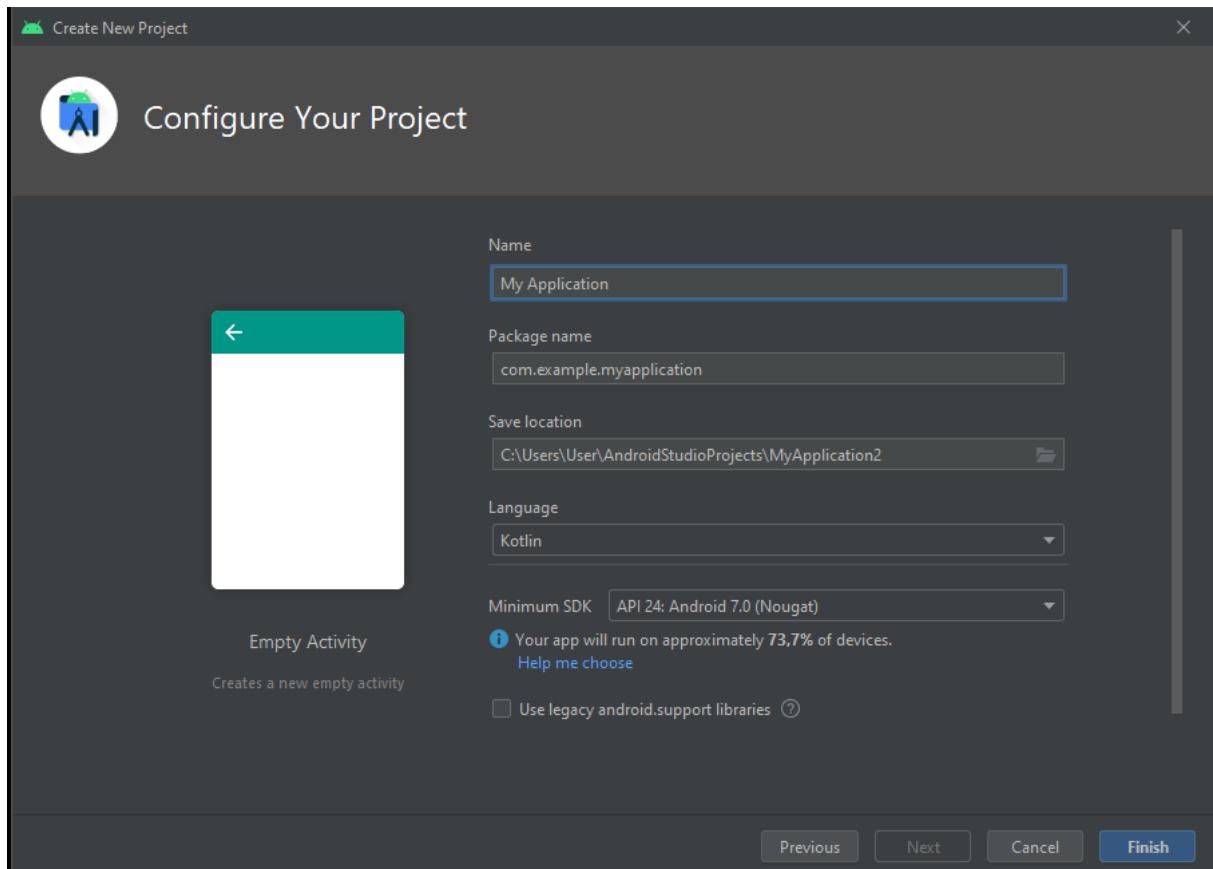


Abbildung 83: Projekt Parameter Fenster

Hier muss der Name des Projekts, sowie der Speicherort des Projekts gewählt werden. Weiters kann bei der Programmiersprache zwischen Kotlin, Java und C++ gewählt werden. Minimum SDK gibt an, ab welcher Android Version die App verwendbar sein soll. Die App wird von allen älteren Android Versionen nicht unterstützt und im Playstore nicht zum Download zur Verfügung stehen. Bei dem blauem i steht die Verteilung der Android Geräte in %, welche die App verwenden können. Das Projekt wird erstellt, wenn auf Finish geklickt wird.



Wenn auf "Help me Choose" geklickt wird, erscheint folgendes Fenster:

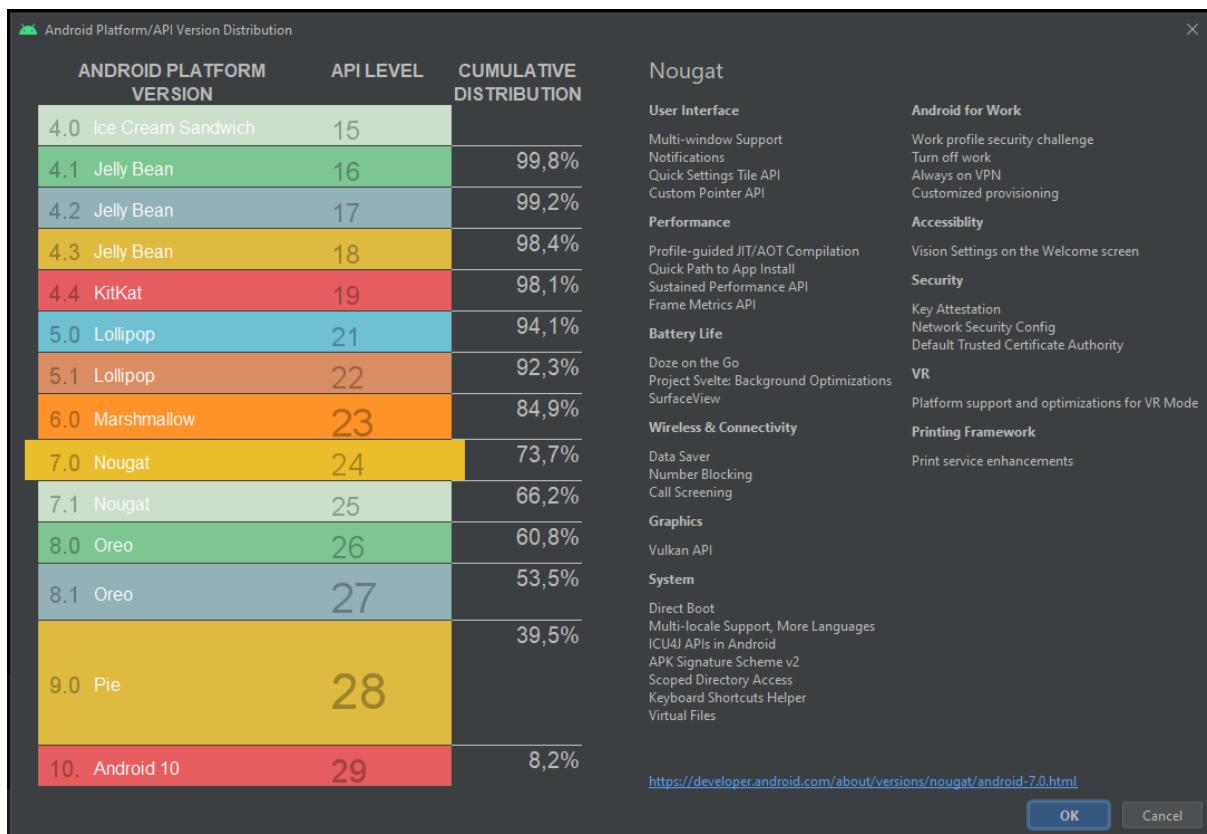


Abbildung 84: Android API-Aufteilung

In diesem Fenster sieht man die genaue Aufteilung, ab welcher Android Version, ein wie großer Anteil an Android Geräten die App unterstützen würde. Weiters werden rechts zu jeder Version Informationen angezeigt, welche Änderungen diese Version mit sich gebracht hat. Bei der Wahl der API muss darauf geachtet werden, welche Funktionen und Features die App mitbringen soll, jedoch sollte die API so niedrig wie möglich gewählt werden.

6.6.2 Navigation in Android Studio

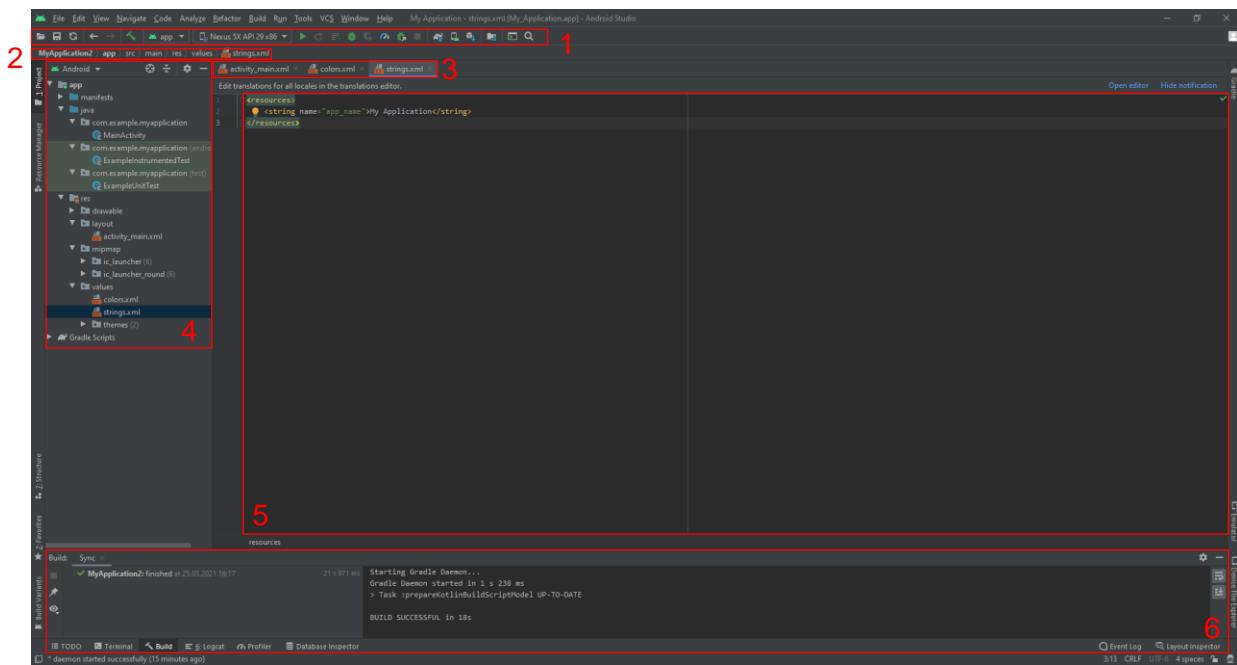


Abbildung 85: Android Studio IDE

- 1) Toolbar
- 2) Pfad des aktuellen Files
- 3) Tabs der geöffneten Files
- 4) Projektbaum
- 5) Editor
- 6) Managementleiste

6.6.3 Toolbar

Über die Toolbar sind einige wichtige Funktionen zur Programmierung und Testung der Android App vorhanden. Beispielsweise der Build-Button sowie die Run- und Debug Buttons. Weiters können folgende Fenster über die Toolbar erreicht werden:

- AVD Manager
- SDK Manager



6.6.4 SDK Manager

Der SDK Manager wird verwendet, um Android SDKs und die mit diesen verbundenen Tools, wie der Android Emulator, zu managen. SDKs sind notwendig für die Programmierung von Android Apps. Es ist notwendig anzugeben, welche SDK Version zum Builden verwendet werden soll.

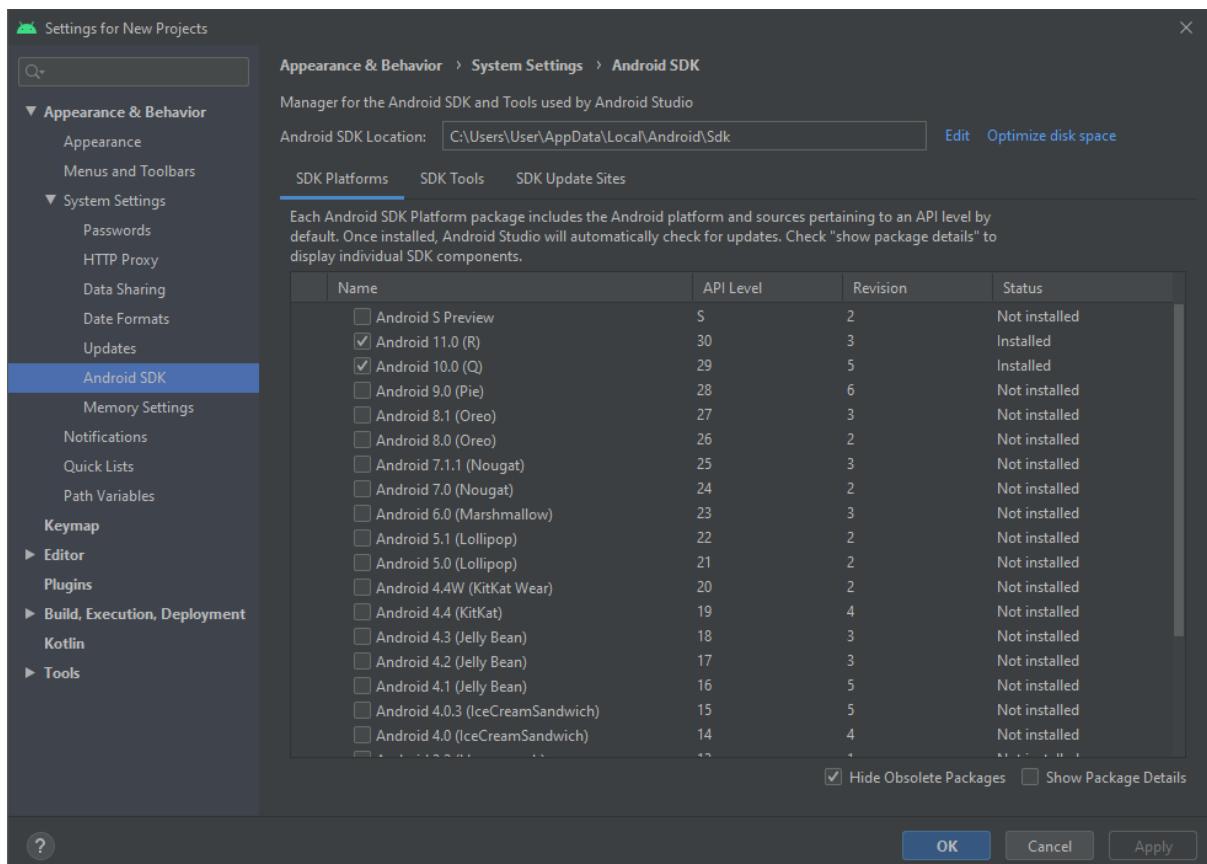


Abbildung 86: Android Studio SDK Manager

Durch das Klicken auf eine der Checkboxen wird die dazugehörige SDK oder das dazugehörige SDK Tool installiert.

6.6.5 AVD Manager

Der AVD Manager wird verwendet, um die Virtual Devices, welche vom Android Emulator simuliert werden, zu verwalten. Man kann wählen, welches Handy, mit welcher Displaygröße und Auflösung und welcher Android Version simuliert werden soll.

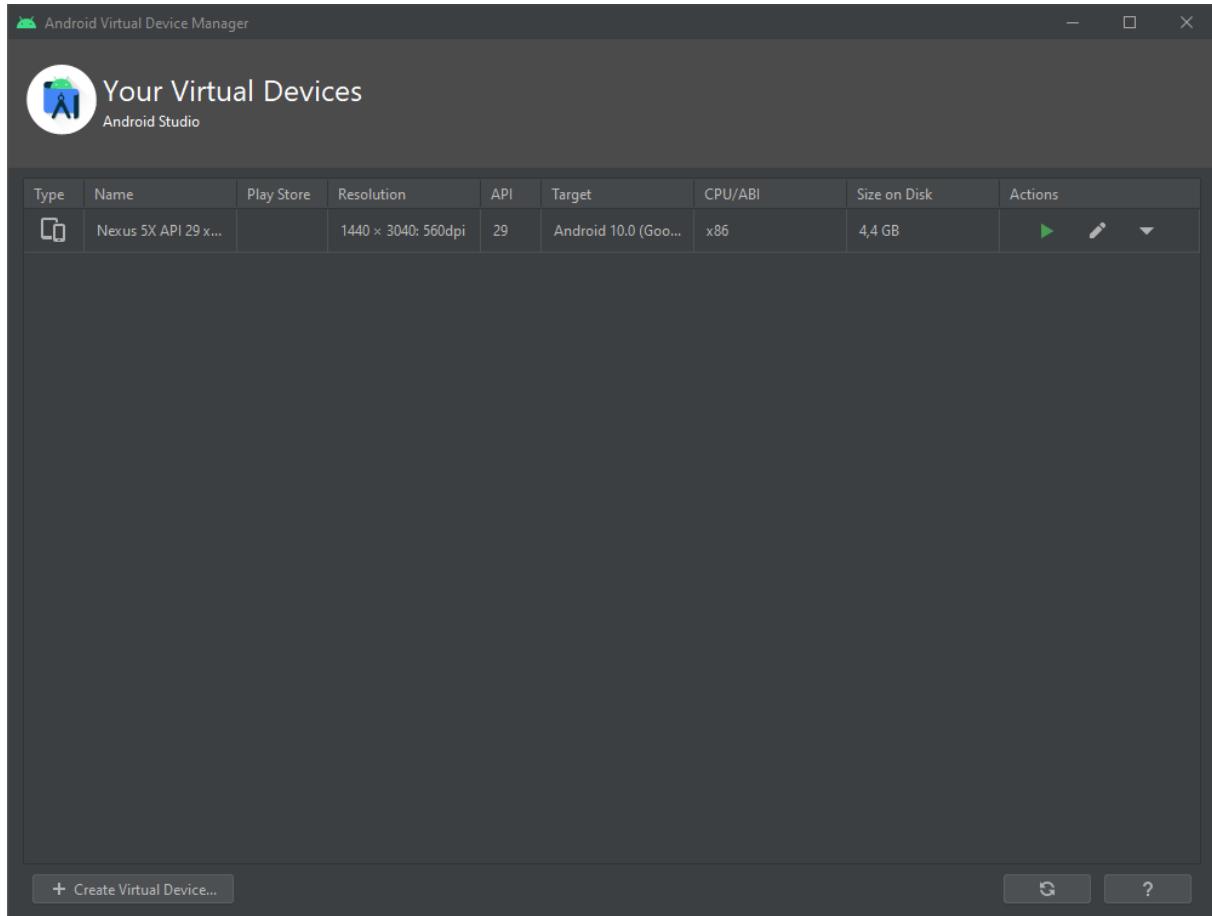


Abbildung 87: Android Studio AVD Manager

Unter „Create Virtual Device“ kann ein neues virtuelles Device erstellt werden. Man wählt aus, welches Gerät simuliert werden soll und welche Android Version das simulierte Gerät installiert haben soll. Wenn man auf den Stift klickt, kann man bestehende Virtual Devices editieren.



6.6.6 Erstellen eines UI's

Um ein UI zu erstellen muss ein XML-File erstellt werden. In Android Studio gibt es einen Layout Editor, welcher eine Live Vorschau, einen Attribute Table, einen Component Tree, eine Component Palette und einen Editor für das XML File bietet.

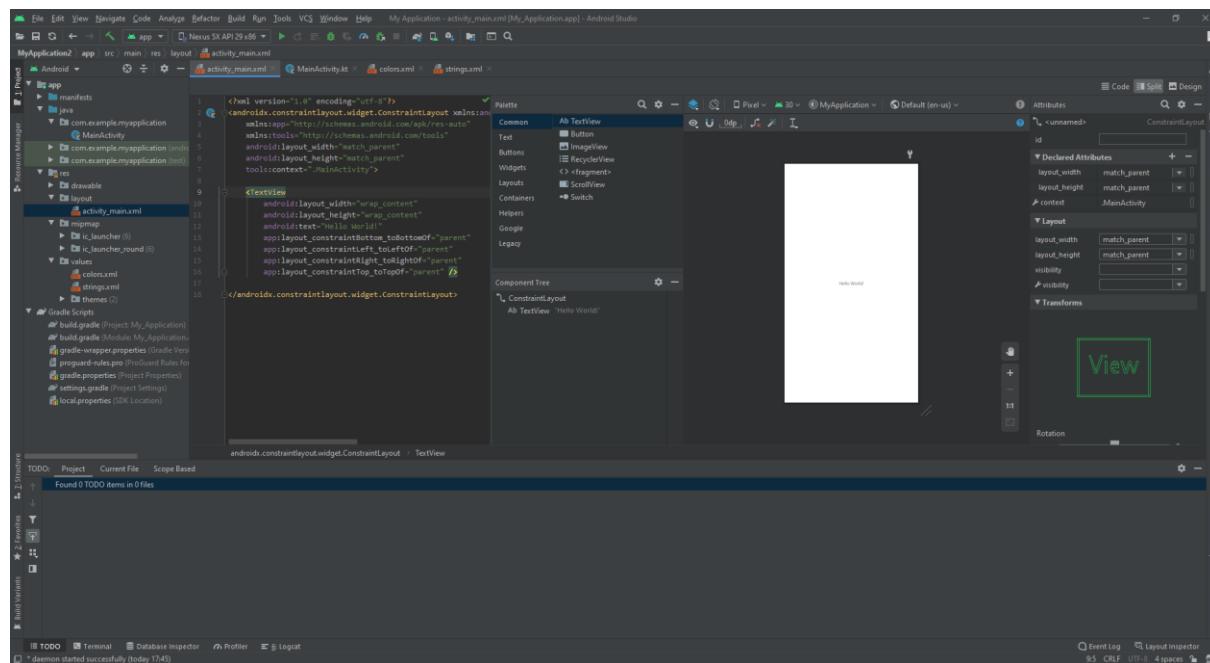


Abbildung 88: Layout Editor

Alle Änderungen bewirken eine Änderung des XML Files.

6.7 App Files

6.7.1 Android Manifest

Im Manifest File werden allgemeine Daten und Informationen zu der Android App definiert. Unter anderem zählen dazu:

- Die Activities
- Die Permissions
- Die verwendeten Phone Features wie z.B NFC
- Das Icon der App

6.7.2 Kotlin Klassen Files

Die eigentliche App wird in den Kotlin Klassen programmiert. Jedes Layout wird in einer dieser Kotlin Klassen durch Methoden mit dieser Klasse verknüpft, erstellt und angezeigt. Dadurch kann auf die einzelnen Elemente des Layouts zugegriffen und diese bearbeitet werden.

6.7.3 Resources

Im Resources Ordner sind die für die App, insbesondere für die Layouts, wichtigen Files gespeichert, welche definieren wie die Layouts von den Kotlin Klassen dargestellt werden sollen. Diese Files sind auf 9 Ordner aufgeteilt.

6.7.3.1 **anim**

Animationen, welche in XML definiert wurden, sind in diesem Ordner zu finden.

6.7.3.2 **color**

Das File tab_color.xml definiert einen color-selector, welcher in der Bottom Navigation Bar verwendet wird, um die Icons der einzelnen Screens farblich zu gestalten. Wenn das Icon nicht ausgewählt ist, ist es weiß. Wenn es ausgewählt wird, wechselt es zur Primary Color, welche die Farbe des Tailored Apps Logos ist.

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:color="@color/primaryColor"
          android:state_checked="true" />
    <item android:color="@android:color/white" />
</selector>
```

Listing 85: Menü Item Farb wähler XML File

6.7.3.3 **drawable**

Im drawable Ordner sind die XML Files für die in der App verwendeten Icons.

6.7.3.4 **layout**

Im layout Ordner sind alle Layouts, welche in der App verwendet werden, gespeichert.

6.7.3.5 **menu**

Im menu Ordner ist das XML File der Bottom Navigation Bar gespeichert. Dieses File und dieser Ordner werden nur benötigt, wenn irgendeine Art von Menü in der App verwendet wird.

Im XML File werden die Menüpunkte sowie deren Icons der Bottom Navigation Bar definiert.

6.7.3.6 **mipmap**

Im Mipmap Ordner sind die XML und PNG Files des App Logos gespeichert. Die PNG files sind in folgenden Qualitäten abgespeichert:

- hdpi (high) ~ 240dpi
- xhdpi (extra-high) ~ 320dpi
- xxhdpi (extra-extra-high) ~ 480dpi
- xxxhdpi (extra-extra-extra-high) ~ 640dpi

6.7.3.7 **navigation**

Im Navigation Ordner ist das File mobile_navigation.xml gespeichert. In diesem XML File ist definiert, wie die einzelnen Fragments miteinander verknüpft sind. Durch Actions wird definiert, wie sich welches Fragment verhält, wenn zu einem anderen Fragment navigiert wird.



```
<?xml version="1.0" encoding="utf-8"?>
<navigation
    xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/mobile_navigation"
        app:startDestination="@+id/navigation_timetracking">

    <fragment
        android:id="@+id/navigation_timetracking"
        android:name="com.example.timetracking_android.ui.
        timetracking.TimeTrackingFragment"
        android:label="Timetracking"
        tools:layout="@layout/fragment_timetracking" >
        <action
            android:id="@+id/action_navigation_timetracking_
            _to_navigation_overview"
            app:destination="@+id/navigation_overview"
            app:enterAnim="@anim/bottom_sheet_slide_in"
            app:exitAnim="@anim/bottom_sheet_slide_out" />

        <action
            android:id="@+id/action_navigation_timetracking_
            _to_navigation_settings"
            app:destination="@+id/navigation_settings"
            app:enterAnim="@anim/bottom_sheet_slide_in"
            app:exitAnim="@anim/bottom_sheet_slide_out" />

        <action
            android:id="@+id/action_navigation_timetracking_to_
            navigation_nfc_reader"
            app:destination="@+id/navigation_nfc_reader"
            app:enterAnim="@anim/bottom_sheet_slide_in"
            app:exitAnim="@anim/bottom_sheet_slide_out" />
    </fragment>

    <fragment
        android:id="@+id/navigation_overview"
        android:name="com.example.timetracking_android.ui.
        overview.OverviewFragment"
        android:label="Overview"
        tools:layout="@layout/fragment_overview" >
```

```
<action
    android:id="@+id/action_navigation_overview_to_
navigation_timetracking"
    app:destination="@+id/navigation_timetracking"
    app:enterAnim="@anim/bottom_sheet_slide_in" />

<action
    android:id="@+id/action_navigation_overview_to_
navigation_settings"
    app:destination="@+id/navigation_settings"
    app:enterAnim="@anim/bottom_sheet_slide_in"
    app:exitAnim="@anim/bottom_sheet_slide_out" />

<action
    android:id="@+id/action_navigation_overview_to_
navigation_nfc_reader"
    app:destination="@+id/navigation_nfc_reader"
    app:enterAnim="@anim/bottom_sheet_slide_in"
    app:exitAnim="@anim/bottom_sheet_slide_out" />
</fragment>

<fragment
    android:id="@+id/navigation_nfc_reader"
    android:name="com.example.timetracking_android.ui.
nfcreader.NFCReaderFragment"
    android:label="NFC-Reader"
    tools:layout="@layout/fragment_nfc_reader" >

<action
    android:id="@+id/action_navigation_nfc_reader_to_
navigation_settings"
    app:destination="@+id/navigation_settings"
    app:enterAnim="@anim/bottom_sheet_slide_in"
    app:exitAnim="@anim/bottom_sheet_slide_out" />

<action
    android:id="@+id/action_navigation_nfc_reader_to_
navigation_timetracking"
    app:destination="@+id/navigation_timetracking"
    app:enterAnim="@anim/bottom_sheet_slide_in"
    app:exitAnim="@anim/bottom_sheet_slide_out" />
```



```
<action
    android:id="@+id/action_navigation_nfc_reader_to_
navigation_overview"
    app:destination="@+id/navigation_overview"
    app:enterAnim="@anim/bottom_sheet_slide_in"
    app:exitAnim="@anim/bottom_sheet_slide_out" />
</fragment>

<fragment
    android:id="@+id/navigation_settings"
    android:name="com.example.timetracking_android.ui.
    settings.SettingsFragment"
    android:label="Settings"
    tools:layout="@xml/root_preferences" >

    <action
        android:id="@+id/action_navigation_settings_to_
navigation_overview"
        app:destination="@+id/navigation_overview"
        app:enterAnim="@anim/bottom_sheet_slide_in"
        app:exitAnim="@anim/bottom_sheet_slide_out" />

    <action
        android:id="@+id/action_navigation_settings_to_
navigation_nfc_reader"
        app:destination="@+id/navigation_nfc_reader"
        app:enterAnim="@anim/bottom_sheet_slide_in"
        app:exitAnim="@anim/bottom_sheet_slide_out" />

    <action
        android:id="@+id/action_navigation_settings_to_
navigation_timetracking"
        app:destination="@+id/navigation_timetracking"
        app:enterAnim="@anim/bottom_sheet_slide_in"
        app:exitAnim="@anim/bottom_sheet_slide_out" />
</fragment>
</navigation>
```

Listing 86: Navigation zwischen den Einzelnen Fragments der App XML File

6.7.3.8 values

Im values Ordner werden verschiedenste XML Files gespeichert, welche allgemein zur Darstellung des Layouts der App verwendet werden. Diese beinhalten z.B:

- colors – Beinhaltet die Farben, welche innerhalb der App verwendet werden.
- string – Beinhaltet jeden statischen String, der in der App verwendet wird.
- styles – Gibt an, wie gewisse Spezielle Elemente des Layouts aussehen sollen.
- themes – Ist in dark und light getrennt, gibt der App vor welche Farben wo und wie verwendet werden sollen und gibt an, welches Element wie erstellt werden soll.

6.7.3.9 xml

Im xml Ordner befindet sich das Layout des Settings Screens.

6.7.4 Assets

Der Assets Ordner wird für externe Files verwendet, welche essenziell für die Funktion der App sind, aber nicht zum Builden oder Starten der App an sich gebraucht werden. Hier befindet sich zum Beispiel das Zertifikat File des Backend Servers.

6.7.5 Gradle

Gradle ist ein Plugin, welches, ähnlich wie Maven oder Ant, das Builden von Projekten automatisiert und vereinfacht. Wichtige Informationen, wie die minimale Android SDK Version und die Android SDK Version, welche zum Builden verwendet werden sollen werden hier eingetragen. Des Weiteren wird definiert welche Kotlin Version verwendet werden soll. Die Dependencies des Projekts, welche die eingebundenen Libraries des Projekts darstellen sind ebenfalls hier zu finden. Gradle ist in der Lage Softwarestrukturen mit mehreren Modulen, welche die einzelnen Projekte darstellen, zu kompilieren und zu bauen.

6.7.5.1 build.Gradle

Das Android Projekt enthält zwei build.Gradle Files.

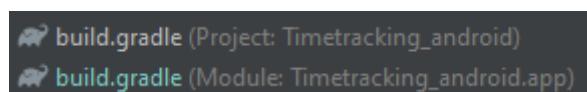


Abbildung 89: Android Gradle Files

6.7.5.2 Projekt Gradle File

Das Projekt Gradle File, enthält die für die ganze Softwarestruktur geltenden Eigenschaften und Regeln, welche beim Builden berücksichtigt werden sollen. Dazu zählen die Gradle und die Kotlin Version.

```
// Top-level build file where you can add configuration options
// common to all sub-projects/modules.

buildscript {
    ext.kotlin_version = '1.4.32'
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:4.1.3'
        classpath "org.jetbrains.kotlin:kotlin-gradle-
plugin:$kotlin_version"

        // NOTE: Do not place your application dependencies here;
        // they belong in the individual module build.gradle files
    }
}
```



```
}
```



```
allprojects {
    repositories {
        google()
        jcenter()
    }
}
task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Listing 87: Projekt Gradle File

6.7.5.3 Module Gradle File

Das Module Gradle File wird verwendet, um ein Modul (Projekt) mit spezifischen Eigenschaften und Regeln zu kompilieren und zu bilden. In unserem Fall gibt es nur ein Modul, welches das Projekt der App darstellt. In diesem File sind die restlichen, bei Punkt 6.7.4 beschriebenen, zum Kompilieren und Builden wichtigen Informationen, eingetragen.

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'

android {
    compileSdkVersion 30
    buildToolsVersion '30.0.3'

    defaultConfig {
        applicationId "com.example.timetracking_android"
        minSdkVersion 24
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner
        "androidx.test.runner.AndroidJUnitRunner"
        multiDexEnabled true
    }

    buildFeatures {
        viewBinding true
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}
```

```
        }

    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation "org.jetbrains.kotlin:kotlin-stdlib:1.4.32"
    implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation
    'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.annotation:annotation:1.2.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'androidx.navigation:navigation-fragment:2.3.4'
    implementation 'androidx.navigation:navigation-ui:2.3.4'
    implementation 'androidx.navigation:navigation-fragment-
ktx:2.3.4'
    implementation 'androidx.navigation:navigation-ui-ktx:2.3.4'
    implementation 'androidx.vectordrawable:vectordrawable:1.1.0'
    implementation 'androidx.preference:preference:1.1.1'
    implementation 'androidx.fragment:fragment:1.3.2'
    implementation 'androidx.fragment:fragment-ktx:1.3.2'
    implementation 'com.squareup.picasso:picasso:2.71828'
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation 'com.jakewharton.picasso:picasso2-okhttp3-
downloader:1.1.0'
    implementation 'com.google.guava:guava:30.1-jre'
    implementation 'androidx.multidex:multidex:2.0.1'

    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-
core:3.3.0'
}
```

Listing 88: Module Gradle File

6.7.5.4 Build Prozess

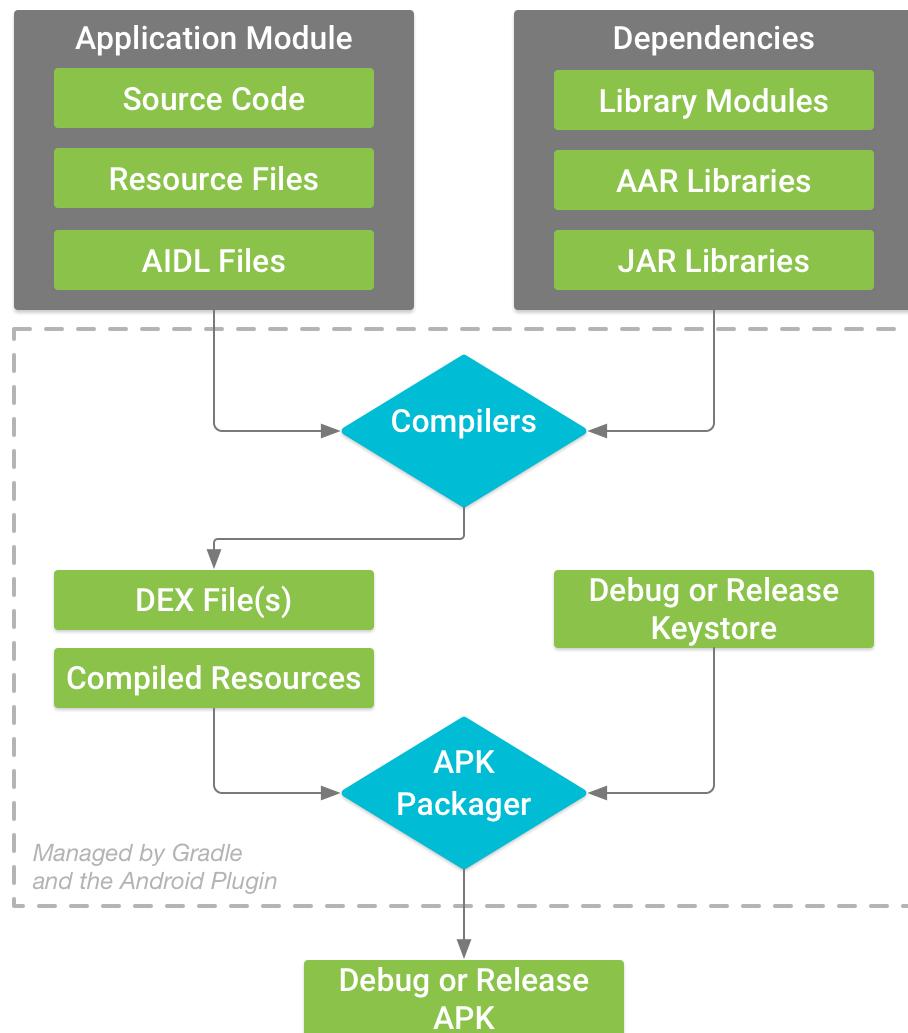


Abbildung 90: Android - Build Prozess [50]

Der Build Prozess wird in folgende Vorgänge unterteilt.

6.7.5.5 Kompilieren

Der Compiler kompiliert den Code in DEX Files (Dalvik Executeable) welche dann in der ART (siehe 6.2.1) ausgeführt werden können.

6.7.5.6 Packagen

Der APK Packager erzeugt aus den DEX Files und den kompilierten Ressourcen ein .apk File, welche zur Installation von Apps verwendet wird.

6.8 App auf Android deployen

Um die programmierte App testen beziehungsweise nutzen zu können, muss die App auf ein Android Gerät deployed werden. Hierfür gibt es zwei Möglichkeiten. Emulatoren und USB-Debugging

6.8.1 Emulator

Der in Android Studio integrierte Emulator, welcher im SDK Manager immer herunterladbar ist, nutzt die im Punkt 6.6.5 beschriebenen Android Virtual Devices, um eine virtuelle Maschine eines Android Gerätes zu simulieren.



Abbildung 92: Android Emulator



Der Emulator simuliert ein Android Gerät und unterstützt die meisten Funktionen eines normalen Android Geräts. Jedoch sind drahtlose Kommunikationstechnologien, wie NFC und Bluetooth, nicht simulierbar und nicht unterstützt. Es gibt zahlreiche andere Features, wie Pulssensoren, Accelerometer und weitere welche simuliert werden können. Diese Features sind im erweiterten Bereich der Toolbar durch einen Klick auf die drei Punkte erreichbar. Standard Features wie Powerknopf, Lautstärke, und Navigationsbuttons sind in der normalen Toolbar vorhanden.

Abbildung 91:
Android Emulator
Toolbar

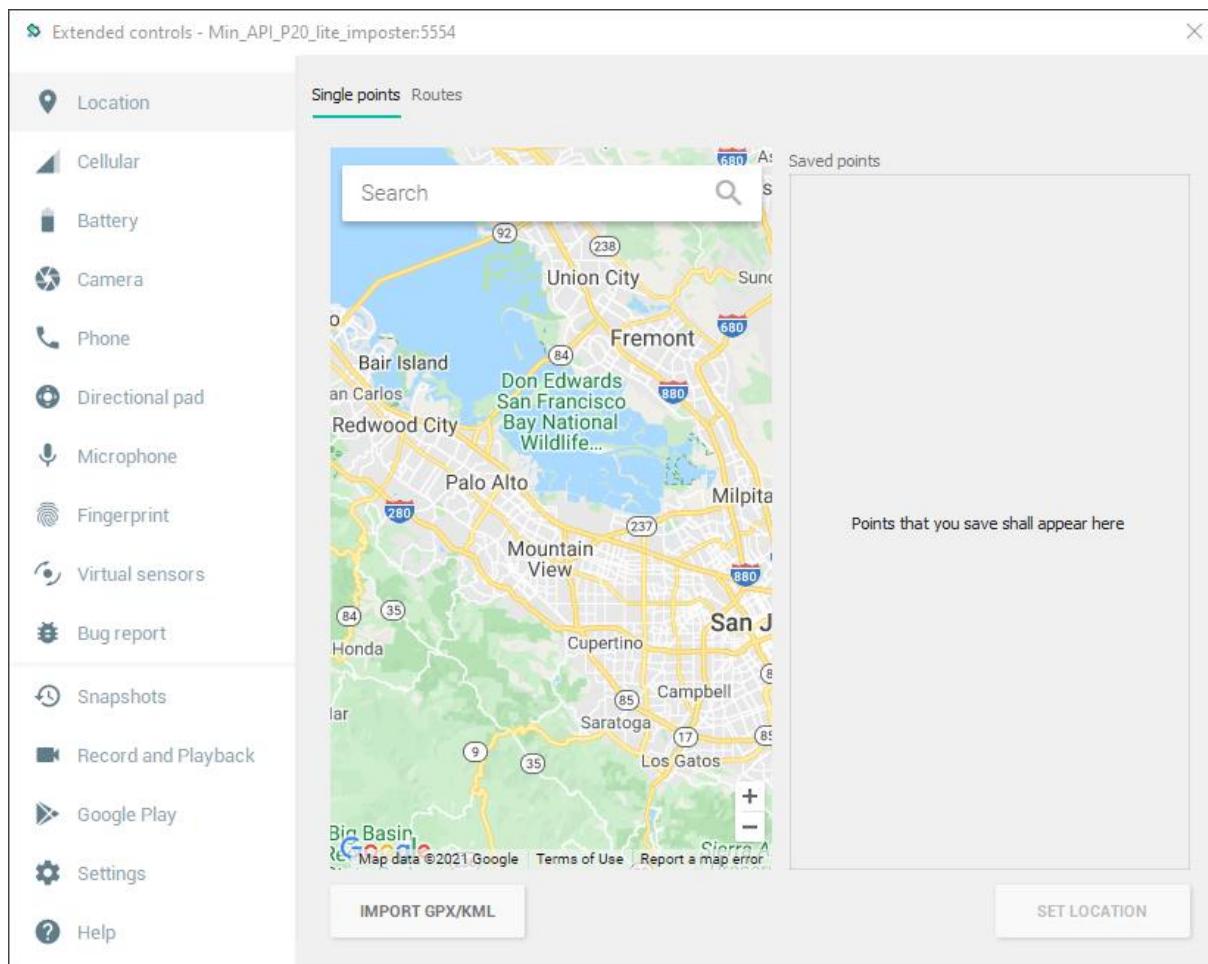
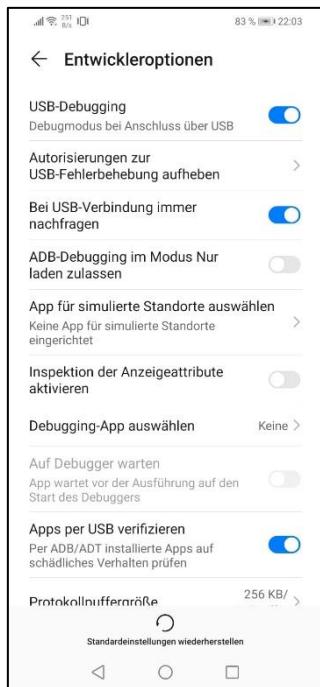


Abbildung 93: Erweiterte Toolbar des Android Emulators

Hier sind die erweiterten Features des Android Simulators zu sehen. Darunter fallen GPS-Daten-Simulation, eine virtuelle Simulation der Gerät Ausrichtung und Position im 3-dimensionalen Raum, Anruf- und SMS-Simulation und vielem mehr.

6.8.2 USB-Debugging

USB-Debugging bedeutet, die App nicht auf einem Emulator, sondern auf einem echten Android Gerät deployed. Dafür wird, wie der Name bereits andeutet eine USB-Verbindung zwischen dem Development-Gerät und dem Android-Gerät benötigt. Der Verbindungsmodus muss standardmäßig „Dateien Übertragen“ sein, kann aber in den Entwickleroptionen so konfiguriert werden, dass USB-Debugging auch im „Nur Laden“ Modus verfügbar ist. Wie bereits angedeutet, muss USB-Debugging in den Entwickleroptionen aktiviert werden, um es nutzen zu können. Dafür muss in den Geräte Optionen in den Geräte Informationen die Buildnummer 7-mal schnell hintereinander gedrückt werden. Nun kann in den Entwickleroptionen, unter „Debugging“ USB-Debugging aktiviert werden.



Die Entwickleroptionen sollten schlussendlich wie folgt konfiguriert sein. Zu beachten ist, dass es, wie bereits erwähnt, möglich ist USB-Debugging im „Nur Laden“ Modus zu verwenden, davon ist aber abzuraten, da dies eine Sicherheitslücke mit sich bringen könnte.

*Listing 89:
Entwickleroptionen USB-
Debugging*

6.9 API-Calls

API-Calls sind die Schnittstelle zum Backend. Sie werden verwendet, um mit dem Backend zu kommunizieren und Daten mit ihm auszutauschen.

6.9.1 Grundlagen

Die API-Calls basieren auf https Requests, deren Daten im JSON Format übertragen werden. In der Android App werden zwei Libraries zum Verbindungsaufbau und zum Verwalten des Requests verwendet. Dieses sind Retrofit2 und OkHttp. Um die JSON-Objekte innerhalb der App nutzen zu können, wurde zur Konvertierung zwischen API-Call Strings und App-JSON Objekten die Library GSON verwendet.

Die JSON-Objekte in der App werden mit Data Klassen erzeugt. Diese besitzen dieselben Variablen und dieselbe Struktur der JSON-Objekte, welche vom Backend verschickt werden.



Diese Klassen sind im File ApiCallJsonObjects.kt definiert. Für jeden Call gibt es eine Call Klasse und eine Response Klasse. Diese Klassen sehen wie folgt aus:

```
//object classes for reading weekly allocated Time of the User
data class WeeklyAllocTimeCalls(
    @SerializedName("idUser")
    var userId : Int? = null,
    @SerializedName("LoginUser")
    var idUser : Int? = null,
    @SerializedName("apikey")
    var apikey : String? = null
)
```

Listing 90: API-Call Call Data Klasse Beispiel

```
data class WeeklyAllocTimeResponse(
    @SerializedName("allocTime")
    var weeklyAllocTime : Int? = null,
    @SerializedName("effTIme")
    var weeklyWorkedTime: Int? = null,
    @SerializedName("neffTime")
    var weeklyTotalTime : Int? = null
)
```

Listing 91: API-Call Response Data Klasse Beispiel

Die Zeilen mit @SerializedName kommen von der GSON Library, welche diese Zeile zum Konvertieren in den übertragbaren API-Call String und beim Erhalten der Antwort, der Response, wieder zurück, verwendet.

6.9.2 API-Calls mit Retrofit2 und OkHttp

Retrofit unterstützt zwei Arten von http Requests. Synchrone und Asynchrone.

6.9.2.1 Synchrone Calls

Synchrone werden auf dem Thread durchgeführt, auf dem sie erzeugt werden. Synchrone Requests werden ähnlich wie Code abgearbeitet. Der Call wird ausgeführt und es wird auf die Antwort gewartet, welche, sobald diese ankommt, im anschließenden Code bearbeitet wird. Während man auf die Antwort wartet, ist der Thread jedoch blockiert und kann nicht verwendet werden. Dies ist in Android nicht erlaubt, da der Hauptthread der App der UI Thread ist und nicht blockiert werden darf. Des Weiteren ist es nicht erlaubt im UI Thread im Generellen Netzwerk Operationen durchzuführen. Synchrone Calls können daher nur in einem eigenen Thread durchgeführt werden.

6.9.2.2 Asynchrone Calls

Asynchrone Calls werden von Retrofit automatisch in einem externen Thread abgearbeitet und der Call funktioniert wie folgt. Es gibt einen Befehl, welcher den Call versendet und es wird ein Callback Objekt definiert, welches die Antwort des Backends darstellt. Diese Objekt besitzt zwei Callback Funktionen, onResponse() und onFailure(). Programmtechnisch ist nach dem Versenden des Calls der Call abgeschlossen und andere Codeabschnitte werden sofort durchgeführt. Die Antwort des Backends ist zu diesem Zeitpunkt noch nicht erhalten worden. Sobald die Antwort erhalten wurde, wird die Callback Funktion onResponse() im Main-Thread dem UI-Thread durchgeführt. Falls gröbere Verbindungsprobleme auftreten wird die Funktion onFailure() aufgerufen.

Ein solcher Call sieht wie folgt aus:

```
private fun sendAllocTimeAPICall(credentials: AllocTimeCalls) {
    val apiCall = (activity as MainActivity)
        .providesGetAllocatedTime()
        .getAllocatedTime(credentials)

    apiCall.enqueue(object : Callback<AllocTimeResponse> {
        override fun onResponse(call: Call<AllocTimeResponse>,
                               response: Response<AllocTimeResponse>) {
            if(response.code() == 200) {
                if(response.body()?.starttime != null &&
                   response.body()?.endTime != null) {
                    allocStartTime = response.body()?.starttime !!
                    allocStopTime = response.body()?.endTime !!
                }
                else {
                    allocStartTime = "00:00:00"
                    allocStopTime = "00:00:00"
                }
            }
        }

        override fun onFailure(call: Call<AllocTimeResponse>, t:
        Throwable) {
            Log.e(requireActivity().toString(), t.toString())
            Toast.makeText(requireActivity(), "Verbindung zum Server
fehlgeschlagen!", Toast.LENGTH_SHORT).show()
        }
    })
}
```

Listing 92: Asynchroner API-Call



Der erste Befehl im Programm erzeugt die Call Instanz, welche mit .enqueue() versendet wird, und startet die Https Verbindung zum Backend. Dafür müssen Interfaces definiert werden, bei welchen angegeben wird, welche Request Methode verwendet werden soll und was beispielsweise in den Body des Requests mitgegeben werden soll.

Diese Interfaces und die Funktion, welche die Verbindung mit dem Interface startet, sehen wie folgt aus:

```
interface GET_ALLOC_TIME {
    @POST("allocTime/")
    fun getAllocatedTime(@Body credentials: AllocTimeCalls) :
        Call<AllocTimeResponse>
}
fun providesGetAllocatedTime(): GET_ALLOC_TIME {
    val retrofit = initServerConnection()
    return retrofit.create(GET_ALLOC_TIME::class.java)
}
```

Listing 93: Interface und Verbindungsstartfunktionen

6.9.3 HTTPS und SSL Zertifikate

Android enthält eine Liste an unterstützten und vertrauenswürdigen CAs. CAs oder certification authorities sind Zertifizierungsstellen, welche im Internet öffentlich verwendbare SSL Zertifikate ausstellen. Diese Zertifikate dienen zur Ver- und Entschlüsselung von https Verbindungen. Durch das Unterschreiben des Zertifikats durch die CA wird das Zertifikat vertrauenswürdig gemacht. Die Liste der vertrauten CAs in Android unterscheiden sich jedoch innerhalb der Android Versionen und zwischen den einzelnen Android Geräte Anbietern, welche teils eine selbst erweiterte Android Version, wie zum Beispiel Huawei und EMUI, auf ihren Geräten verwenden. Dies kann manchmal dazu führen, dass das Zertifikat eines Servers durch eine nicht vertrauenswürdige CA unterschrieben wurde und die Verbindung zum Server durch Android abgelehnt wird. Um dies zu verhindern muss in der App das Zertifikat gelesen werden um die CA des Zertifikats mittels einem Trustmanager innerhalb der App vertrauenswürdig zu machen. Der Code dafür sieht wie folgt aus.

```
fun initServerConnection(): Retrofit {
    // Load CAs from an InputStream
    val cf: CertificateFactory =
        CertificateFactory.getInstance("X.509")
    val inputStream: InputStream =
        resources.assets.open("certificate/cert.crt")
    val caInput: InputStream = BufferedInputStream(inputStream)
    val ca: X509Certificate = caInput.use {
        cf.generateCertificate(it) as X509Certificate
    }
}
```

```

// Create a KeyStore containing our trusted CAs
val keyStoreType = KeyStore.getDefaultType()
val keyStore = KeyStore.getInstance(keyStoreType).apply {
    load(null, null)
    setCertificateEntry("ca", ca)
}

//Create a TrustManager that trusts the CAs inputStream our KeyStore
val tmfAlgorithm: String =
    TrustManagerFactory.getDefaultAlgorithm()
val tmf: TrustManagerFactory =
    TrustManagerFactory.getInstance(tmfAlgorithm).apply {
        init(keyStore)
    }
// Create an SSLContext that uses our TrustManager
val sslContext: SSLContext = SSLContext.getInstance("TLS").apply{
    init(null, tmf.trustManagers, null)
}

val trustManager: X509TrustManager = getX509TrustManager(tmf)

// Creating the OkHttp Client Builder
val client = OkHttpClient().newBuilder()
    .addInterceptor(GetApiCallInterceptor())
    .sslSocketFactory(sslContext.socketFactory, trustManager)
    .connectTimeout(30, TimeUnit.SECONDS)
    .readTimeout(30, TimeUnit.SECONDS)
    .writeTimeout(30, TimeUnit.SECONDS)
    .retryOnConnectionFailure(true)
    .build()

// Creating the instance of a Retrofit2 Connection Builder
return Retrofit.Builder()
    // The API server
    .baseUrl("https://wudi.serveminecraft.net:8080/")
    // Adding OkHttp Client
    .client(client)
    // Object Converter
    .addConverterFactory(GsonConverterFactory.create())
    .build()
}

```

Listing 94: SSL Zertifikat CA vertrauenswürdig machen

In diesem Code Abschnitt wird gezeigt, wie ein Zertifikat File gelesen wird, die CA des Zertifikats extrahiert und diese mittels einem Trustmanager vertrauenswürdig gemacht wird. Der Trustmanager wird in Kombination mit einem erzeugten SSL Kontext verwendet, um einen SSLSocket zu erzeugen, welcher vom OkHttp Client verwendet wird.



6.10 Tracking Methoden

Die Hauptfunktion der App ist es, Zeitdaten zu sammeln, wann gearbeitet wurde und wann Pause gemacht wurde. Dies kann in der Android App auf zwei Arten erfolgen. Manuell und mit NFC Tags. Manuell kann man mittels Buttons das Tracken der Zeit starten pausieren und stoppen. Dies kann auch durch das Lesen von NFC-Tags erfolgen.

6.10.1 NFC

Die NFC Tags werden, nachdem der Scanbutton gedrückt wurde, wird der Datensatz im Speicherabteil des NFC Tags gelesen und je nachdem, was gelesen wurde, wird das Timetracking gestartet, gestoppt oder pausiert. Die Datensätze der NFC Tags müssen daher mit Apps wie NFC Tools [53] geschrieben werden.

6.10.1.1 Foreground Dispatch

Wenn NFC aktiviert ist und ein NFC-Tag gescannt wird, wird der Inhalt dieses Tags durch die Standard NFC-App, welche im Betriebssystem integriert ist, ausgelesen und dargestellt. Damit die App, wenn sie aktiv ist, Vorrang gegenüber der Standard App beim Scannen des NFC-Tags hat, muss im onResume() Callback der Activity ein Foreground Dispatch des NFC Adapters durchgeführt werden. In der Callback Funktion onPause() wird dieser deaktiviert.

```
override fun onResume() {
    super.onResume()
    val tagDetected = IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED)
    val ndefDetected =
        IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED)
    val techDetected =
        IntentFilter(NfcAdapter.ACTION_TECH_DISCOVERED)
    val nfcIntentFilter = arrayOf(techDetected, tagDetected,
        ndefDetected)
    val pendingIntent = PendingIntent.getActivity(this, 0,
        Intent(this, javaClass)
            .addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0)
    mNfcAdapter = NfcAdapter.getDefaultAdapter(this)
    if (mNfcAdapter != null && mNfcAdapter.isEnabled) {
        mNfcAdapter?.enableForegroundDispatch(this,
            pendingIntent,
            nfcIntentFilter,
            null)
    }
}

override fun onPause() {
    super.onPause()
    if (mNfcAdapter != null) mNfcAdapter
        ?.disableForegroundDispatch(this)
}
```

Listing 95: Foreground Dispatch: MainActivity.kt

Sobald ein NFC-Tag erkannt wurde, wird ein Intent an die Activity gesendet. Intents werden verwendet, um die 4 Components von Android Apps zu starten. Intents enthalten Informationen, wofür der Intent erstellt worden ist und in manchen Fällen, wie bei dem Intent, welcher durch das Scannen des NFC-Tags erzeugt wurde, können sogar Daten an das Ziel des Intents übergeben werden. Im Callback onNewIntent() der Activity können diese dann verarbeitet werden.

```
@RequiresApi(Build.VERSION_CODES.N)
override fun onNewIntent(intent: Intent?) {
    super.onNewIntent(intent)
    if(scanningEnabled) {
        val tag: Tag? =
            intent?.getParcelableExtra(NfcAdapter.EXTRA_TAG)
        Log.d(TAG, "onNewIntent: " + intent?.action)
        if (tag != null) {
            Toast.makeText(this,
                getString(R.string.message_tag_detected),
                Toast.LENGTH_SHORT).show()
            val ndef = Ndef.get(tag)
            mNfcReadDialog =
                supportFragmentManager.findFragmentByTag
                (NFCReadDialog.TAG) as NFCReadDialog?
            if(mNfcReadDialog != null) {
                mNfcReadDialog !!!.onNfcDetected()
            }
        try {
            ndef.connect()
            val ndefMessage = ndef.ndefMessage
            val message = String(ndefMessage.records[0].payload)
            Log.d(NFCReadDialog.TAG, "readFromNFC: $message")
            ndef.close()
            when (message) {
                "Start" -> {
                    sendStartTrackingCall()
                    sharedpreferences =
                        getSharedPreferences("TIMETRACKING_DATA",
                            Context.MODE_PRIVATE)
                    val editor: SharedPreferences.Editor =
                        this.sharedPreferences.edit()
                    editor.putBoolean("WORKACTIVE", true)
                    editor.putBoolean("PAUSEACTIVE", false)
                    editor.putBoolean("ACTIVE", true)
                    editor.apply()

                    navController.navigate
                    (R.id.action_navigation_nfc_reader_
                    to_navigation_timetracking)
                }
            }
        }
    }
}
```

```

        "Stop" -> {
            sendStopTrackingCall()
            sharedPreferences =
                getSharedPreferences("TIMETRACKING_DATA",
                Context.MODE_PRIVATE)
            val editor: SharedPreferences.Editor =
                this.getSharedPreferences.edit()
            editor.putLong("WORKTIME", 0)
            editor.putLong("PAUSETIME", 0)
            editor.putLong("TIMEATPAUSE",
                SystemClock.elapsedRealtime())
            editor.putLong("REMTIME", 27720000)
            editor.putBoolean("WORKACTIVE", false)
            editor.putBoolean("PAUSEACTIVE", false)
            editor.putBoolean("ACTIVE", false)
            editor.apply()

            navController.navigate
                (R.id.action_navigation_nfc_reader_
                    to_navigation_timetracking)
        }
        "Pause" -> {
            sendPauseTrackingCall()
            sharedPreferences =
                getSharedPreferences("TIMETRACKING_DATA",
                Context.MODE_PRIVATE)
            val editor: SharedPreferences.Editor =
                this.getSharedPreferences.edit()
            editor.putBoolean("WORKACTIVE", false)
            editor.putBoolean("PAUSEACTIVE", true)
            editor.putBoolean("ACTIVE", true)
            editor.apply()

            navController.navigate
                (R.id.action_navigation_nfc_reader_
                    to_navigation_timetracking)
        }
    }
} catch (e: IOException) {
    e.printStackTrace()
} catch (e: FormatException) {
    e.printStackTrace()
}
}

scanningEnabled = false
}
}

```

Listing 96: Verarbeitung des Gescannten NFC Tags: MainActivity.kt

6.11 User Interface

Zur Darstellung des User-Interfaces wurde ein Huawei P20 Lite verwendet. Das User-Interface gibt es ebenfalls im Dark Mode. Dafür muss das Gerät in den Systemeinstellungen in den Dark Mode gewechselt werden. Das User-Interface der Android App ist auf Deutsch.

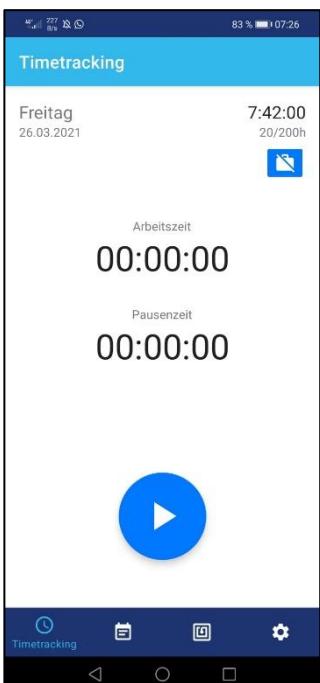
6.11.1 Login



Für die Redmine-Authentifikation muss der Benutzername (Username) und das Passwort (Password) des Users eingegeben werden. Bei erfolgreicher Anmeldung wird der User auf den Home Screen weitergeleitet.

Abbildung 94: Login Screen

6.11.2 Home Screen



Der Home Screen zeigt folgende Informationen an:

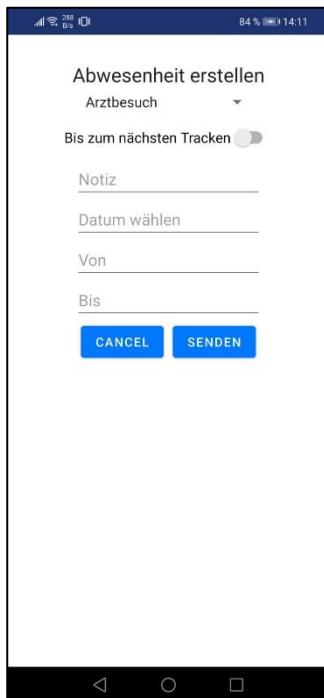
- Aktueller Tag
- Aktuelles Datum
- Sollarbeitszeit
- Gearbeitete Stunden / Gesamtsollstunden der Woche
- Arbeitszeit (getrackt)
- Pausenzeit (getrackt)
- Manueller Button zum Tracken

Es ist ebenfalls möglich, Abwesenheiten zu erstellen. Erreichbar ist dieses Feature über einen Klick auf den durchgestrichenen Koffer (rechts oben). Die Navigation innerhalb der App erfolgt über eine Bottom Navigation Bar, da dies von den Material Design Guidelines empfohlen wird.

Abbildung 95: Home Screen



6.11.3 Abwesenheit erstellen Dialog



Hier können neue Abwesenheiten hinzugefügt werden. Folgende Informationen müssen angegeben werden:

- Bis zur nächsten Zeitaufzeichnung (Abwesenheit gilt so lange, bis neu getrackt wird)
- Der Abwesenheitsgrund
- Datum der Abwesenheit
- Beginn der Abwesenheit (Zeit, nur wenn nicht ganztags)
- Ende der Abwesenheit (Zeit, nur wenn nicht ganztags)
- Notizen (Relevante Zusatzinformationen)

Abbildung 96: Abwesenheiten Dialog

6.11.4 NFC Reader

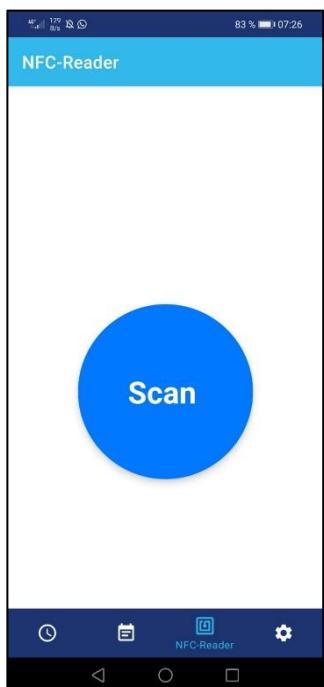


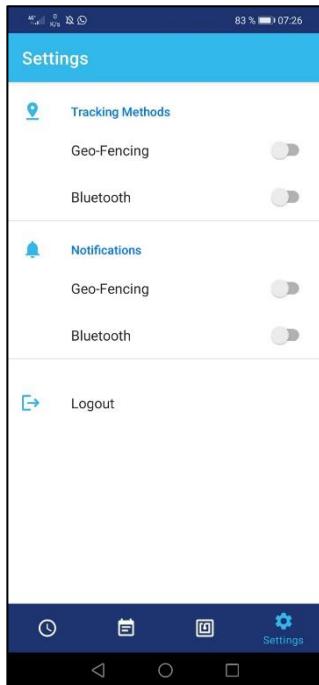
Abbildung 98: NFC Screen



Abbildung 97: NFC Lese Dialog

Nach einem Druck auf den Scan Button wird ein Dialog angezeigt, welcher den Scanstatus des NFC-Tags zeigt. Nach erfolgreichem Scan wird man auf den Homescreen umgeleitet.

6.11.5 Einstellungen



In den Einstellungen kann man sich abmelden und sowohl die Tracking Methoden als auch die Notifications ein- oder ausschalten. Da diese Funktionen aus Zeitgründen nicht implementiert werden konnten, sind diese Einstellungen derzeit nur ein Platzhalter. Standardmäßig sind alle Switches auf aus.

Listing 97: Einstellungen Screen

6.12 App Funktionen

Die App ist aus zwei Activities aufgebaut. Der Login Activity und der Main Activity.

6.12.1 Login Activity

Die Login Activity besteht aus einem Screen, dem Login Screen und zum Authentifizieren des Users wird Redmine verwendet. Die Anmelde Daten werden per http Call an Redmine geschickt und die Userdaten, welche vom Backend und der App verwendet werden, sind die Antwort auf eine erfolgreiche Authentifizierung. Der Screen besteht aus zwei Textfeldern, welche beide ausgefüllt sein müssen, um den Login Button klickbar zu machen. Sobald der Login Button gedrückt wurde, wird ein Spinner angezeigt und sobald die Authentifizierung abgeschlossen ist, wird man zur Main Activity weitergeleitet.



```
package com.example.timetracking_android.ui.login

import android.app.Activity
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProviders
import android.os.Bundle
import androidx.annotation.StringRes
import androidx.appcompat.app.AppCompatActivity
import android.text.Editable
import android.text.TextWatcher
import android.view.View
import android.view.inputmethod.EditorInfo
import android.widget.*
import com.example.timetracking_android.MainActivity
import com.example.timetracking_android.R

class Login : AppCompatActivity() {

    private lateinit var loginViewModel: LoginViewModel
    private lateinit var loginStatus:SharedPreferences

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_login)

        val username = findViewById<EditText>(R.id.username)
        val password = findViewById<EditText>(R.id.password)
        val login = findViewById<Button>(R.id.login)
        val loading = findViewById<ProgressBar>(R.id.loading)
        val loadingTint = findViewById<ImageView>(R.id.loading_tint)
        val remember = findViewById<CheckBox>(R.id.Remember)
        loginStatus = getSharedPreferences("USERDATA",
            Context.MODE_PRIVATE)

        loginViewModel = ViewModelProviders.of(this,
            LoginViewModelFactory())
            .get(LoginViewModel::class.java)
```

```
if(loginStatus.getBoolean("REMEMBERED", false)) {
    loading.visibility = View.VISIBLE
    loadingTint.visibility = View.VISIBLE
    remember.isChecked = true
    remember.isEnabled = false
    val username: String =
        loginStatus.getString("USERNAME", "") !!
    val passwrd: String =
        loginStatus.getString("PASSWORD", "") !!
    loginViewModel.login(username, passwrd)
}

loginViewModel.loginFormState.observe(this@Login, Observer {
    val loginState = it ?: return@Observer

    // disable login button unless both username / password
    // is valid
    login.isEnabled = loginState.isDataValid

    if (loginState.usernameError != null) {
        username.error = getString(loginState.usernameError)
    }
    if (loginState.passwordError != null) {
        password.error = getString(loginState.passwordError)
    }
})

loginViewModel.loginResult.observe(this@Login, Observer {
    val loginResult = it ?: return@Observer

    loading.visibility = View.GONE
    loadingTint.visibility = View.GONE
    if (loginResult.error != null) {
        showLoginFailed(loginResult.error)
        username.text.clear()
        password.text.clear()
    }
    if (loginResult.success != null) {
        updateUiWithUser(loginResult.success)
        setResult(Activity.RESULT_OK)
        //Complete and destroy login activity once
        //successful
        finish()
    }
})
}
```



```
username.afterTextChanged {
    loginViewModel.loginDataChanged(
        username.text.toString(),
        password.text.toString()
    )
}

password.apply {
    afterTextChanged {
        loginViewModel.loginDataChanged(
            username.text.toString(),
            password.text.toString()
        )
    }
}

setOnEditorActionListener { _, actionId, _ ->
    when (actionId) {
        EditorInfo.IME_ACTION_DONE ->
            loginViewModel.login(
                username.text.toString(),
                password.text.toString()
            )
    }
}
false
}

login.setOnClickListener {
    loading.visibility = View.VISIBLE
    loadingTint.visibility = View.VISIBLE
    if (remember.isChecked) {
        val editor: SharedPreferences.Editor =
            loginStatus.edit()
        editor.putBoolean("REMEMBERED", true)
        editor.putString("USERNAME",
            username.text.toString())
        editor.putString("PASSWORD",
            password.text.toString())
        editor.apply()
    }
    loginViewModel.login(username.text.toString(),
        password.text.toString())
}
}

private fun updateUiWithUser(model: LoggedInUserView) {
    switchToMain()
```

```
}

private fun switchToMain()
{
    val intent = Intent(this, MainActivity::class.java)
    // start your next activity
    startActivity(intent)
}

private fun showLoginFailed(@StringRes errorString: Int) {
    Toast.makeText(applicationContext, errorString,
        Toast.LENGTH_SHORT).show()
}

/**
 * Extension function to simplify setting an afterTextChanged action
to EditText components.
 */
fun EditText.afterTextChanged(afterTextChanged: (String) -> Unit) {
    this.addTextChangedListener(object : TextWatcher {
        override fun afterTextChanged(editable: Editable?) {
            afterTextChanged.invoke(editable.toString())
        }

        override fun beforeTextChanged(s: CharSequence, start: Int,
            count: Int, after: Int) {}

        override fun onTextChanged(s: CharSequence, start: Int,
            before: Int, count: Int) {}
    })
}
```

Listing 98: Login Activity Login.kt

Dieser Code zeigt, wie die Login Activity programmiert wurde. Um Daten zu speichern, welche über die ganze App verwendet werden, werden Shared Preferences verwendet.



6.12.2 Main Activity

Die Main Activity ist das Herzstück der App. Sie besteht aus mehreren Fragments, welche durch eine Bottom Navigationbar erreichbar sind. Es gibt folgende Elemente in der Bottom Navigationbar:

- Timetracking
- Überblick
- NFC-Reader
- Einstellungen

Durch den Klick auf das Icon des Elements, wird man zu dem Element zugehörigen Fragment navigiert.

Die Elemente werden wie folgt definiert:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/navigation_timetracking"
        android:icon="@drawable/ic_timetracking_24"
        android:title="@string/timetracking"/>

    <item
        android:id="@+id/navigation_overview"
        android:icon="@drawable/ic_overview_24"
        android:title="@string/overview" />

    <item
        android:id="@+id/navigation_nfc_reader"
        android:icon="@drawable/ic_nfc_24"
        android:title="@string/nfc_reader" />

    <item
        android:id="@+id/navigation_settings"
        android:icon="@drawable/ic_settings_24"
        android:title="@string/settings" />
</menu>
```

Listing 99: Bottom Navigationbar Elements bottom_nav_menu.xml

Die Bottom Navigationbar wird wie folgt in der Main Activity erzeugt:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val navView: BottomNavigationView = findViewById(R.id.nav_view)

    navController = findNavController(R.id.nav_host_fragment)
    // Passing each menu ID as a set of IDs because each
    // menu should be considered as top level destinations.
    val appBarConfiguration = AppBarConfiguration(setOf(
        R.id.navigation_timetracking,
        R.id.navigation_overview,
        R.id.navigation_nfc_reader,
        R.id.navigation_settings))
    setupActionBarWithNavController(navController,
    appBarConfiguration)
    navView.setupWithNavController(navController)
}
```

Listing 100: Bottom Navigationbar erzeugen MainActivity.kt

Das Ergebnis sieht wie folgt aus:



Abbildung 99: Android Bottom Navigationbar

6.12.3 Timetracking

Der Timetracking Screen besteht aus drei Chronometern, welche einen Countdown für die Sollarbeitszeit des Tages, einen Counter der gearbeiteten Zeit und einen Counter der Pausenzeiten beinhalten. Links oben wird das Aktuelle Datum und der Wochentag angezeigt. Rechts oben sind der Countdown Chronometer, eine Übersicht der wöchentlich gearbeiteten und benötigten Arbeitsstunden und ein Button welcher den User zum Abwesenheit erstellen Dialog führt. Unten ist anfangs ein Startknopf positioniert. Sobald dieser gedrückt wird, startet der Arbeitszeitchronometer und der Button wird mit einem Pause- und einem Stopp Button ersetzt. Der Pause Button, pausiert den Arbeitszeitchronometer und startet den Pausenzeitchronometer. Außerdem wird er durch einen Resume Button ersetzt, welcher den Arbeitszeitchronometer startet, den Pausenzeitchronometer stoppt und wieder durch den Pause Button ersetzt wird. Der Stopp Button beendet das Tracken und setzt alle Chronometer zurück auf den Ursprungswert.



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ui.timetracking.TimeTrackingFragment">

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/StartBtn"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:clickable="true"
        android:focusable="true"
        android:src="@drawable/ic_baseline_play_arrow_24"
        app:fabCustomSize="100dp"
        app:maxImageSize="55dp"
        app:tint="@android:color/white"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.80"
        android:contentDescription="@string/starts_timetracking" />

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/PauseBtn"
        android:layout_width="70dp"
        android:layout_height="70dp"
        android:clickable="true"
        android:focusable="true"
        android:visibility="gone"
        android:src="@drawable/ic_baseline_pause_24"
        app:fabCustomSize="70dp"
        app:maxImageSize="35dp"
        app:tint="@android:color/white"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.3"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.785"
        android:contentDescription="@string/pauses_timetracking" />
```

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/ResumeBtn"
    android:layout_width="70dp"
    android:layout_height="70dp"
    android:clickable="true"
    android:focusable="true"
    android:visibility="gone"
    android:src="@drawable/ic_baseline_play_arrow_24"
    app:fabCustomSize="70dp"
    app:maxImageSize="35dp"
    app:tint="@android:color/white"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.3"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.785"
    android:contentDescription="@string/resumes_timetracking" />

<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/StopBtn"
    android:layout_width="70dp"
    android:layout_height="70dp"
    android:clickable="true"
    android:focusable="true"
    android:visibility="gone"
    android:src="@drawable/ic_baseline_stop_24"
    app:fabCustomSize="70dp"
    app:maxImageSize="35dp"
    app:tint="@android:color/white"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.7"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.785"
    android:contentDescription="@string/stops_timetracking" />
```



```
<Chronometer
    android:id="@+id/PauseTime"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:format="00:%s"
    android:textAlignment="center"
    android:textAppearance="@style/TextAppearance.AppCompat
        .Large"
    android:textSize="40sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.451" />

<Chronometer
    android:id="@+id/WorkTime"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:format="00:%s"
    android:textAlignment="center"
    android:textAppearance="@style/TextAppearance.AppCompat
        .Large"
    android:textSize="40sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.283" />

<Chronometer
    android:id="@+id/RemTime"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:textAlignment="center"
    android:textAppearance="@style/TextAppearance.AppCompat
        .Large"
    android:textSize="20sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```
<TextView
    android:id="@+id/Date"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintStart_toStartOf="@id/weekDay"
    app:layout_constraintTop_toBottomOf="@id/weekDay"
    tools:ignore="HardcodedText" />

<TextView
    android:id="@+id/WorkText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/worktime"
    app:layout_constraintBottom_toTopOf="@id/WorkTime"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="1" />

<TextView
    android:id="@+id/PauseText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pausetime"
    app:layout_constraintBottom_toTopOf="@id/PauseTime"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintVertical_bias="1" />

<TextView
    android:id="@+id/weeklyHours"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="20/200h"
    app:layout_constraintEnd_toEndOf="@id/RemTime"
    app:layout_constraintTop_toBottomOf="@id/RemTime"
    tools:ignore="HardcodedText" />
```



```
<TextView  
    android:id="@+id/weekDay"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="16dp"  
    android:layout_marginTop="16dp"  
    android:textSize="20sp"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent"  
    tools:ignore="HardcodedText"/>  
  
<ImageButton  
    android:id="@+id/AbsencesBtn"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="4dp"  
    android:layout_marginEnd="16dp"  
    android:src="@drawable/ic_baseline_work_off_24"  
    android:backgroundTint="@color/Accent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/weeklyHours"  
    android:contentDescription="@string/addAbsences"  
    tools:targetApi="lollipop" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 101: Android Timetracking Fragment Layout fragment_timetracking.xml

Hier ist zu sehen, wie das Layout des Timetracking Fragments aufgebaut ist.

Hier sind die Variablen definiert, welche für die Logik des Fragments benötigt werden.

```
lateinit var root: View  
  
private lateinit var sharedPreferences: SharedPreferences  
private var mAbsenceDialog: AbsencesDialog? = null  
  
private var curWorkTime:Long = 0  
private var curPauseTime:Long = 0  
private var finWorkTime:Long = 0  
private var finPauseTime:Long = 0  
private var curRemTime:Long = 0  
private var workActive:Boolean = false  
private var pauseActive:Boolean = false  
private var Active:Boolean = false  
private var mLastClickTime: Long = 0  
private var timeAtPause:Long = 0  
@RequiresApi(Build.VERSION_CODES.N)  
val c: Calendar = Calendar.getInstance()
```

Listing 102: Variablen definition TimeTrackingFragment.k

Die Buttonclickhandler sind alle ähnlich aufgebaut und sind wir folgt programmiert:

```
pauseBtn.setOnClickListener {
    if (SystemClock.elapsedRealtime() - mLastClickTime < 1000) {
        return@setOnClickListener
    }

    (activity as MainActivity).sendPauseTrackingCall()

    mLastClickTime = SystemClock.elapsedRealtime()

    curWorkTime = workTime.base - SystemClock.elapsedRealtime()
    workTime.stop()
    workActive = false

    curRemTime = remTime.base - SystemClock.elapsedRealtime()
    remTime.stop()

    pauseTime.base = SystemClock.elapsedRealtime() + curPauseTime
    pauseTime.start()
    pauseActive = true

    startBtn.hide()
    resumeBtn.show()
    pauseBtn.hide()
    stopBtn.show()
}
```

Listing 103: Buttonhandler TimeTrackingFragment.kt

Der Absence Button, welcher den Abwesenheit erstellen Dialog öffnet, wird wie folgt programmiert:

```
absenceBtn.setOnClickListener {
    mAbsenceDialog =
        childFragmentManager.findFragmentByTag(AbsencesDialog.TAG)
        as AbsencesDialog?
    if (mAbsenceDialog == null) {
        mAbsenceDialog = AbsencesDialog.newInstance()
    }
    mAbsenceDialog !!.show(childFragmentManager, AbsencesDialog.TAG)
}
```

Listing 104: Absence Button TimeTrackingFragment.kt



Sobald das Fragment in den Hintergrund geschoben wird, werden die Trackingdaten mit Shared Preferences gespeichert.

```
override fun onDestroyView() {
    super.onDestroy()

    val workTime: Chronometer = root.findViewById(R.id.WorkTime)
    val pauseTime: Chronometer = root.findViewById(R.id.PauseTime)
    val remTime: Chronometer = root.findViewById(R.id.RemTime)

    val editor: SharedPreferences.Editor =
        this.sharedPreferences.edit()
    if(workActive) {
        curWorkTime = workTime.base - SystemClock.elapsedRealtime()
        curRemTime = remTime.base - SystemClock.elapsedRealtime()
    }
    else if(pauseActive)
    {
        curPauseTime = pauseTime.base -
            SystemClock.elapsedRealtime()
    }

    editor.putLong("WORKTIME", curWorkTime)
    editor.putLong("PAUSETIME", curPauseTime)
    editor.putLong("TIMEATPAUSE", SystemClock.elapsedRealtime())
    editor.putLong("REMTIME", curRemTime)
    editor.putBoolean("WORKACTIVE", workActive)
    editor.putBoolean("PAUSEACTIVE", pauseActive)
    editor.putBoolean("ACTIVE", Active)
    editor.apply()
}
```

Listing 105: Trackingdatenspeicherung TimeTrackingFragment.kt

Folgender Code wird ausgeführt, sobald das Fragment wieder in den Vordergrund geholt wird und die Trackingdaten gelesen und weiterverwendet werden.

```
curWorkTime = sharedPreferences.getLong("WORKTIME", 0)
curPauseTime = sharedPreferences.getLong("PAUSETIME", 0)
curRemTime = sharedPreferences.getLong("REMTIME", 27720000)
timeAtPause = sharedPreferences.getLong("TIMEATPAUSE", 0)
workActive = sharedPreferences.getBoolean("WORKACTIVE", false)
pauseActive = sharedPreferences.getBoolean("PAUSEACTIVE", false)
Active = sharedPreferences.getBoolean("NOTACTIVE", false)

when {
    workActive -> {
        curWorkTime += (timeAtPause - SystemClock.elapsedRealtime())
        curRemTime += (timeAtPause - SystemClock.elapsedRealtime())

        workTime.base = SystemClock.elapsedRealtime() + curWorkTime
        pauseTime.base = SystemClock.elapsedRealtime() +
            curPauseTime
        remTime.base = SystemClock.elapsedRealtime() + curRemTime

        workTime.start()
        remTime.start()

        startBtn.hide()
        resumeBtn.hide()
        pauseBtn.show()
        stopBtn.show()
    }
    pauseActive -> {
        curPauseTime += (timeAtPause -
            SystemClock.elapsedRealtime())

        workTime.base = SystemClock.elapsedRealtime() + curWorkTime
        pauseTime.base = SystemClock.elapsedRealtime() +
            curPauseTime
        remTime.base = SystemClock.elapsedRealtime() + curRemTime

        pauseTime.start()

        startBtn.hide()
        resumeBtn.show()
        pauseBtn.hide()
        stopBtn.show()
    }
}
```



```
!Active -> {  
    sendAllocTimeRequest(4321, 4321, "4321", apiCallDate)  
  
    workTime.base = SystemClock.elapsedRealtime()  
    pauseTime.base = SystemClock.elapsedRealtime()  
    remTime.base = SystemClock.elapsedRealtime() + curRemTime  
  
    startBtn.show()  
    resumeBtn.hide()  
    pauseBtn.hide()  
    stopBtn.hide()  
}  
}
```

6.12.4 Abwesenheiten

Der Abwesenheit erstellen Dialog besteht aus einem Drop-down Menü, aus dem man wählen kann, welche Abwesenheit erstellt werden soll und 4 Textfelder für,

- Die Notiz
- Die Startzeit
- Die Stoppzeit
- Das Datum

Und einem Switch der die Abwesenheit auf unbestimmte Zeit, bis zum nächsten Timetracken, erstellt. Der Cancel Button schließt den Dialog ohne weitere Aktionen. Je nach ausgewählter Abwesenheit und ob der Switch betätigt wurde, müssen das Datum und die Start- und Stopzeiten nicht angegeben werden. Erst so bald alle sichtbaren Felder beim Druck des Senden Buttons ausgefüllt sind wird die Abwesenheit erstellt. Die Zeiten und das Datum werden mit sogenannten Picker Dialogen ausgewählt.

Ein solcher Picker Dialog wird wie folgt programmiert:

```
startTime.setOnTouchListener { _: View, motionEvent: MotionEvent ->
    // time picker dialog
    val timePicker = TimePickerDialog(requireActivity(),
        { _, sHour, sMinute ->
            chosenStartTime = StringBuilder()
                .append(sHour)
                .append(":")
                .append(sMinute)
                .toString()
            val tempStartTime: Date =
                SimpleDateFormat("H:m").parse(chosenStartTime)
            chosenStartTime =
                SimpleDateFormat("HH:mm").format(tempStartTime)
            startTime.setText(chosenStartTime)
        }, hour, minutes, true)

    if (motionEvent.action == ACTION_DOWN) {
        timePicker.show()
        return@setOnTouchListener true
    } else {
        return@setOnTouchListener false
    }
}
```

Listing 106: Picker Dialog AbsenceDialog.kt



Die Logik hinter dem Sende Button wird wie folgt programmiert:

```
sendAbsenceBtn.setOnClickListener {
    if(untilTracking.isChecked) {
        sendAbsenceTimestamp(4321, 4321, "4321",
            note.text.toString(), untilTracking.isChecked,
            selectedAbsence)
        dismiss()
    }
    else if( !untilTracking.isChecked && (selectedAbsence ==
        "Urlaub" || selectedAbsence == "Dienstverhinderung") )
    {
        if(::formattedChosenDate.isInitialized) {
            sendAbsenceTimestamp(4321, 4321, "4321",
                note.text.toString(), untilTracking.isChecked,
                selectedAbsence)
            dismiss()
        }
        else{
            Toast.makeText(requireActivity(), "Benötigte Daten nicht
                Ausgefüllt!", Toast.LENGTH_SHORT).show()
        }
    }
    else
    {
        if(::formattedChosenDate.isInitialized &&
            ::chosenStartTime.isInitialized &&
            ::chosenStopTime.isInitialized ){
            sendAbsenceTimestamp(4321, 4321, "4321",
                note.text.toString(), untilTracking.isChecked,
                selectedAbsence)
            dismiss()
        }
        else{
            Toast.makeText(requireActivity(), "Benötigte Daten nicht
                Ausgefüllt!", Toast.LENGTH_SHORT).show()
        }
    }
}
```

7 Entwicklung der Web App

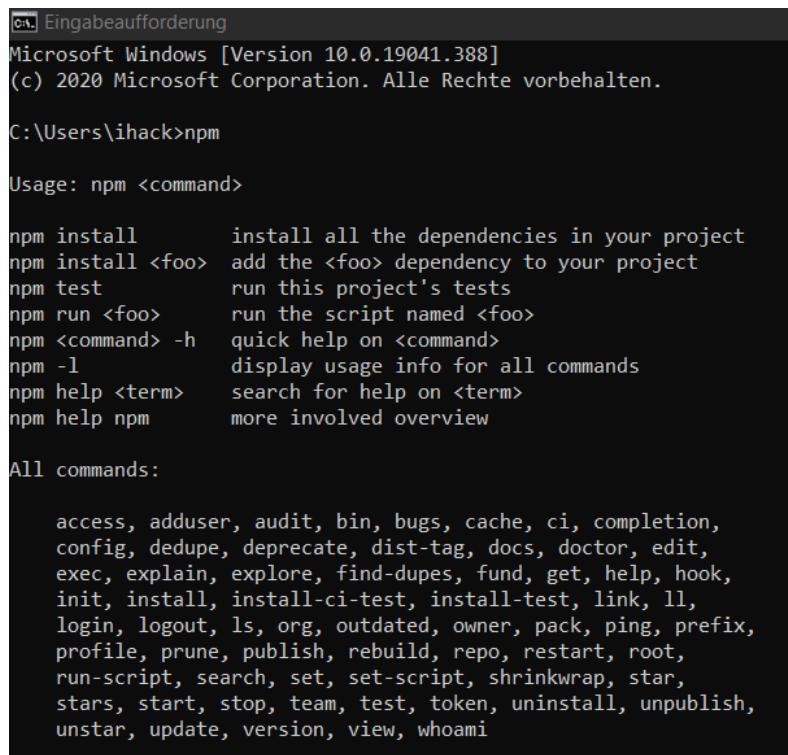
7.1 Vorbereitung

7.1.1 Node.js

Node.js ist eine plattformübergreifende JavaScript Laufzeitumgebung. Node.js erlaubt es einem JavaScript-Code außerhalb eines Webbrowsers auszuführen, daher kann man damit auch einen Webserver betreiben. Über diese Runtime werden die notwendigen Pakete zur Installation/Initialisierung einer Vue.js-Web App über „npm-Befehle“ in der Windows-Konsole (cmd) installiert. Node.js ist eine Open-Source-Software.

Nach dem Download [9] und der Installation ist die Engine in der Windows-Konsole integriert und kann verwendet werden. Um nachzuprüfen, ob Node.js verwendbar ist kann in „cmd“ der command „npm“ eingegeben werden [23].

Danach sollte eine Ausgabe folgen, die folgendermaßen aussieht:



```
C:\ Eingabeaufforderung
Microsoft Windows [Version 10.0.19041.388]
(c) 2020 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\ihack>npm

Usage: npm <command>

npm install      install all the dependencies in your project
npm install <foo>  add the <foo> dependency to your project
npm test         run this project's tests
npm run <foo>    run the script named <foo>
npm <command> -h  quick help on <command>
npm -l           display usage info for all commands
npm help <term>   search for help on <term>
npm help npm     more involved overview

All commands:

  access, adduser, audit, bin, bugs, cache, ci, completion,
  config, dedupe, deprecate, dist-tag, docs, doctor, edit,
  exec, explain, explore, find-dupes, fund, get, help, hook,
  init, install, install-ci-test, install-test, link, ll,
  login, logout, ls, org, outdated, owner, pack, ping, prefix,
  profile, prune, publish, rebuild, repo, restart, root,
  run-script, search, set, set-script, shrinkwrap, star,
  stars, start, stop, team, test, token, uninstall, unpublish,
  unstar, update, version, view, whoami
```

Abbildung 100: cmd

7.1.2 Installation von Vue CLI

Vue CLI ist ein vollständiges System für die schnelle Entwicklung von Vue.js-Applikationen. Unter den Features befindet sich ein Compiler (Syntax-Check), ein Hoster (Anzeigen der Web App auf localhost:8080 [default Port in Vue CLI]) und ein Builder der die einzelnen ausprogrammierten Komponenten in ein index.html-File und einem static-Ordner übersetzt. Diese befinden sich unter dem Ordner „dist/“ im Projektverzeichnis. Das index.html-File und der static-Ordner können so dann auf einem beliebigen http- oder https-Webserver [24].



Zur Klarstellung: Vue CLI ist die softwaremäßige Entwicklungsumgebung. Programmiert wird in der Vue.js-Syntax. Es muss nicht Vue CLI verwendet werden um in Vue.js programmieren zu können. Dieses Framework kann daher auch in der klassischen Websiteprogrammierung (html-files + js-files + css-files) verwendet werden. Dazu muss die Vue.js-Library in Form eines CDN-Links in den head-Tag im html File eingefügt werden.

Installation (cmd-Command):

Newest version (Vue CLI 4 [03.01.2021]):

npm install -g @vue/cli

Vue CLI 4.x setzt die Node.js Version 8.9 oder höher voraus.

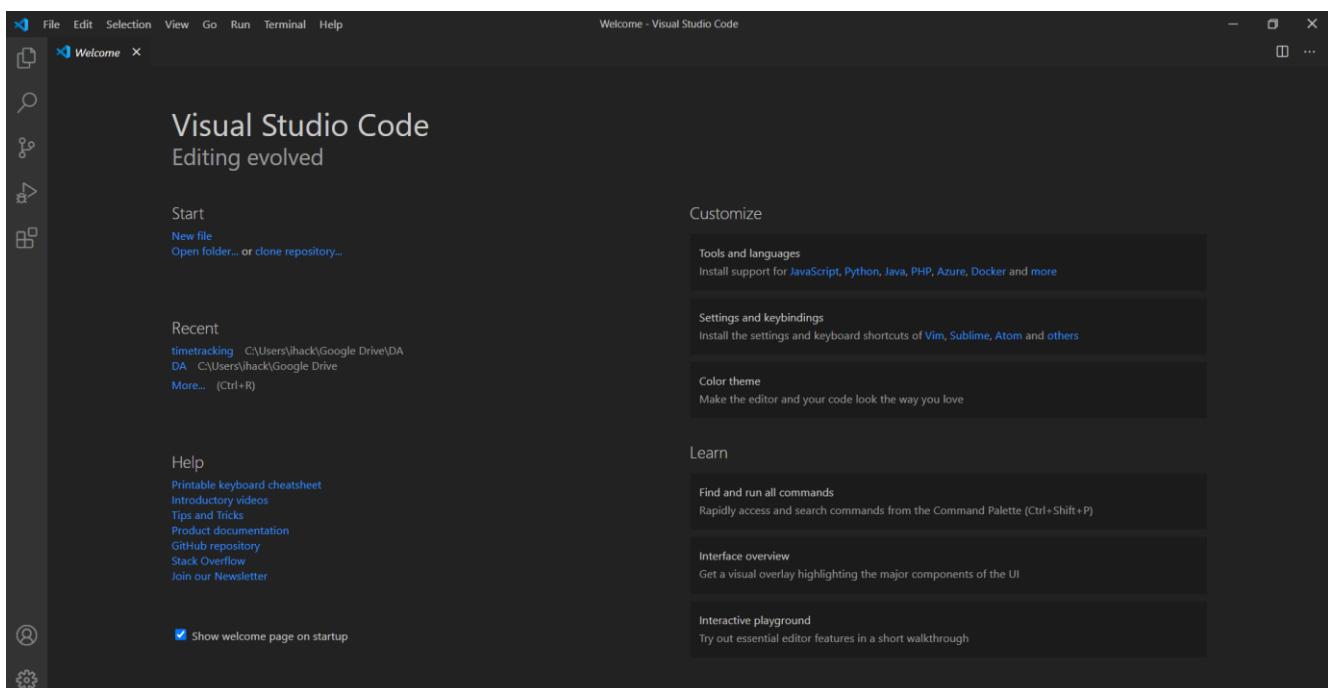
Ist die Installation von Vue CLI abgeschlossen, kann nun ein Vue.js Projekt angelegt werden.

7.1.3 Installation und Einrichtung von Visual Studio Code

Zur Entwicklung einer Vue.js Web App ist die Verwendung von Visual Studio Code (von Microsoft; Abkürzung: VS Code) empfehlenswert, da diese Entwicklungsumgebung nicht nur eine Freeware ist, sondern auch aufgrund ihrer Erweiterbarkeit durch verschiedene Extensions praktikabel ist. Unter diesen Extensions befinden sich auch eine Vue.js Erweiterung mit der der Programmcode als Vue.js Code erkannt wird und die Syntax gehighlightet wird.

Nach der Installation:

Ist die Installation abgeschlossen sollte nach dem Öffnen von VS Code das folgende Fenster angezeigt werden:



Einrichtung:

Für die Vue.js-Entwicklung ist die Windows-Konsole essenziell. Um die Konsole einzublenden kann die Tastenkombinationen STRG + ö verwendet werden. Dadurch ergibt sich folgende Ansicht:

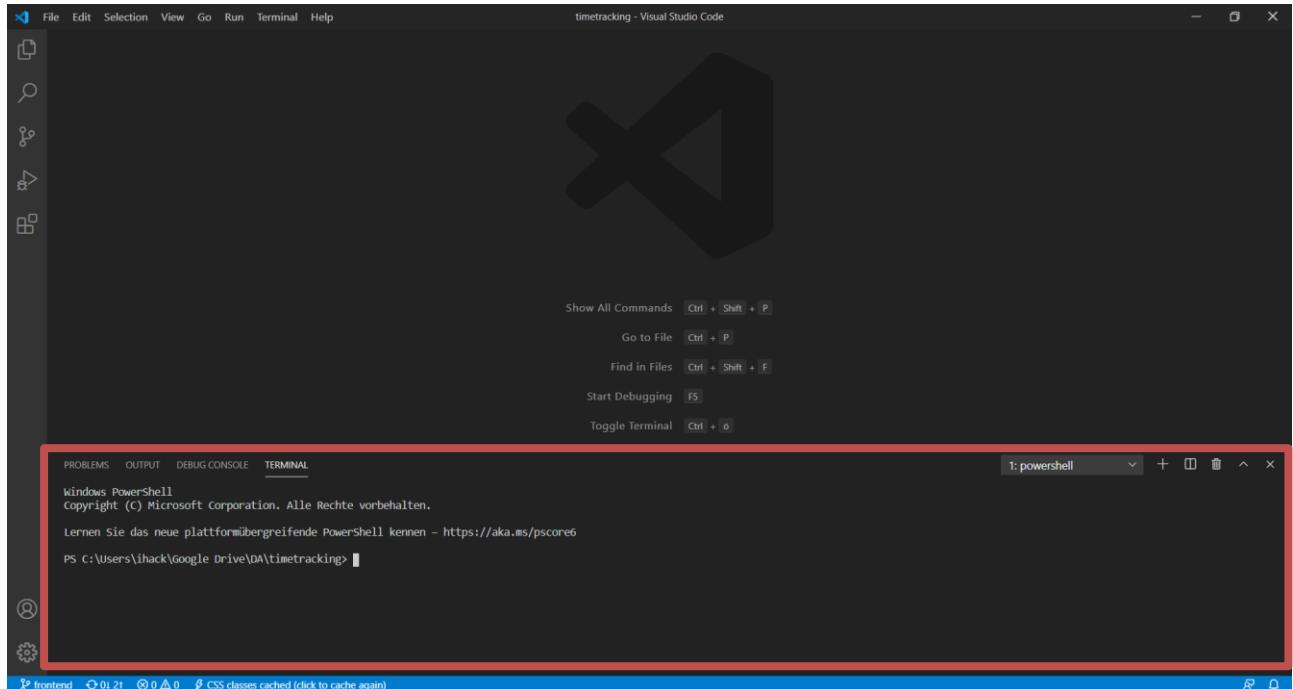


Abbildung 102: VSCode Konsole

Über dieses Terminal kann in weiterer Folge das VUE CLI Projekt angelegt, das Projekt am localhost hochgefahrene und gebuildet werden.

7.1.3.1 Erweiterungen

Es ist empfehlenswert für das Vue.js hilfreiche Erweiterungen hinzuzufügen. Die Erweiterungen können durch Klicken folgender Buttons aufgerufen werden:

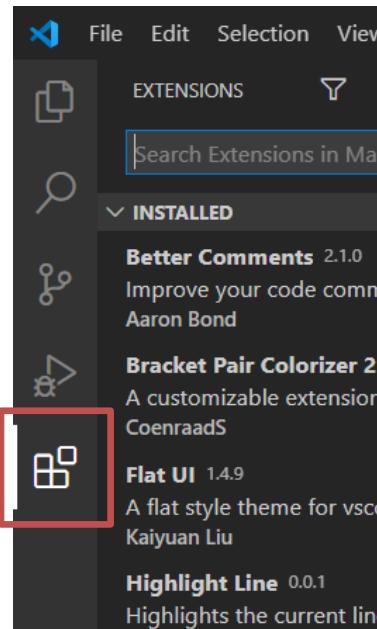


Abbildung 103: VSCode Erweiterungen

Über die Suchzeile kann nun der Extension-Name eingegeben werden. Ist in der Auflistung die Extension gefunden, kann diese ausgewählt und installiert werden:

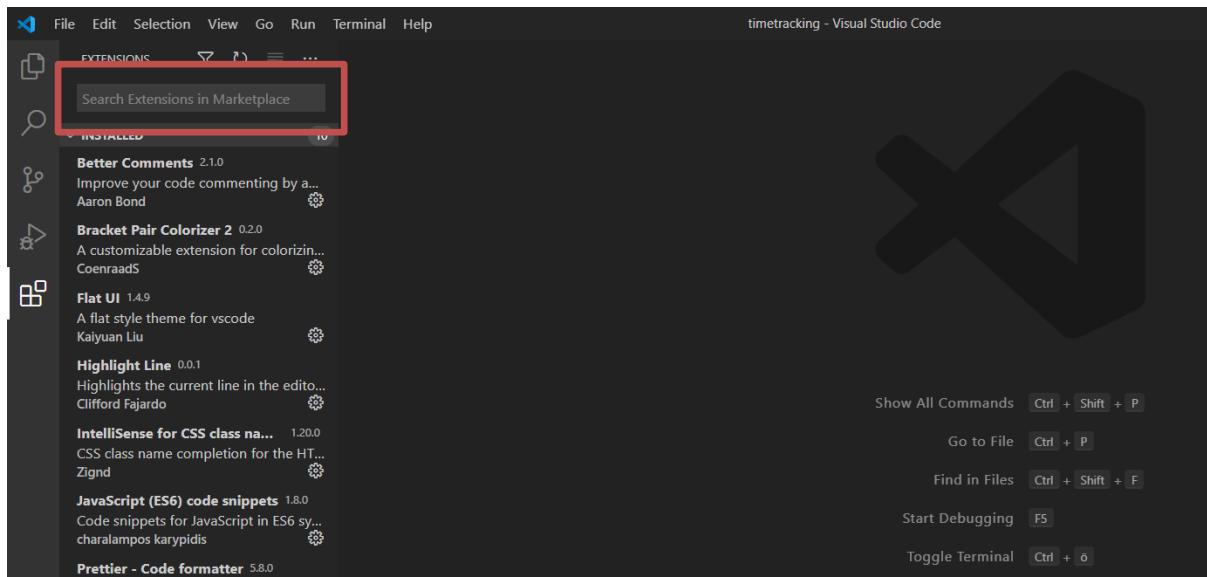


Abbildung 104: Suchfeld

7.1.3.2 Vetur

Diese Erweiterung sorgt für das Highlighten des Vue.js-Programmcodes.

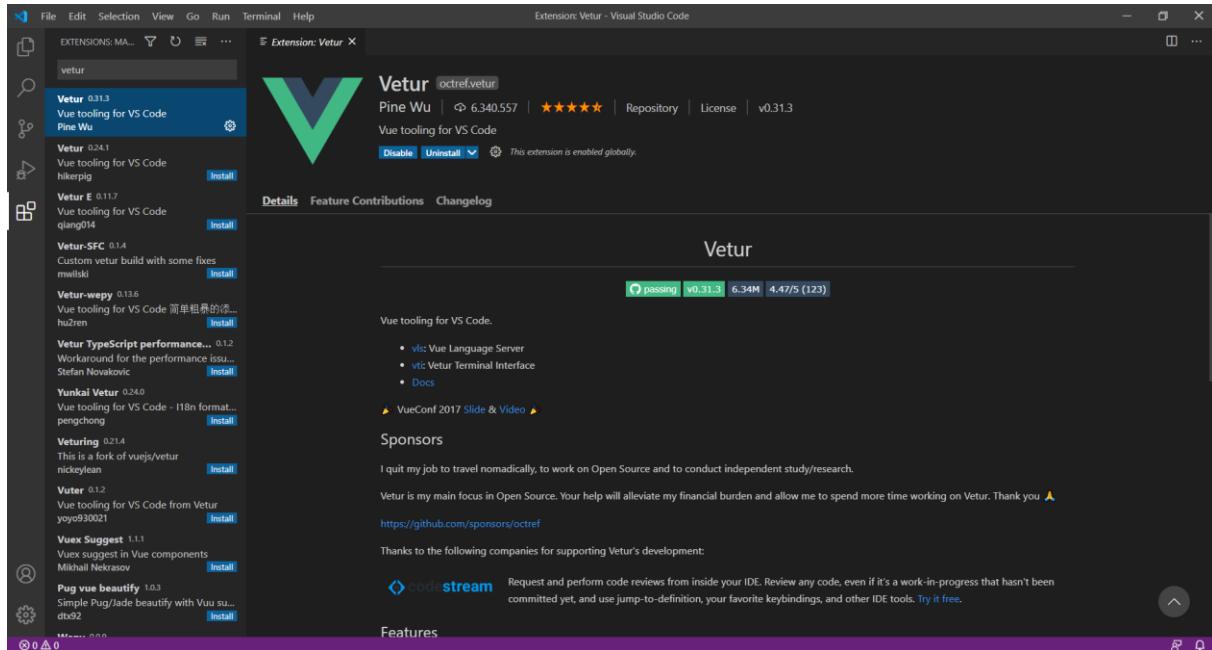


Abbildung 105: Vetur

7.1.3.3 Vue Snippets

Vue Snippets liefert ein Code-Snippets für Vue.js.

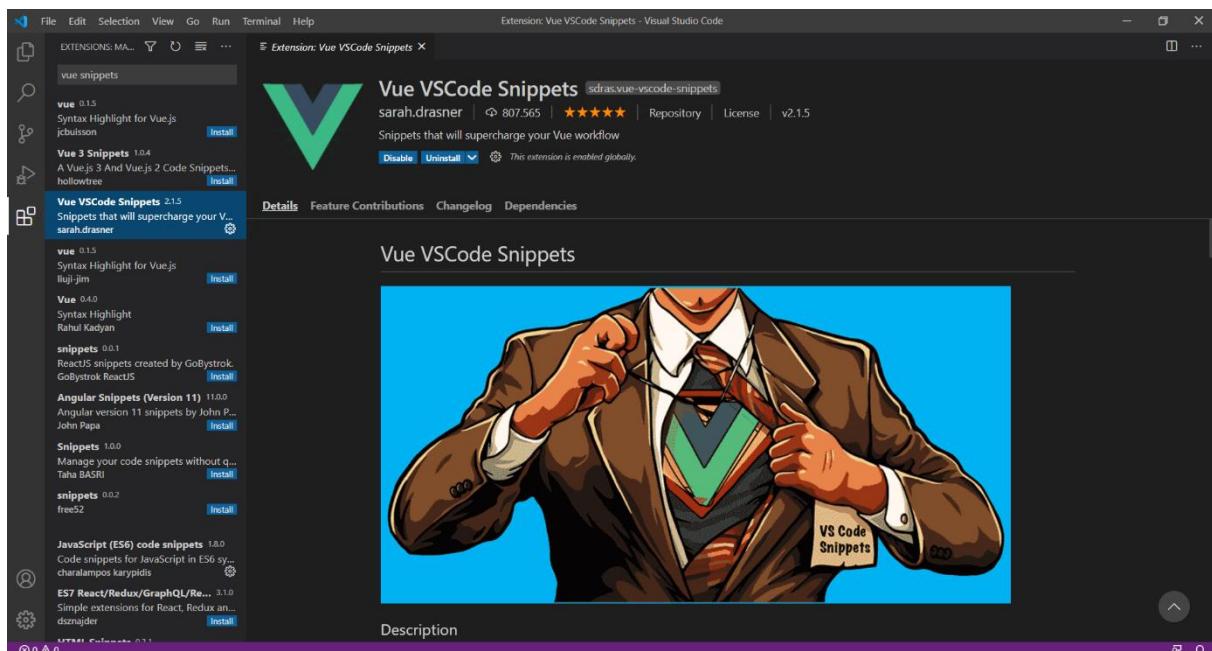


Abbildung 106: Vue VSCode Snippets



7.1.3.4 Prettier

Prettier ist ein Code-Formatierer, welcher den Programmcode besser lesbar macht.

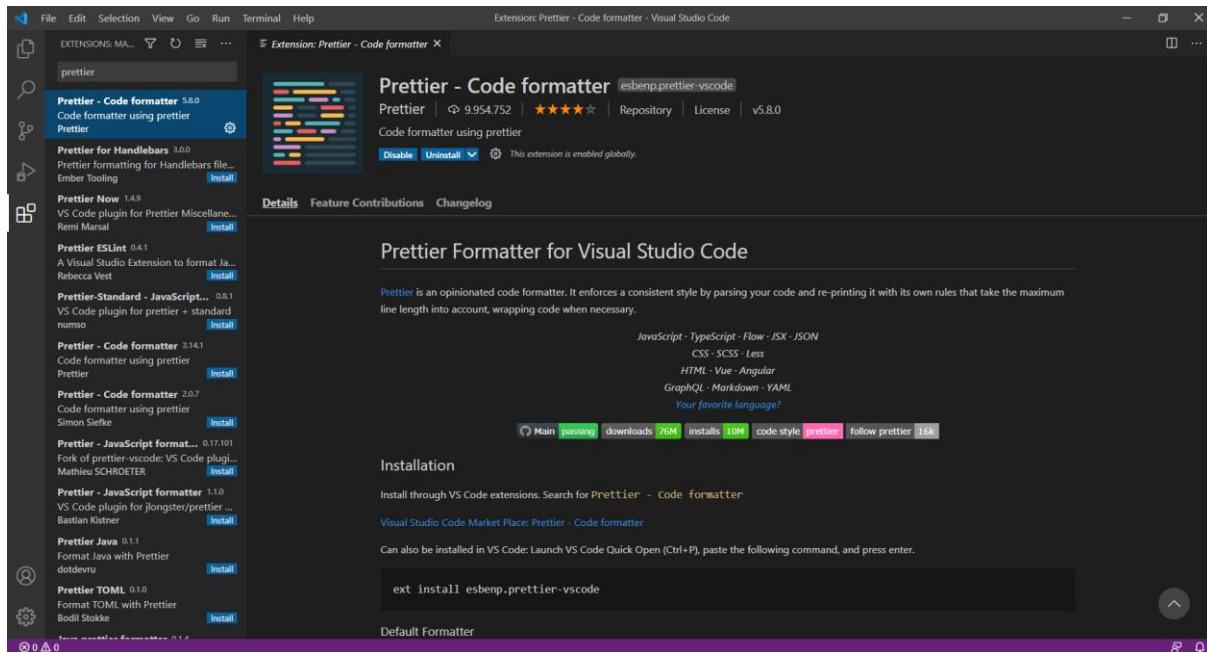


Abbildung 107: Prettier

7.1.3.5 Highlight Line

Highlight Line, markiert die derzeitige Zeile, in welcher der Cursor steht.

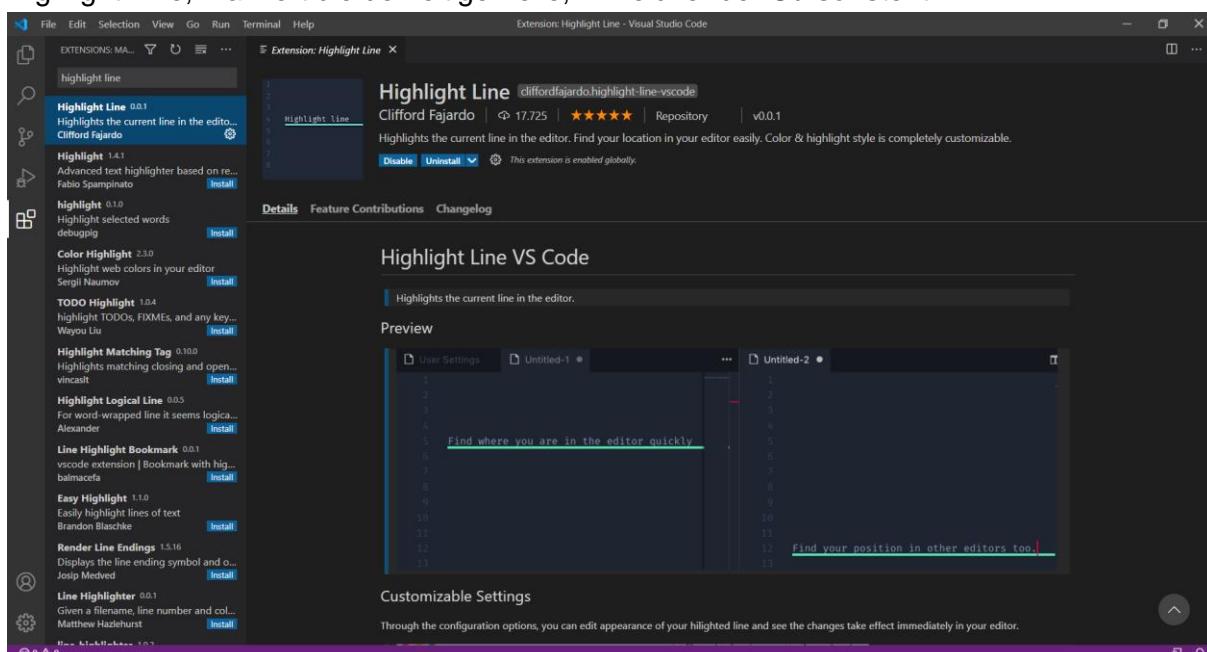


Abbildung 108: Highligth Line

7.1.3.6 Bracket Pair Colorizer 2

Bracket Pair Colorizer 2 markiert bestimmte Klammersegmente mittels Farben, um eine bessere Übersicht zu haben.

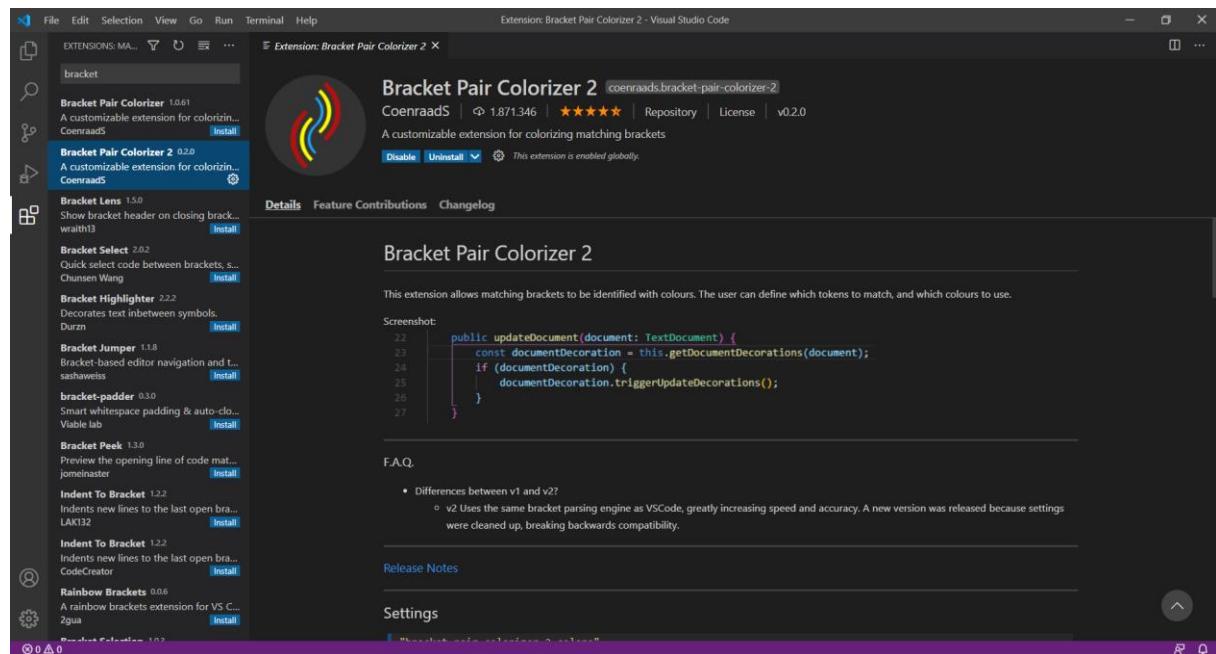


Abbildung 109: Bracket Pair Colorizer 2

7.1.3.7 Better Comments

Better Comments erlaubt eine ausführlichere Kommentierung und Dokumentation des Codes.

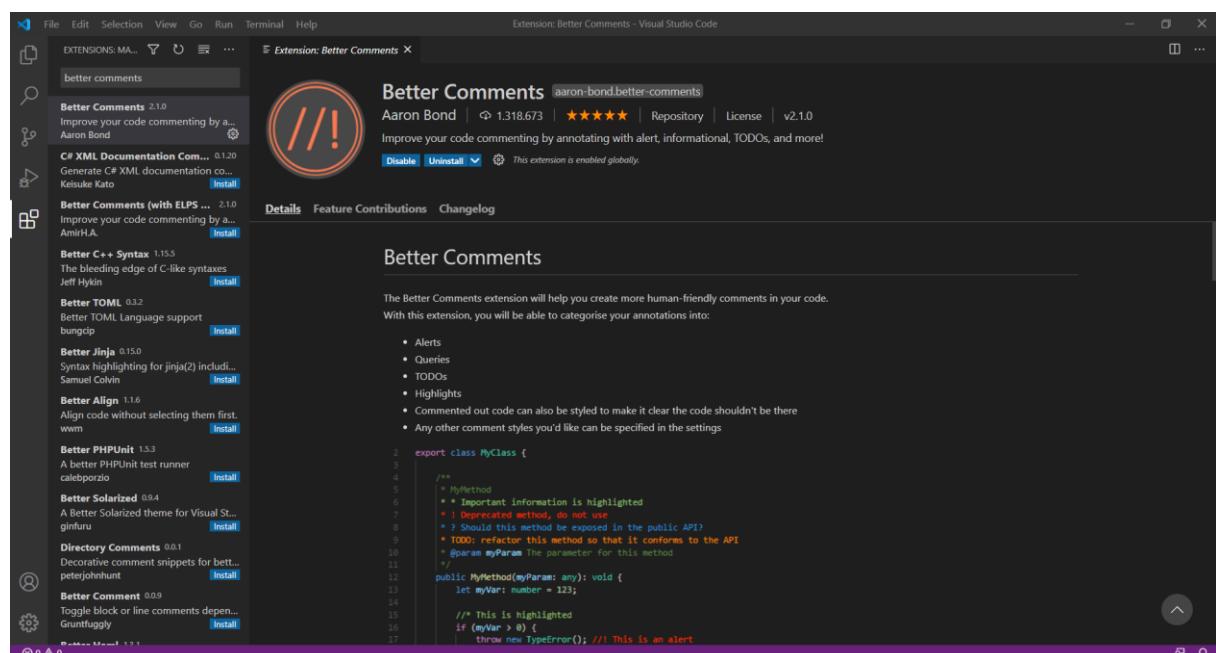


Abbildung 110: Better Comment



7.1.3.8 IntelliSense for CSS

IntelliSense for CSS ist eine intelligente Anwendung, welche schon zu Beginn alle verfügbaren Klassennamen in CSS abfragt und anzeigt.

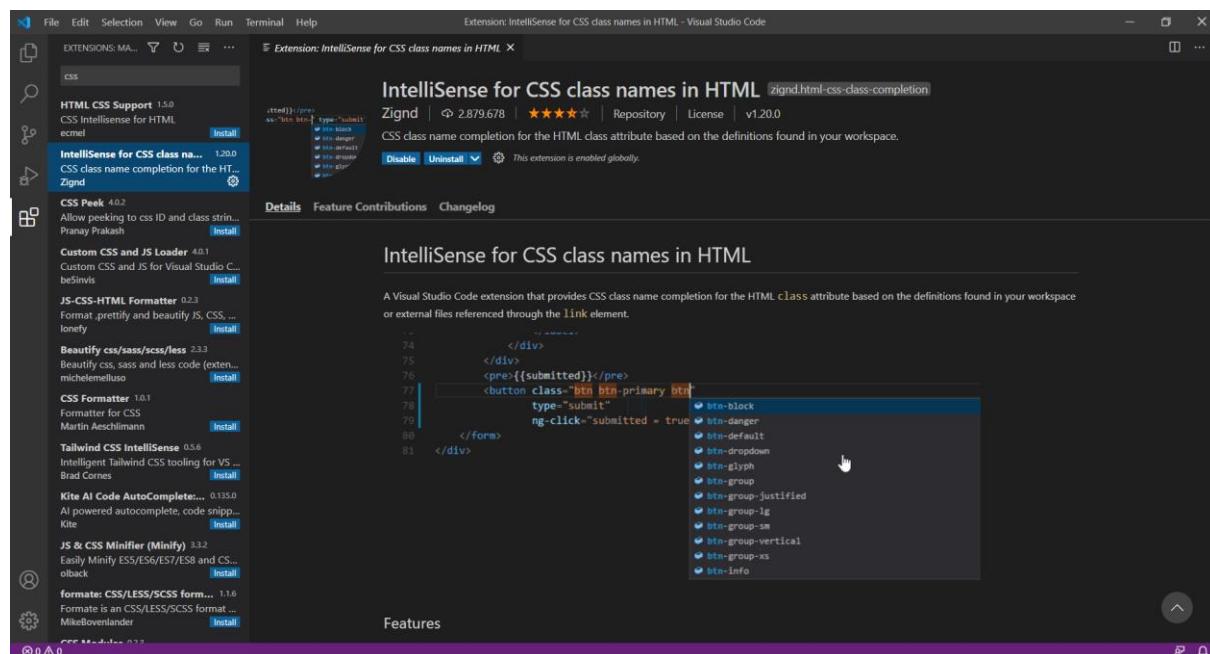


Abbildung 111: IntelliSense CSS

7.1.3.9 JavaScript (ES6) Snippets

JavaScript (ES6) Snippets bietet aktuelle Code-Snippets für JavaScript an. Damit lassen sich auch Konsolen Logs schnell erstellen.

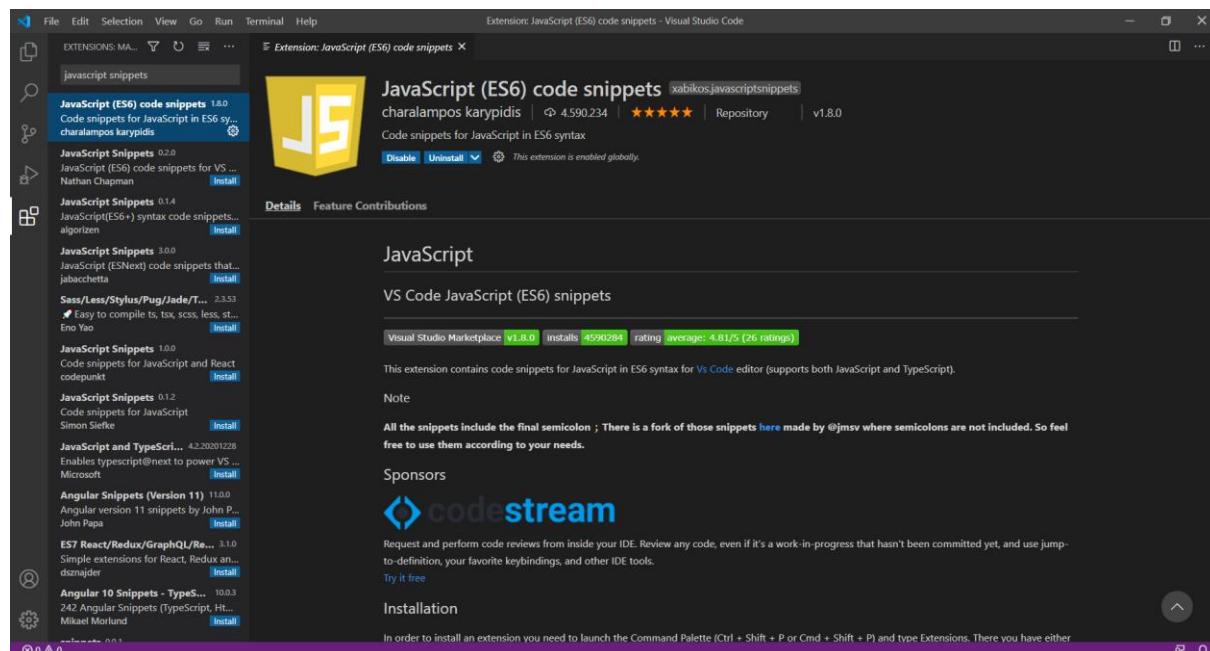


Abbildung 112: JavaScript Snippets

7.1.3.10 Vscode-icons

Vscode-icons ist eine Library, welche File-Icons zur Verfügung stellt.

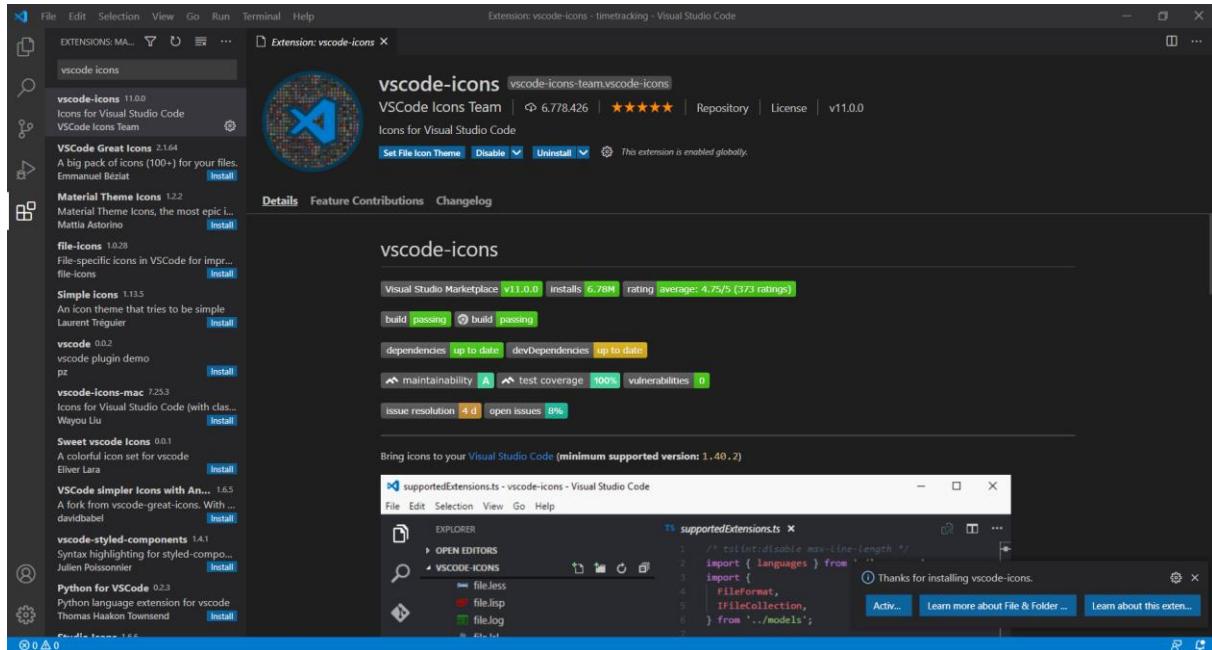


Abbildung 113: VSCode-Icons

7.1.3.11 Flat UI

Als Theme wird Flat UI empfohlen. Dabei handelt es sich um einen grafisch minimalistischen Gestaltungsstil.

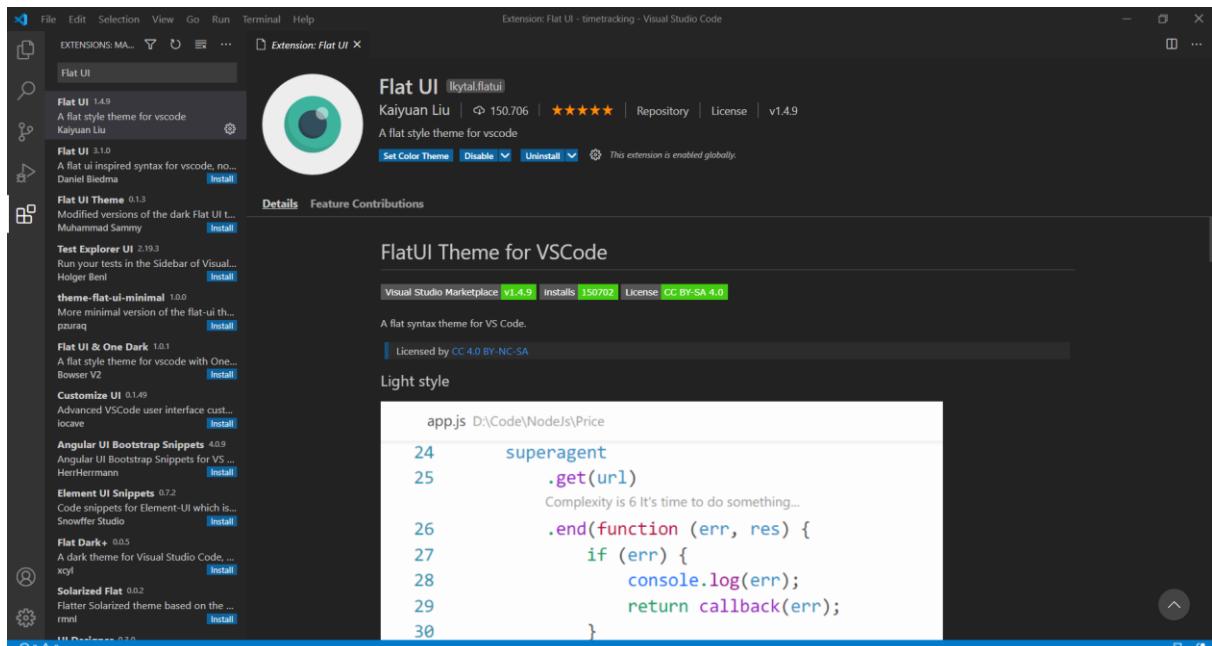


Abbildung 114: Flat UI

Danach kann mit der Projektinitialisierung fortgefahren werden.

7.2 Theoretische Kenntnisse

7.2.1 Lifecycle einer Vue.js Web App

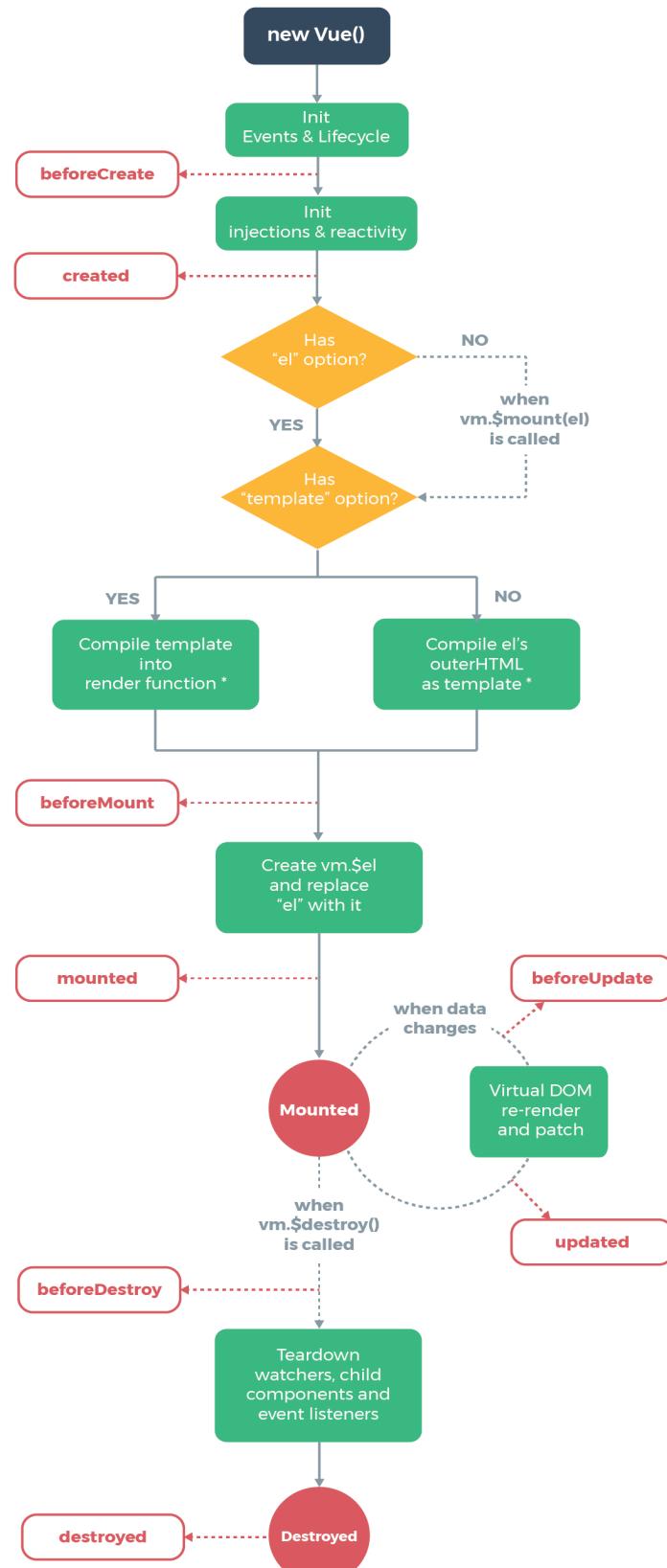


Abbildung 115: Lebenszyklus Vue [25]

Erläuterung:

Mit „new Vue()“ wird eine neue Instanz erstellt. Bei der Initialisierung (init) kann bereits die Funktion „created()“ genutzt werden. Diese wird ausgeführt, wenn die Initialisierung fertiggestellt ist und kann z.B.: dazu verwendet werden, um vorab Daten von einer Datenbank abzurufen. Weiters wird der html-Inhalt stehend im template-tag gemounted, was bedeutet, dass er angezeigt wird. Dabei kann die Funktion mounted() verwendet werden, um diverse Bearbeitungen durchzuführen. Nach dem die App gemounted ist, wird bei jeder Änderung der Daten (data) die App geupdated. Bei Beendigung der App werden alle Components, Watcher, Listeners, etc. abgeschaltet und die Instanz destroyed.

7.3 JavaScript

Als Programmiersprache in den Vue Components wird JavaScript verwendet: „JavaScript (kurz JS) ist eine Skriptsprache, die ursprünglich 1995 von Netscape für dynamisches HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von HTML und CSS zu erweitern. Heute findet JavaScript auch außerhalb von Browsern Anwendung.“

Vorteile:

- Die Syntax ähnelt der Programmiersprache C (Schleifen (for), Verzweigungen (if), ...)

Nachteil:

- Ein großer Vor-, aber zugleich auch ein großer Nachteil von JavaScript ist, dass es (syntaxmäßig) sehr viel Handlungsspielraum zulässt. Dadurch kann es zu unübersichtlichen Programmteilen kommen. Daher muss der Programmierer bzw. die Programmiererin genau darauf achten denselben Workflow beizubehalten, um die Übersichtlichkeit zu gewährleisten.

7.3.1 Wichtige Konstrukte

7.3.1.1 Async & Await

Async / Await wird oft bei REST-API-Anfragen verwendet, bei der die Daten vollständig geladen werden sollen, bevor man die Ansicht rendert. Es hilft dem Entwickler, funktionale Programmierung in JavaScript zu implementieren, und erhöht auch der Lesbarkeit des Codes.

```
const showPosts = async () => {
  const response = await fetch("https://jsonplaceholder.typicode.com/posts");
  const posts = await response.json();
  console.log(posts);
};

showPosts();
```

Listing 107: Asynch & await-Code



7.3.1.2 Callbacks

In JavaScript sind Callbacks Funktionen, welche als Parameter auch Funktionen haben, die innerhalb dieser Funktion ausgeführt werden. Meistens werden diese bei asynchronen Operationen verwendet, da diese Funktionen eine gewisse Synchronität bieten.

```
let x = function () {
    console.log("Ich werden aus einer Funktion heraus aufgerufen");
};

let y = function (callback) {
    console.log("mach mal wieder was");
    callback();
};

y(x);
// Output:
// mach mal wieder was
// Ich werden aus einer Funktion heraus aufgerufen
```

Listing 108: Callbacks in JavaScript

7.3.1.3 Ausführen zeitbedingter Funktionen

Um eine Funktion nach einer definierten Zeit auszuführen kann die `setTimeout()`-Funktion verwendet werden.

```
setTimeout(function () {
    alert("Hello");
}, 3000);
```

Listing 109: Zeitbedingter Code

7.3.1.4 This-Zeiger

Um auf die „Component-data“ in einer Funktion zugreifen zu können, wird der sogenannte „this-Pointer“ angewendet. Hier wird der this-Zeiger (dieser zeigt auf die zuvor instanzierte Komponente) in eine neu angelegte Variable, meist „vm“ oder auch „self“ genannt, abgespeichert. Nun kann man in einer Funktion auf die „Component-data“ zugreifen.

```
new Vue({
    data: {
        a: 1
    },
    created: function () {
        // `this` points to the vm instance
        console.log('a is: ' + this.a)
    }
})
```

```
})
// => "a is: 1"
```

Listing 110: This-Zeiger

7.4 Vue.js

„Vue.js ist ein clientseitiges JavaScript-Webframework zum Erstellen von Single-Page-Webanwendungen. Das Framework kann allerdings auch in Multipage Webseiten für einzelne Abschnitte verwendet werden. Ab Version 2.0 unterstützt es auch serverseitiges Rendern.“ [26]

7.4.1 Instanz

Die sogenannte „Vue-Instanz“ wird mit der Eigenschaft „el“ an einen HTML-Knoten gebunden.

7.4.2 Komponenten

Vue.js verwendet eine HTML-basierte Templatesyntax, mit der man das gerenderte DOM an die Daten der zugrunde liegenden Vue-Instanz binden kann (auch Verschachtelungen sind möglich). Komponenten enthalten daher ein eigenes HTML-Template und werden nicht an HTML-Knoten angebunden.

```
Vue.component("alert-box", {
  template: `<div class="demo-alert-box">
    <strong>Error!</strong>
    <slot></slot>
  </div>`,
}) ;
```

Listing 111: Vue-Komponente

7.4.3 Double Curly Syntax

Um in einem Template-Tag auf die Daten zugreifen zu können, wird die „Double Curly Syntax“ verwendet. Dabei wird mit doppelt geschwungenen Klammern auf eine Vue-Variable zugegriffen. Generell unterliegen die Variablen einem „Two-way data binding“. Das heißt, dass Variablen einfach von einem User manipuliert werden können, ohne dass man die Variable explizit verfolgen und aktualisieren muss.

```
<select v-model="selected">
  <option disabled value="">
    Please select one
  </option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
```



```
<span>Selected: {{selected}}</span>

new Vue({
  el: '#app',
  data: { selected: '' }
})
```

Listing 112: Double Curly Syntax

Durch das Attribut v-model lassen sich Variablen für “Two-way data binding” festlegen.

7.4.4 Iterationen

7.4.5 Single File Komponenten

Der Aufbau eines Single File Components beinhaltet einen template-Teil, einen Script-Teil und ein Style-Teil. Diese werden jeweils mit dem Tag eingeleitet (z.B.: template). In den template-Teil kommt der HTML-Code. An dieser Stelle können nun eben auch die Vue-Aktionen, wie beispielsweise Schleifen (v-for) benutzt werden. Im script-Teil werden unter der data-property die Component-Variablen definiert, die sowohl im script-tag als auch im template-tag zugreifbar (two-way-binding) sind.

```
<template>
  <div class="shine">{{ message }}</div>
</template>

<script>
export default {
  data: () => ({
    message: "Life is good",
  }),
};
</script>

<style scoped>
.shine {
  font-size: 3 rem;
  text-align: center;
}
</style>
```

Listing 113: Single File Komponenten

Dabei können aber z.B. der Script-Teil und der Style-Teil weggelassen werden, wenn man nur eine reine Anzeige-Seite implementieren möchte. Eine User-Eingabe und eine entsprechende Datenverarbeitung sind in diesem Fall nicht erforderlich.

7.4.6 Navigation Guards

Der Vue Router bietet neben einem Routing der Komponenten auch noch sogenannte NavGuards an. Diese schützen unangemessene Navigationen mittels Umleitens oder Abbrechen. Dabei gibt es verschiedene Möglichkeiten, sich in den Routennavigationsprozess einzubinden: Global, auf eine Route bezogen oder in der Komponente.

```
router.beforeEach((to, from, next) => {
  if (to.name !== 'Login' && !isAuthenticated) next({ name: 'Login' })
  else next()
})
```

Listing 114: Navigation Guards

Erklärung: BeforeEach bedeutet, dass dieser Codeblock bei jeder Navigation durchlaufen wird (also Global). Der Parameter „to“ steht für die Zielroute, „from“ steht für die Quellroute und der Parameter „next“ liefert eine Funktion für die Manipulation der nächsten Route. Innerhalb dieses Guards befindet sich eine Verzweigung, welche eine Route vergleicht, und ein Variablenwert. Ist dieser Vergleich richtig, so wird zur Route mit dem Namen „Login“ navigiert. Ist dieser Vergleich falsch, so darf der User zur gewünschten Route weitergeleitet werden.

7.4.7 Vuetify

Vuetify ist ein vollständiges UI-Framework, das auf Vue.js basiert. Entwicklern werden damit Tools an die Hand gegeben, die man für das Erstellen einer umfassenden und ansprechenden Benutzererfahrung benötigt. Die bereitgestellten Komponenten von Vuetify sind mittels Material-Design konzipiert [27].

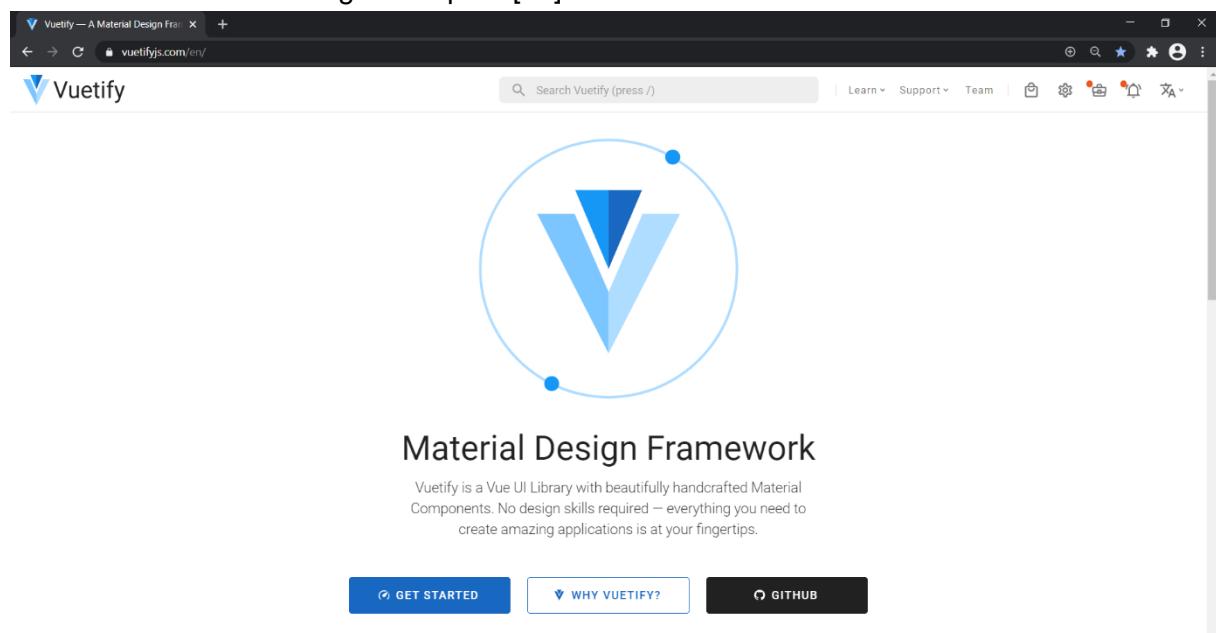


Abbildung 116: Vuetify [28]

Mit dem Befehl `vue add vuety` kann das Plugin hinzugefügt werden.

7.5 Vuex

Vuex ist eine Bibliothek, welche die Verwaltung von States erlaubt. Sie wurde eigens für Vue.js entwickelt. Vuex dient als globaler Speicher für alle Komponenten einer Anwendung, wobei Regeln sicherstellen, dass der State nur auf vorhersehbare Weise mutieren kann. Dieses Shared State Management Package ist bei großen Applikationen zu empfehlen. Die Grundidee hinter Vuex wurde von Flux, Redux und der Ulmenarchitektur inspiriert.

Hinweis: Die Vue-Komponenten können auch weiterhin ihren eigenen State besitzen.

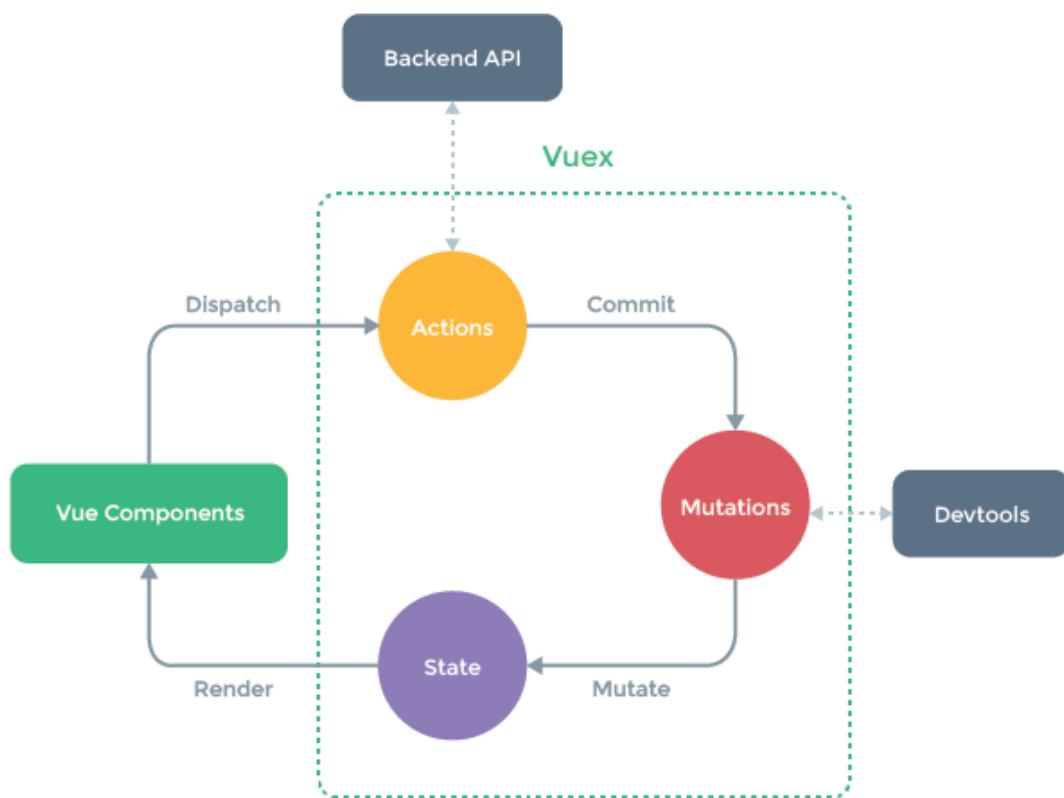


Abbildung 117: Vuex [29]

Erklärung:

Der Speicher besitzt einen State, dieser wird den Komponenten übergeben. Die Vue-Komponenten können wiederum Actions auslösen, welche mit dem Backend kommunizieren. Actions können danach Mutations aufrufen, die den State im Speicher manipulieren. Der State wird geupdatet und den Komponenten mitgeteilt.

Mutations sollten immer nur synchrone Aufgaben durchführen. Die Actions sind für asynchrone Tätigkeiten zuständig. Es gibt auch noch neben dem State, die Getters, diese sind jedoch nur für das Zurückgeben des States verantwortlich.

7.5.1 Store

```
const store = new Vuex.Store({
    // State
    state: {
        count: 0
    },
    // Mutations
    mutations: {
        increment (state) {
            state.count++
        }
    },
    // Actions
    actions: {
        increaseCounter ({commit}) {
            commit("increment")
        }
    },
    // Getters
    getters: {
        getCounter: (state) => state.count
    }
})
```

Listing 115: Vuex Store

7.6 Internationalisierung

Damit die Vue-Applikation leicht in andere Sprachen übersetzt werden kann, wird eine Internationalisierungssoftware verwendet. Mit dieser Software lässt sich der zu übersetzende Quellcode zur Laufzeit rendern.

7.6.1 Vue I18n

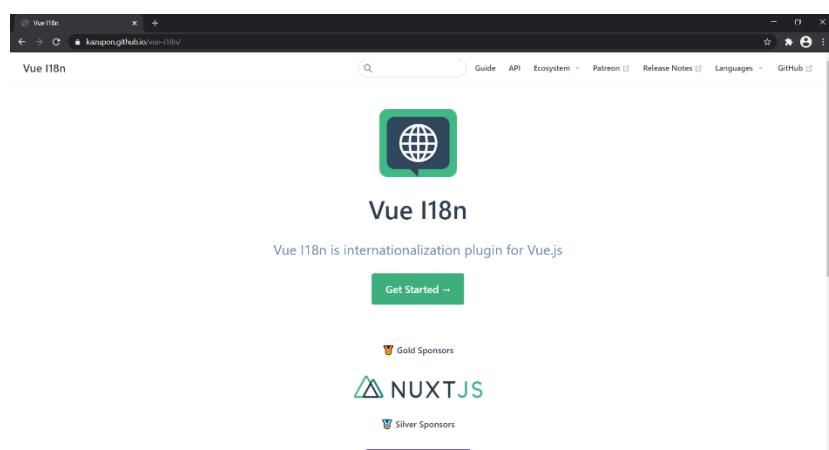


Abbildung 118: Vue I18n [30]



Vue I18n ist eine Internationalisierung-Library eigens für Vue.js entwickelt. Das Plugin erlaubt eine einfache Übersetzung für eine Vue.js-Applikation.

7.7 Entwicklungsumgebung

7.7.1 Projekterstellung

Ein Vue Projekt kann mittels vue create im gewünschten Zielverzeichnis erstellt werden. In diesem Fall wird der Name „timetracking“ gewählt:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\ihack\Google Drive\DA>
PS C:\Users\ihack\Google Drive\DA> vue create timetracking
```

Abbildung 119: VUE CLI Projekterstellung

Nach der Ausführung des Befehls wird man auf folgende Einstellungsmöglichkeiten stoßen: Warum die folgenden Konfigurationen vorgenommen wurden und, was die einzelnen Begriffe bedeuten, wird in späteren Textpassagen ausführlich erläutert.

1.)

```
vue CLI v4.5.9
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
> Manually select features
```

Abbildung 120: Projektauswahl

Hier „Manually“ wählen

2.)

```
? Check the features needed for your project:
>(*) Choose Vue version
(*) Babel
( ) TypeScript
( ) Progressive Web App (PWA) Support
(*) Router
(*) Vuex
( ) CSS Pre-processors
(*) Linter / Formatter
( ) Unit Testing
( ) E2E Testing
```

Abbildung 121: Featureauswahl

3.)

```
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, Linter
? Choose a version of Vue.js that you want to start the project with (use arrow keys)
> 2.x
  3.x (Preview)
```

Abbildung 122: Vue Versionsauswahl

4.)

```
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) (Y/n) y
```

Abbildung 123: History mode

5.)

```
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a linter / formatter config:
  ESLint with error prevention only
  ESLint + Airbnb config
  ESLint + Standard config
> ESLint + Prettier
```

Abbildung 124: Linter und Formatter-Auswahl

6.)

```
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a linter / formatter config: Prettier
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert selection)
>(*) Lint on save
( ) Lint and fix on commit
```

Abbildung 125: Lintingauswahl

7.)

```
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a linter / formatter config: Prettier
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
> In dedicated config files
  In package.json
```

Abbildung 126: Einstellungsfile

8.)

```
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a linter / formatter config: Prettier
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? (y/N) 
```

Abbildung 127: Einstellungen speichern



Hier kann selbst entschieden werden, ob man die Einstellungen für dieses Projekt für zukünftige Projekte speichern möchte. (Wenn man ähnliche Projekte machen möchte, dann empfiehlt es sich, die Einstellung zu speichern. Möchte man die Einstellungen nicht speichern, ist „N“ oder „n“ einzugeben)

Nachdem man auch hier die Eingabe gedrückt hat, ist die Projektkonfiguration abgeschlossen und das Projekt wird erstellt. Dabei wird ein Ordner mit dem Ordnernamen des Projektnamens (in vorliegenden Fall „timetracking“) erstellt. Während der Projekterstellung werden die notwendigen Dependencies heruntergeladen und in den Ordner „node_modules“ innerhalb des Projektordners abgelegt.

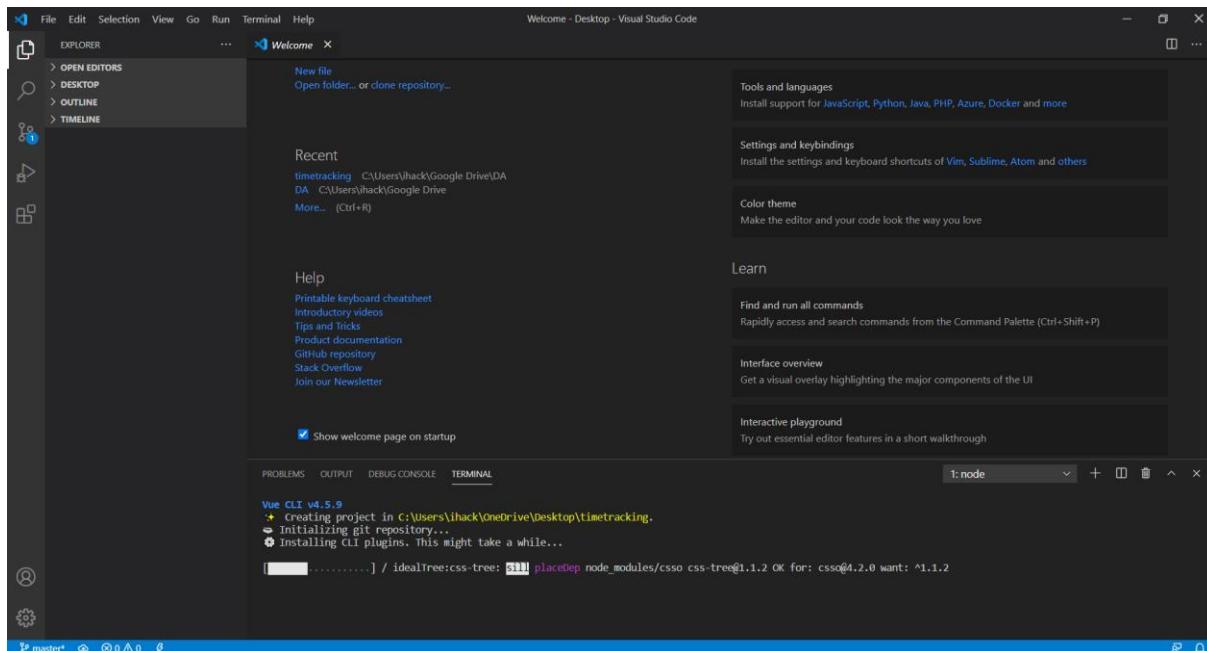


Abbildung 128: Projektgenerierung

In dem darüberliegenden Screenshot kann man erkennen, dass der Projektordner erstellt wurde und die Dependencies heruntergeladen werden.

Ist der Prozess abgeschlossen, gelangt man zu folgendem End-Screen:

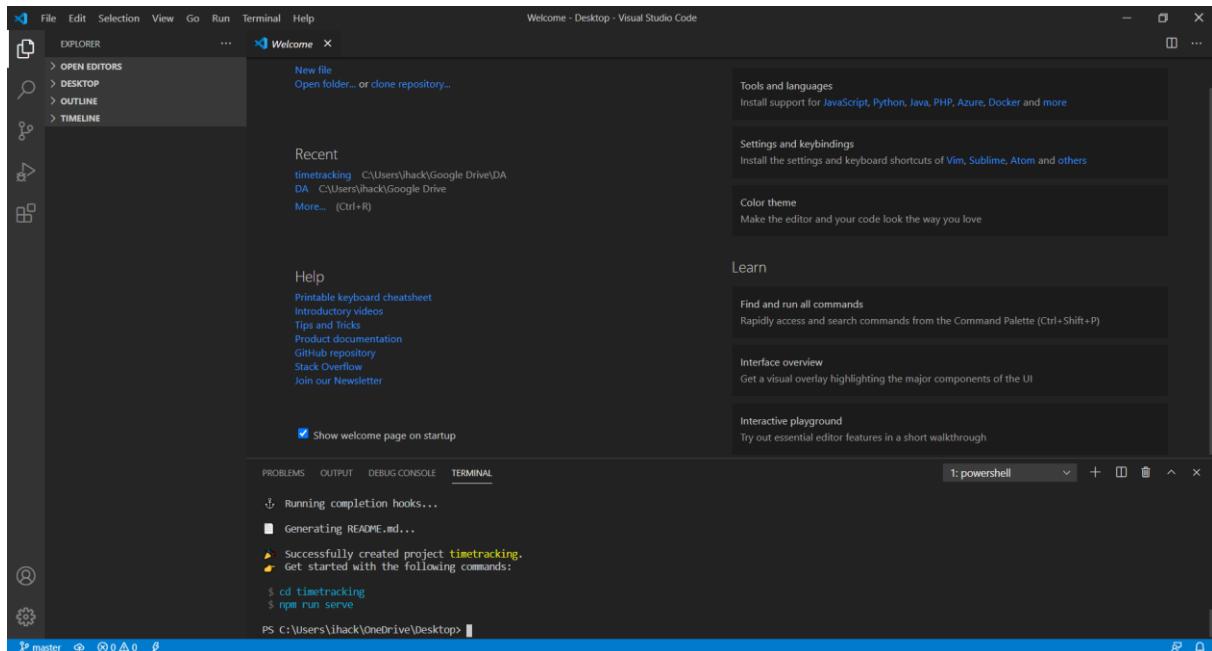


Abbildung 129: Projektgenerierung abgeschlossen

Nach Abschluss der Projekterstellung, scheinen die nächsten Anweisungen im Terminal auf. Durch Ausklappen des Projektordners „timetracking“ ist der Verzeichnisinhalt ersichtlich. Den im Terminal stehenden Anweisungen folgend, muss nun mit dem Befehl „cd timetracking“ in das Projektverzeichnis gewechselt werden.

7.7.2 Projektinitialisierung über Terminal in VS Code

Es müssen verschiedenste npm-Packages installiert werden, um mit der Entwicklung der Web App beginnen zu können.

1) Hinzufügen des Design-Templates Vuetify Terminal Befehl:

vue add vuetify

Danach muss bei der aufkommenden Auswahl „default“ ausgewählt werden.

2) Installieren der http-Library axios Terminal Befehl:

vue add axios

2) Installieren von Internationalisierung Vuei18n Terminal Befehl:

vue add i18n

Hinweis: Bei manchen Packages ist es nötig die Entwicklungsumgebung mit einer Administratorberechtigung zu öffnen.



7.7.3 Testen der Web App

Um zu überprüfen, ob das Projekt korrekt erstellt wurde und mit der Web App-Entwicklung fortgefahren werden kann ist zu empfehlen die automatisch erzeugte Test-Web App auszuführen, um die Funktion zu testen.

Um die Web App ausführen zu können, muss folgender Befehl verwendet werden:

npm run serve

Nach Ausführung dieses Befehls erscheint folgende Terminal-Anweisung:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

App running at:
- Local:  http://localhost:8080/
- Network: http://172.31.129.229:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

Abbildung 130: `npm run serve`

Die IP-Adresse „172.31.129.229“ ist die Adresse des lokalen Rechners. Diese kann durch das Schlüsselwort „localhost“ beim Eingeben im Browser ersetzt werden. Erhält man diese Terminal-Ausgabe wurde der Code erfolgreich kompiliert und auf „localhost:8080“ gehostet. Somit kann nun in jedem beliebigen Browser „localhost:8080“ eingegeben werden, um das Ergebnis des Programmcodes als Web App zu sehen.

Der große Vorteil an dem Hoster von Vue CLI ist, dass während der Entwicklung dynamisch nach einem Speichervorgang neu kompiliert und gehostet wird. Hat man also den Vorgang des Codierens eines Programmabschnittes abgeschlossen und drückt man im Zuge dessen STRG + S (Tastenkombination für File speichern), wird auf „localhost“ die veränderte Web App neu geladen ohne dabei, die Taste „F5“ drücken zu müssen.

7.7.3.1 CORS

CORS steht für Cross-Origin-Ressource-Sharing und ist eine Sicherheitsrichtlinie moderner Web-Browser.

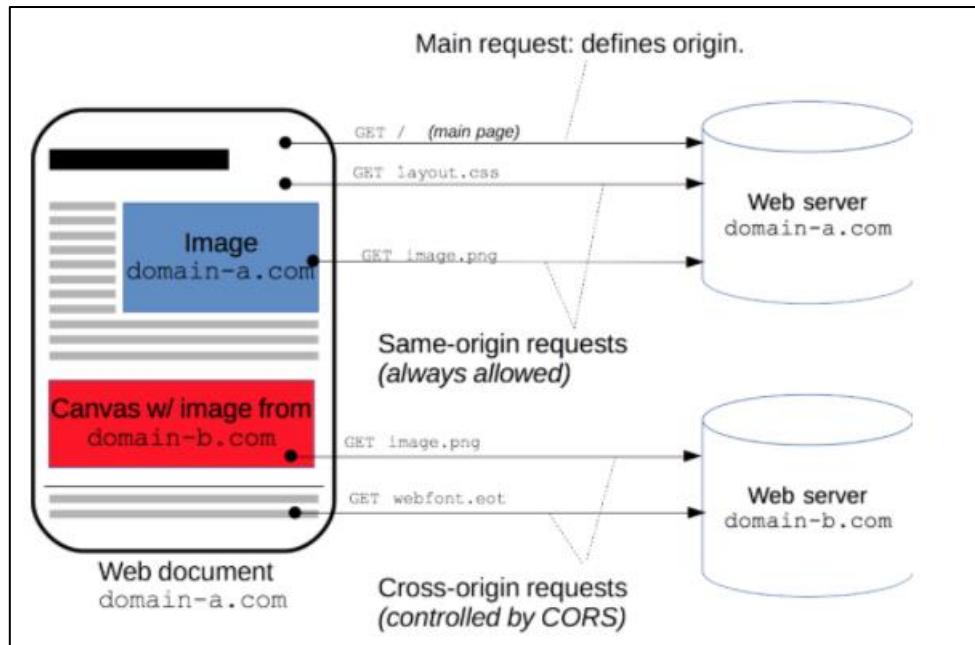


Abbildung 131: CORS [31]

Versucht man sich mit dem Login bei Redmine zu authentifizieren bekommt man ein CORS-Policy-Verstoß.

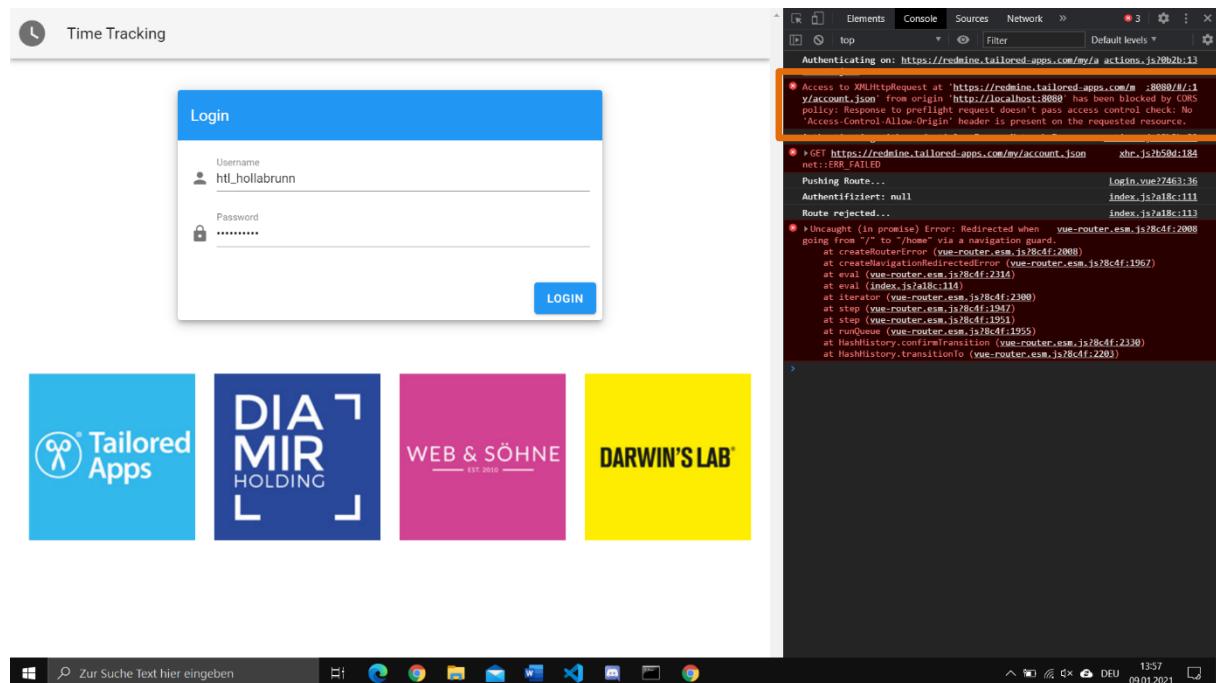


Abbildung 132: CORS Policy Browser



Dabei wird folgender Error in der Konsole des Browsers angezeigt:

```
✖ Access to XMLHttpRequest at 'https://redmine.tailored-apps.com/m :8080/#/:1
y/account.json' from origin 'http://localhost:8080' has been blocked by CORS
policy: Response to preflight request doesn't pass access control check: No
'Access-Control-Allow-Origin' header is present on the requested resource.
```

Abbildung 133: CORS Policy

„Eine Webanwendung, die diese APIs verwendet, kann nur HTTP-Ressourcen aus der gleichen Herkunft anfordern, aus der die Anwendung geladen wurde, es sei denn, die Antwort aus der anderen Herkunft enthält die richtigen CORS-Header.“

Der Grund dafür ist also ein fehlender Access-Control-Allow-Origin-Header im http-Header, dieser muss vom Sender hinzugefügt werden, damit der Browser des Empfängers berechtigt ist diese Daten zu empfangen. Diesen Header selbst hinzuzufügen bringt nichts, da dieser dann als selbstsigniert gilt.

Lösungen für dieses Problem:

1. Ausschalten dieser Richtlinie im Browser (unsicher)
2. Erstellen eines http-Proxies

Für Entwicklungszwecke wurde sich für die erste Lösung gewählt, da diese die schnellere ist.

Mit folgendem Kommando in cmd, können die Sicherheitsrichtlinien in Chrome deaktiviert werden:

chrome.exe --disable-web-security --disable-gpu --user-data-dir=~/chromeTemp

Dafür muss man in den Applications-Ordner von Google Chrome navigieren:

cd .. (im Verzeichnis nach oben)

cd name (Verzeichnis wechseln)

Zielordner:

C:\Program Files (x86)\Google\Chrome\Application

```
Administrator: Eingabeaufforderung
Microsoft Windows [Version 10.0.19042.685]
(c) 2020 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Windows\system32>cd ..

C:\Windows>cd ..

C:>cd "Program Files"

C:\Program Files>cd Google

C:\Program Files\Google>cd Chrome

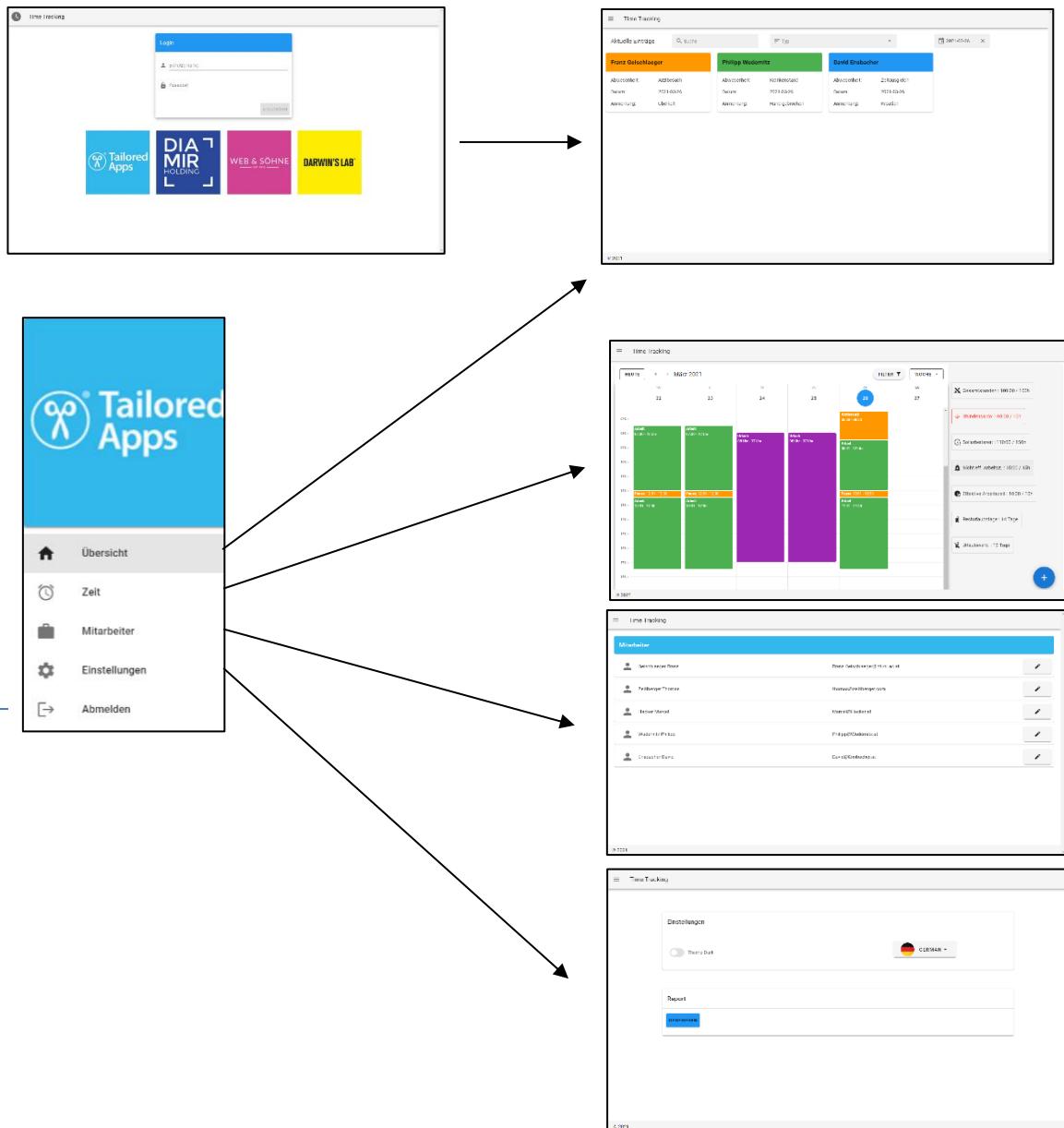
C:\Program Files\Google\Chrome>cd Application

C:\Program Files\Google\Chrome\Application>chrome.exe --disable-web-security --disable-gpu --user-data-dir=~/chromeTemp
```

Abbildung 134: cmd Chrome öffnen

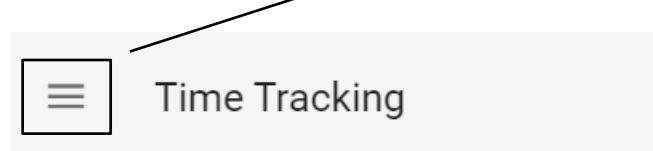
Die Eingabeaufforderung muss mit Administrator-Rechten ausgeführt werden. Anschließend sollte sich Google Chrome ohne Sicherheitsunterstützung öffnen. Nun kann man sich ohne Probleme auf Redmine authentifizieren.

7.8 Navigation und Layout der Web App



7.8.1 Navigationsbar

Aufrufen der Sidebar



7.8.1.1 Sidebar – Admin

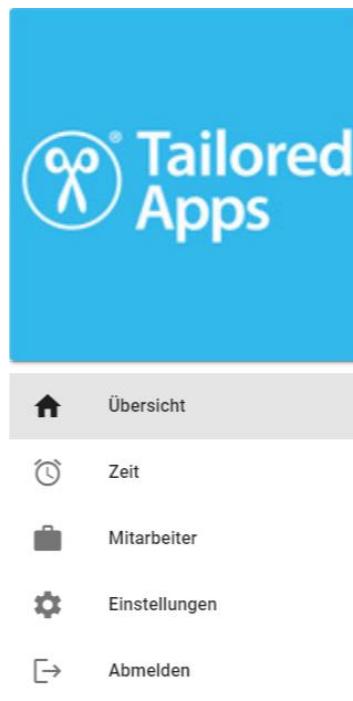


Abbildung 135: Sidebar - Admin

7.8.1.2 Sidebar – User

Bei einem User wird der Mitarbeiterabschnitt ausgeblendet und auch über eine Route mithilfe eines NavGuards geschützt.

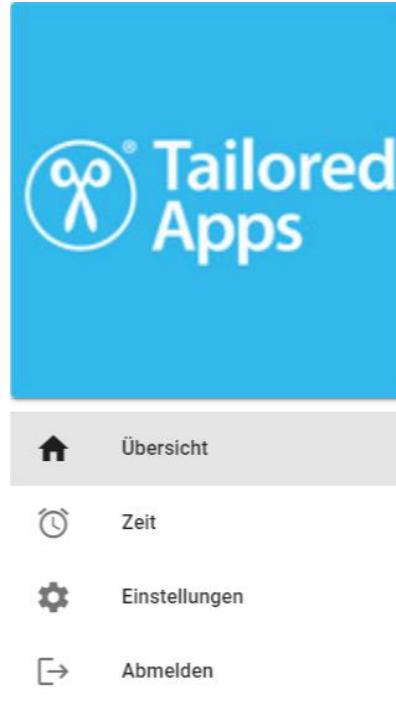


Abbildung 136: Sidebar - User

7.8.2 Login

Als Index der Applikation sieht man die Loginseite. Hier kann man sich mit seinem Redmine-Account authentifizieren.

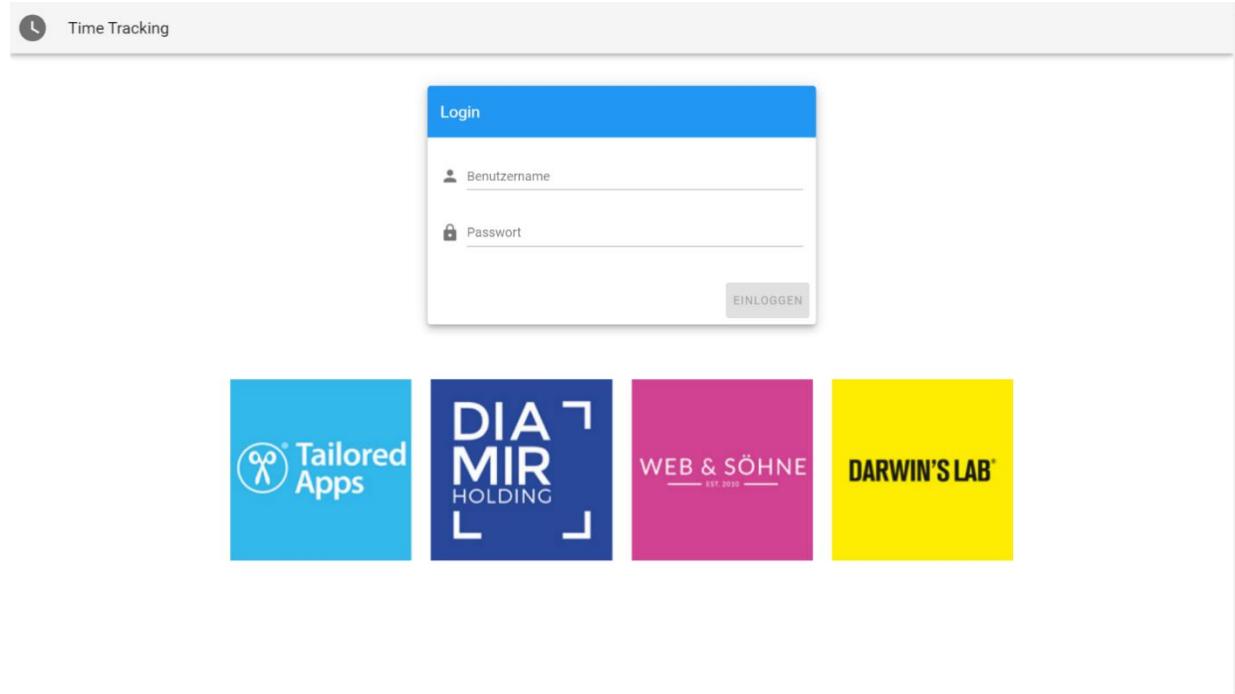


Abbildung 137: Loginseite

7.8.2.1 Laden

Nachdem man den Knopf „Einloggen“ gedrückt hat, oder beim letzten Eingabefeld „Enter“ betätigt, wird der User bei Redmine authentifiziert. Während des Authentifizierungsvorganges werden die Felder als ladend animiert.

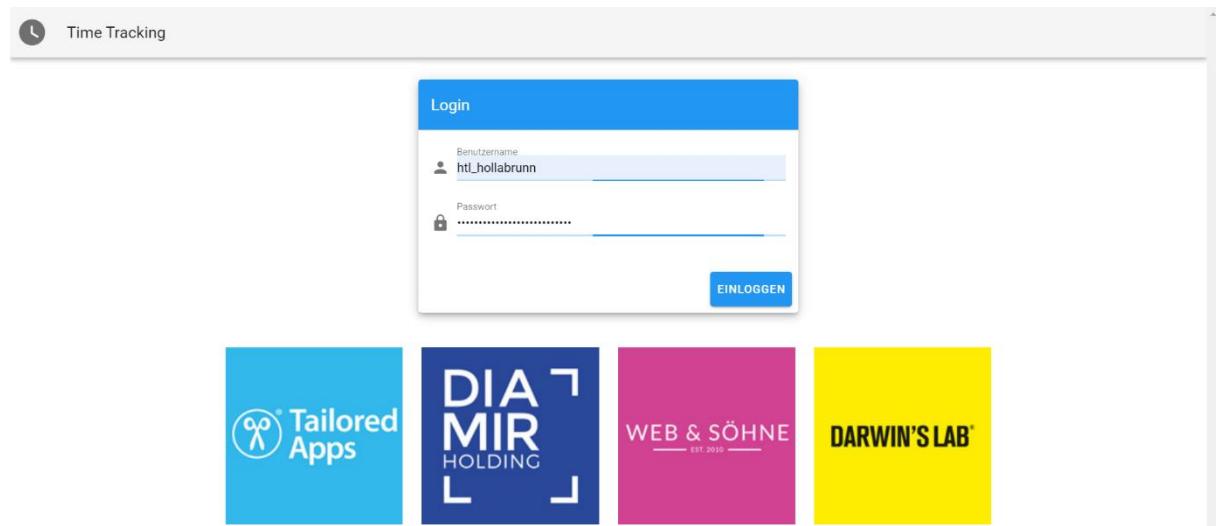


Abbildung 138: Loginseite - ladend

7.8.2.2 Fehler

Wird eine fehlerhafter Benutzername oder ein falsches Passwort eingegeben, so werden die Eingabefelder rot markiert und eine entsprechende Nachricht angezeigt.

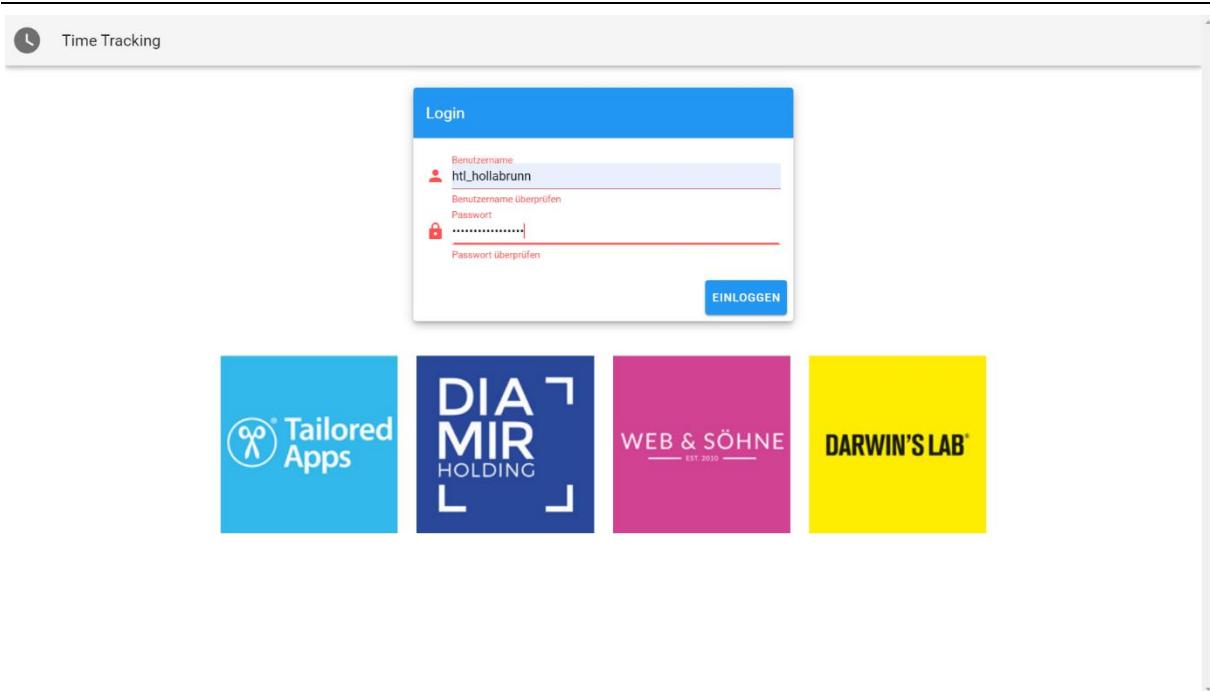
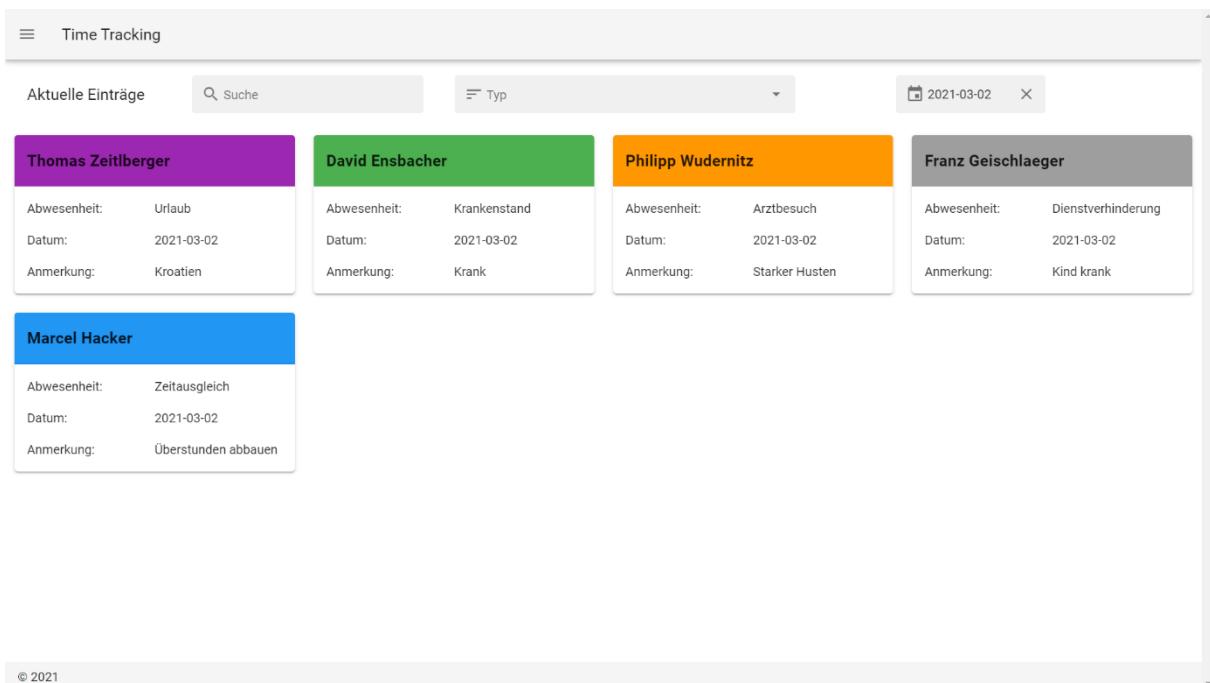


Abbildung 139: Loginseite - Fehlerfall

7.8.3 Übersicht

Nach dem Einloggen wird man auf die Übersicht verwiesen. Hier sieht jeder User und Admin die heutigen Abwesenheitseinträge.

7.8.3.1 Übersicht - Admin



The screenshot shows the 'Übersicht' (Overview) page for an Admin. At the top, there are search and filter options: 'Aktuelle Einträge', 'Suche' (Search), 'Typ' (Type), and a date filter '2021-03-02'. Below this, five user entries are listed in colored boxes:

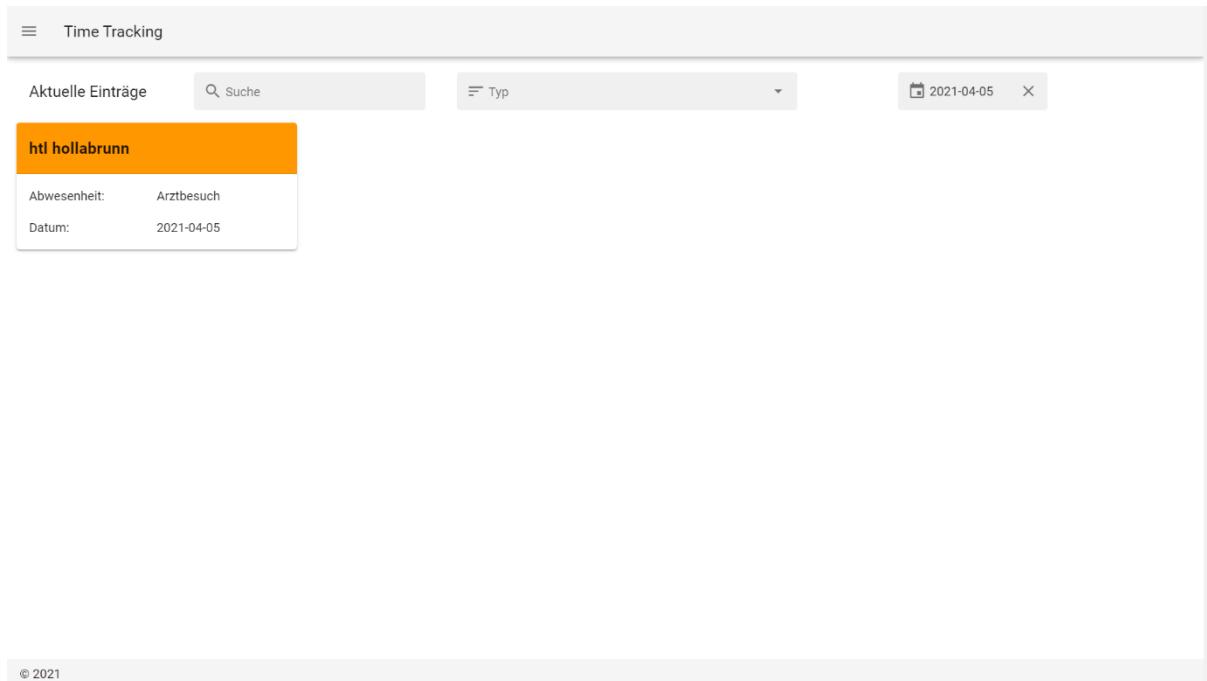
- Thomas Zeitlberger** (Purple): Abwesenheit: Urlaub, Datum: 2021-03-02, Anmerkung: Kroatien
- David Ensbacher** (Green): Abwesenheit: Krankenstand, Datum: 2021-03-02, Anmerkung: Krank
- Philipp Wudernitz** (Orange): Abwesenheit: Arztbesuch, Datum: 2021-03-02, Anmerkung: Starker Husten
- Franz Geischlaeger** (Grey): Abwesenheit: Dienstverhinderung, Datum: 2021-03-02, Anmerkung: Kind krank
- Marcel Hacker** (Blue): Abwesenheit: Zeitausgleich, Datum: 2021-03-02, Anmerkung: Überstunden abbauen

At the bottom left, a copyright notice reads '© 2021'.

Abbildung 140: Übersichtsseite - Admin

7.8.3.2 Übersicht – User

Bei einem User wird der Anmerkungsabschnitt ausgeblendet.

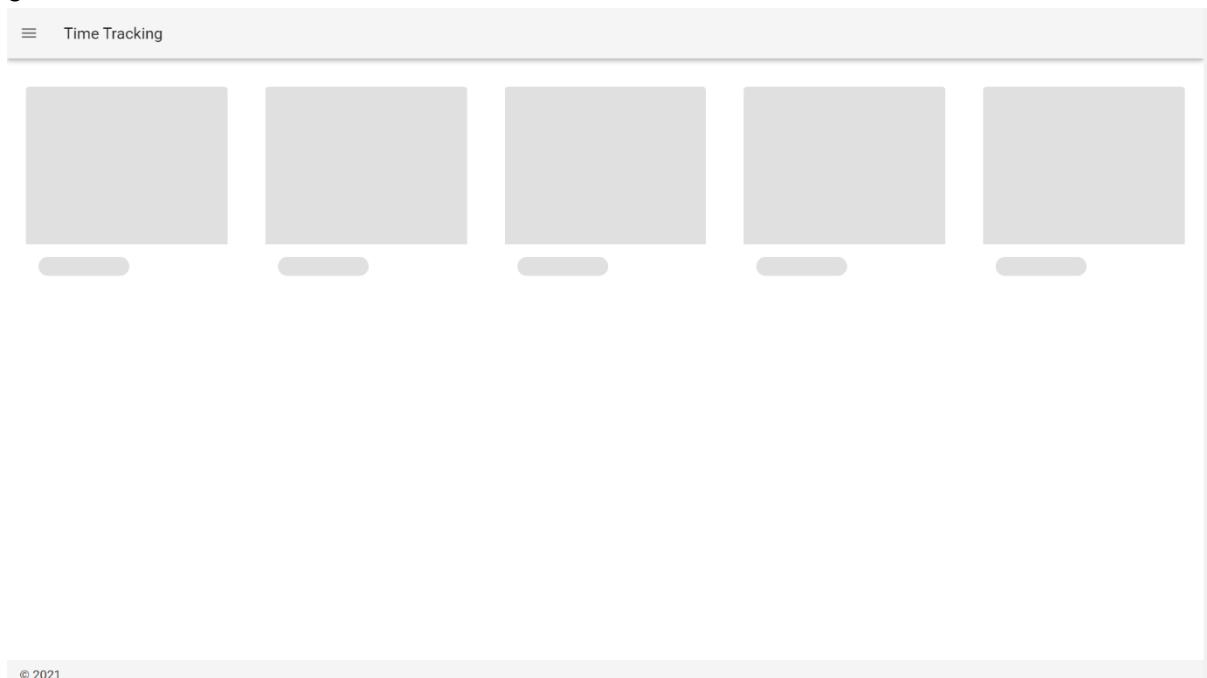


The screenshot shows the 'Time Tracking' application interface. At the top, there is a navigation bar with a menu icon, the text 'Time Tracking', and several search/filter options: 'Aktuelle Einträge', a search input field with placeholder 'Suche', a dropdown for 'Typ', and a date selector set to '2021-04-05'. Below the header, a large orange box displays the user information: 'htl hollabrunn'. Underneath this, two details are shown: 'Abwesenheit: Arztbesuch' and 'Datum: 2021-04-05'. The main content area below the header is currently empty, indicating no entries for the selected date. The bottom of the screen features a footer bar with the text '© 2021'.

Abbildung 141: Übersichtsseite - User

7.8.3.3 Laden

Wird gerade ein http-Request getätigt, so wird ein Loader vor der eigentlichen Komponente geschalten.



The screenshot shows the 'Time Tracking' application interface in a loading state. The top navigation bar is visible, but the main content area is filled with five large, light-gray rectangular placeholders, each with a small horizontal progress bar at the bottom, indicating that data is being loaded from the server. The bottom of the screen features a footer bar with the text '© 2021'.

Abbildung 142: Übersichtsseite - ladend



7.8.3.4 Fehler

Ist ein Fehler bei ein http-Request aufgetreten, wird eine Fehlermeldung angezeigt.

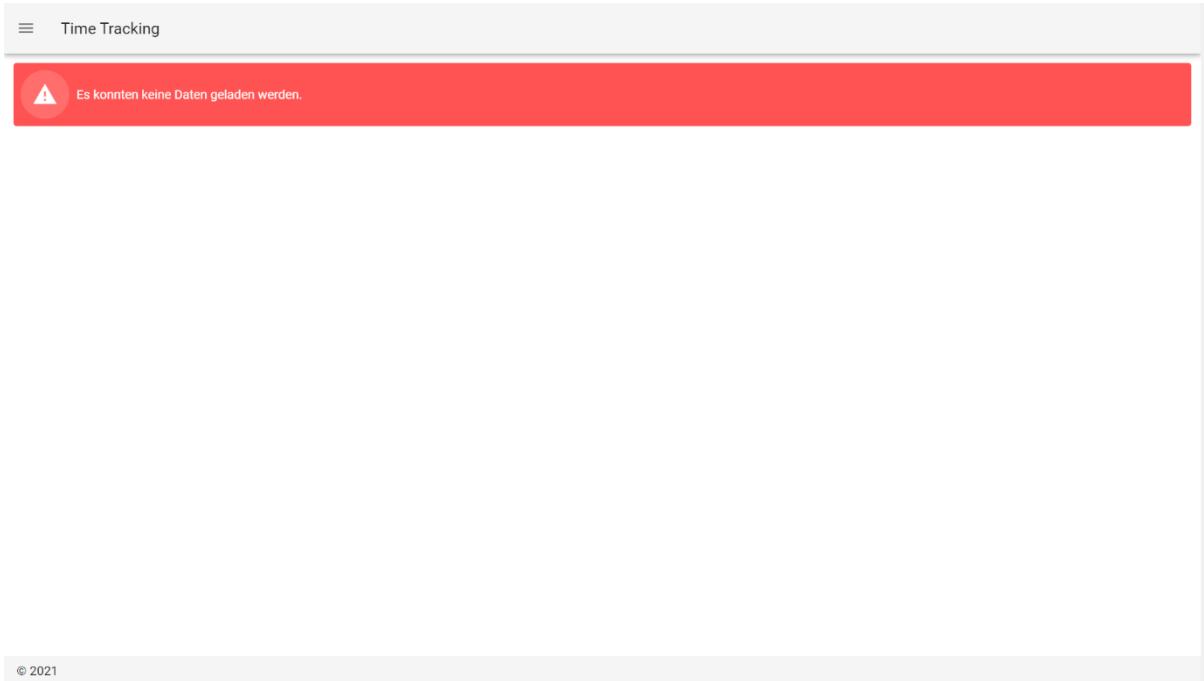


Abbildung 143: Übersichtsseite - Fehlerfall

7.8.4 Zeit

Jeder User und Admin besitzt seinen eigenen Kalender. Man kann Einträge erstellen, hinzufügen und bearbeiten.

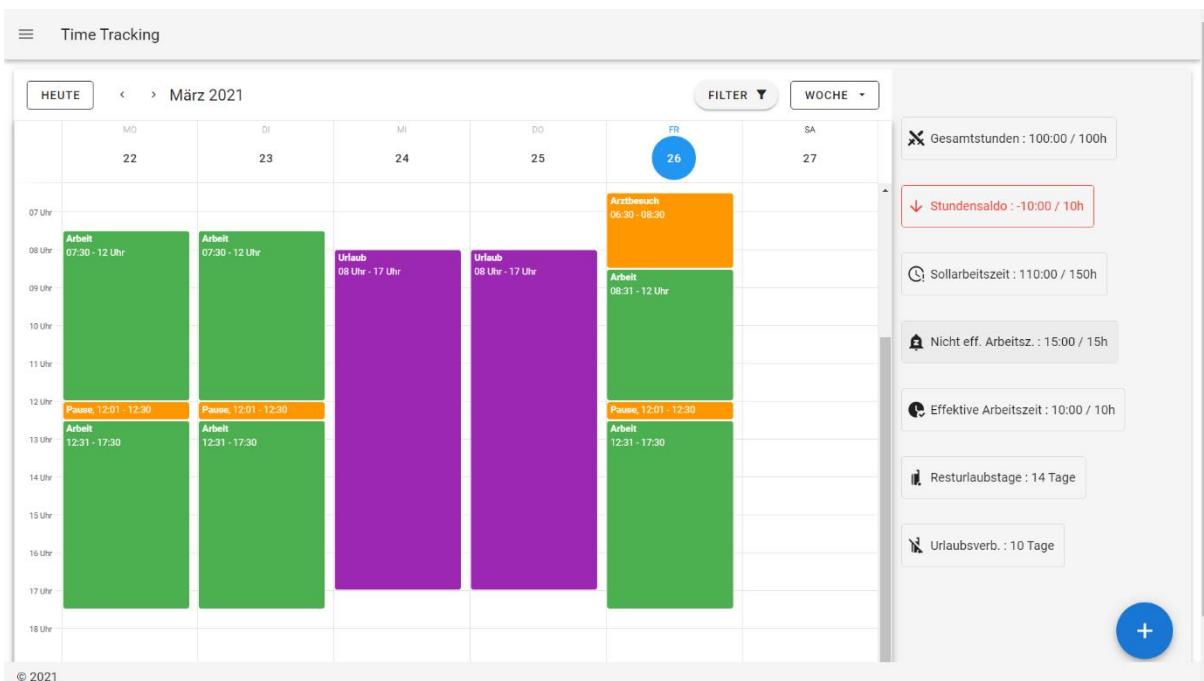


Abbildung 144: Kalenderseite

7.8.4.1 Laden

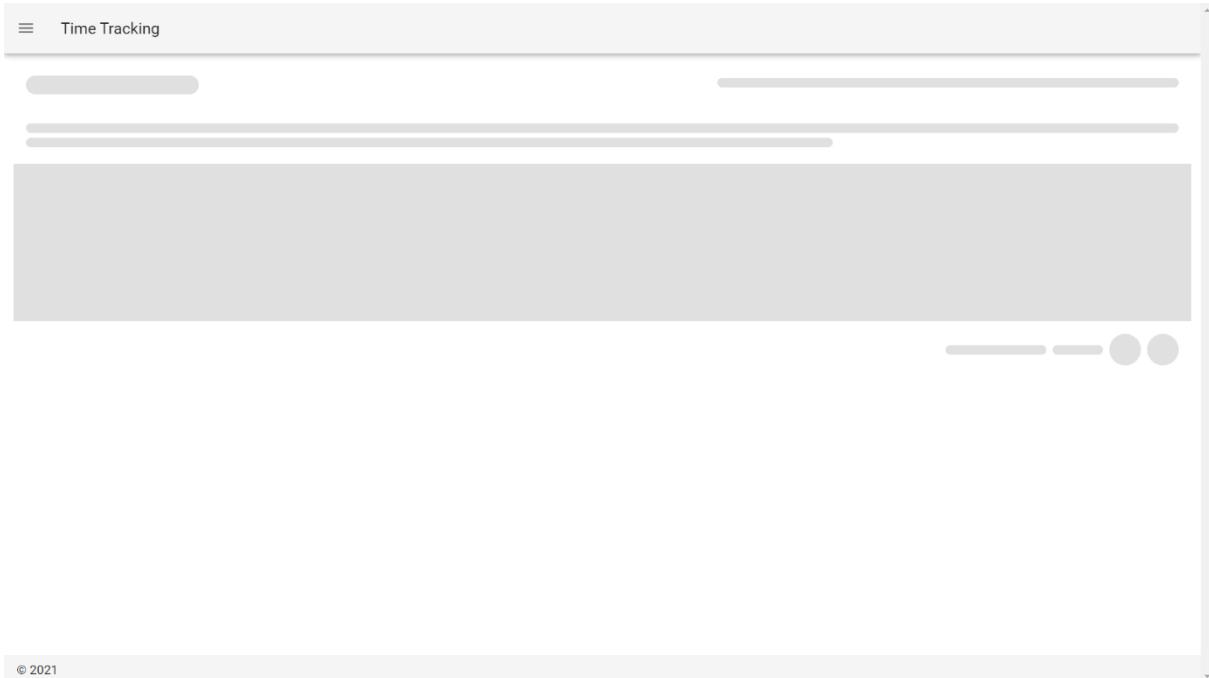


Abbildung 145: Kalenderseite - ladend

7.8.4.2 Fehler

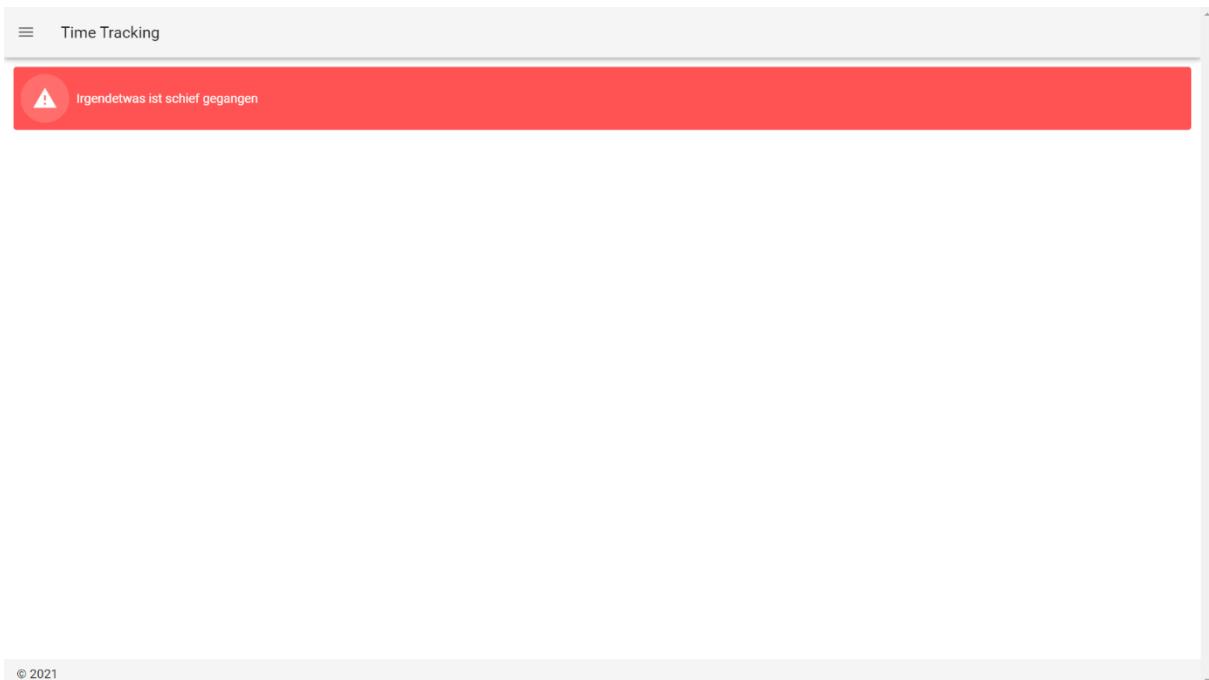
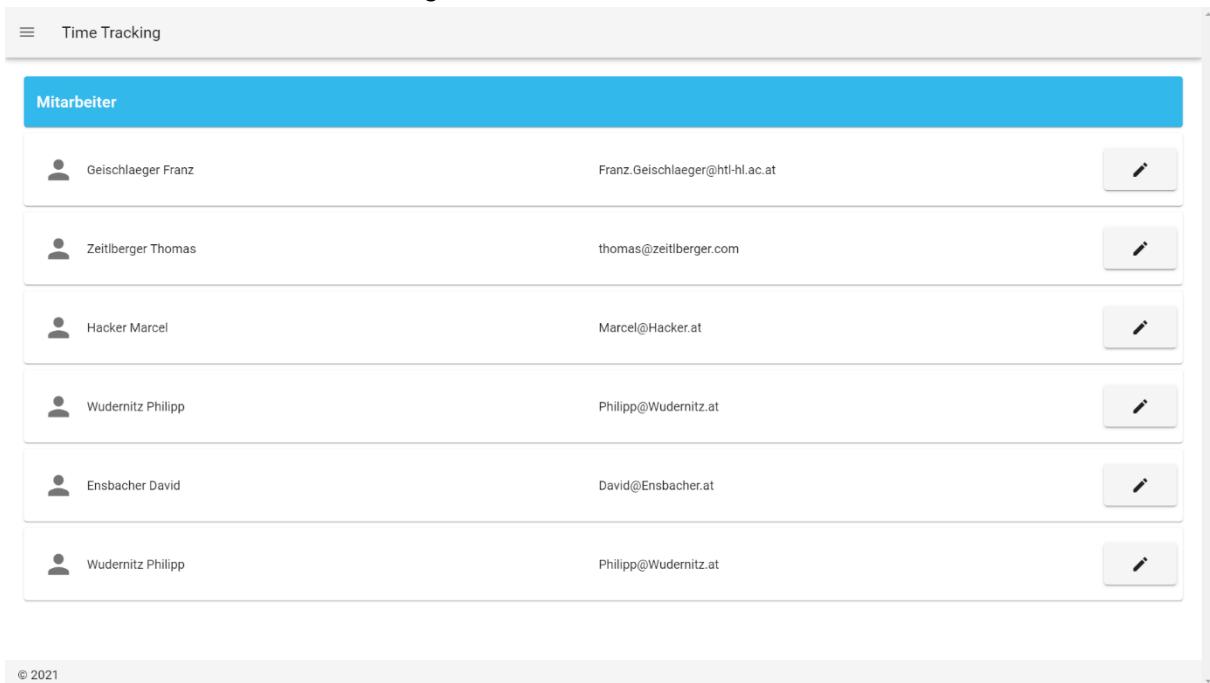


Abbildung 146: Kalenderseite - Fehlerfall

7.8.5 Mitarbeiter

Admins können auch einen Blick auf den Mitarbeiterabschnitt werfen. Hier kann man den jeweiligen Zeitkalender einsehen und ihn auch bearbeiten. Darüber hinaus können die Arbeitszeiten eines Mitarbeiters geändert werden.

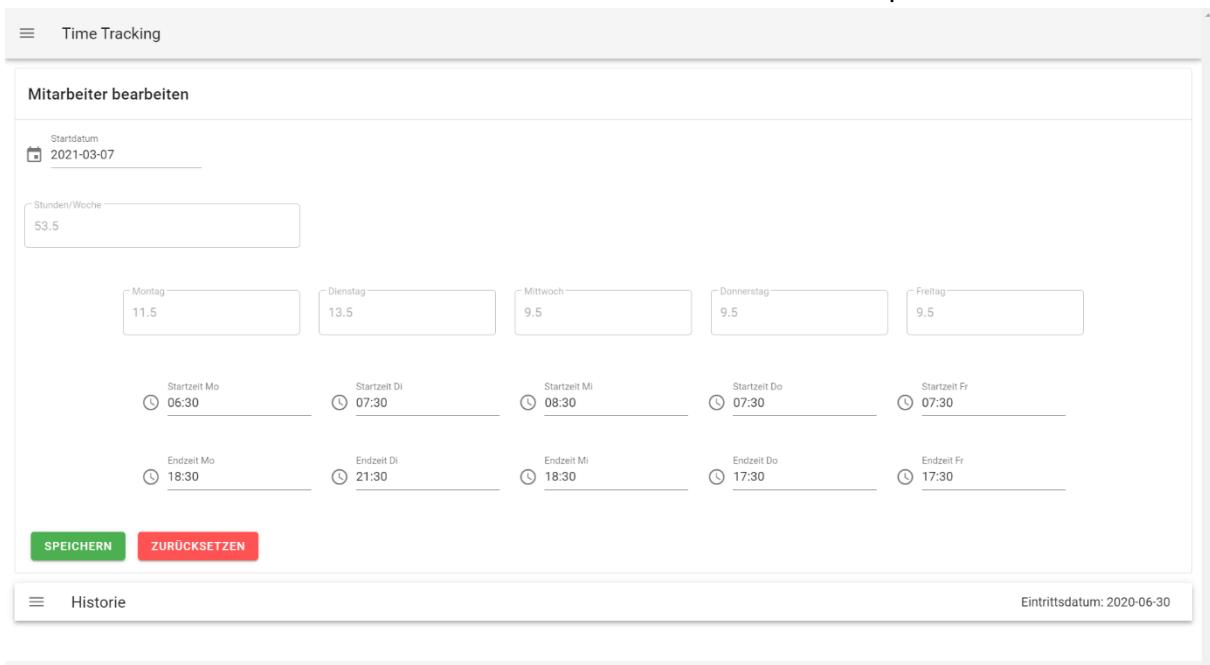


The screenshot shows a list of employees under the heading 'Mitarbeiter'. Each entry includes a profile icon, the employee's name, their email address, and an edit icon (pencil symbol). The employees listed are: Geischlaeger Franz, Zeitberger Thomas, Hacker Marcel, Wudernitz Philipp, Ensbacher David, and Wudernitz Philipp (repeated). At the bottom left, there is a copyright notice: © 2021.

Abbildung 147: Mitarbeiterseite

7.8.5.1 Bearbeiten

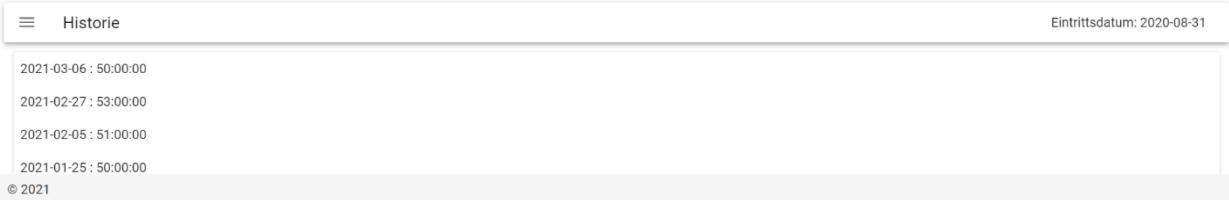
Dem Admin ist es auch erlaubt die Daten eines Mitarbeiters zu manipulieren.



The screenshot shows the 'Mitarbeiter bearbeiten' (Employee Edit) page. It includes fields for 'Startdatum' (2021-03-07), 'Stunden/Woche' (53.5), and five daily time blocks (Montag: 11.5, Dienstag: 13.5, Mittwoch: 9.5, Donnerstag: 9.5, Freitag: 9.5). Below these are start and end times for each day: Montag (06:30-18:30), Dienstag (07:30-21:30), Mittwoch (08:30-18:30), Donnerstag (07:30-17:30), and Freitag (07:30-17:30). At the bottom are 'SPEICHERN' (Save) and 'ZURÜCKSETZEN' (Reset) buttons. A history section at the bottom right shows an entry from 2020-06-30.

Abbildung 148: Mitarbeiter - bearbeiten

Historie:



The screenshot shows a list of historical entries on the left, each consisting of a date and time range. On the right, there is a timestamp indicating the entry was made on 2020-08-31.

Datum	Zeitraum
2021-03-06	50:00:00
2021-02-27	53:00:00
2021-02-05	51:00:00
2021-01-25	50:00:00

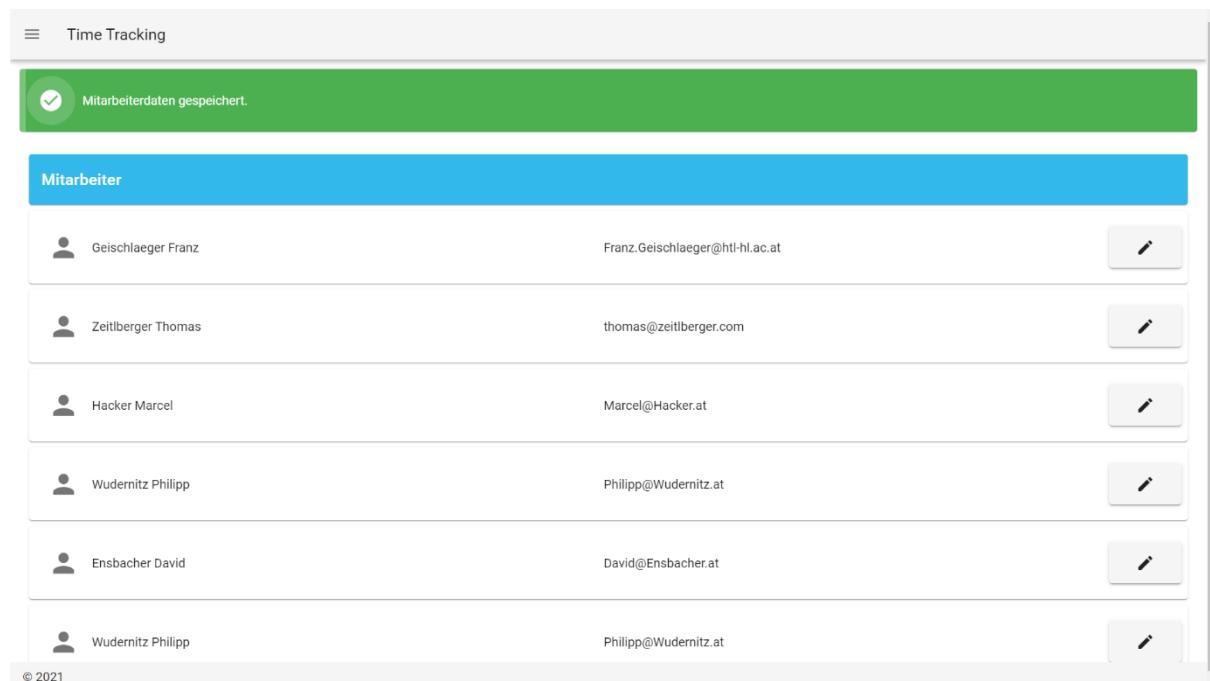
© 2021

Abbildung 149: Mitarbeiters - Historie

In der Historie wird links das Startdatum und rechts die Stunden/Woche angezeigt.

7.8.5.2 Erfolg – Mitarbeiter bearbeiten

Konnten die Mitarbeiterdaten erfolgreich geändert werden, wird ein Success-Alert angezeigt.



The screenshot shows a list of employees on the left, each with a name and email address. On the right, there are edit icons for each entry. A green success bar at the top indicates that employee data has been saved.

Mitarbeiter	E-Mail	Aktion
Geischlaeger Franz	Franz.Geischlaeger@htl-hl.ac.at	
Zeitberger Thomas	thomas@zeitberger.com	
Hacker Marcel	Marcel@Hacker.at	
Wudernitz Philipp	Philipp@Wudernitz.at	
Ensbacher David	David@Ensbacher.at	
Wudernitz Philipp	Philipp@Wudernitz.at	

© 2021

Abbildung 150: Mitarbeiterseite - gespeichert



7.8.5.3 Laden

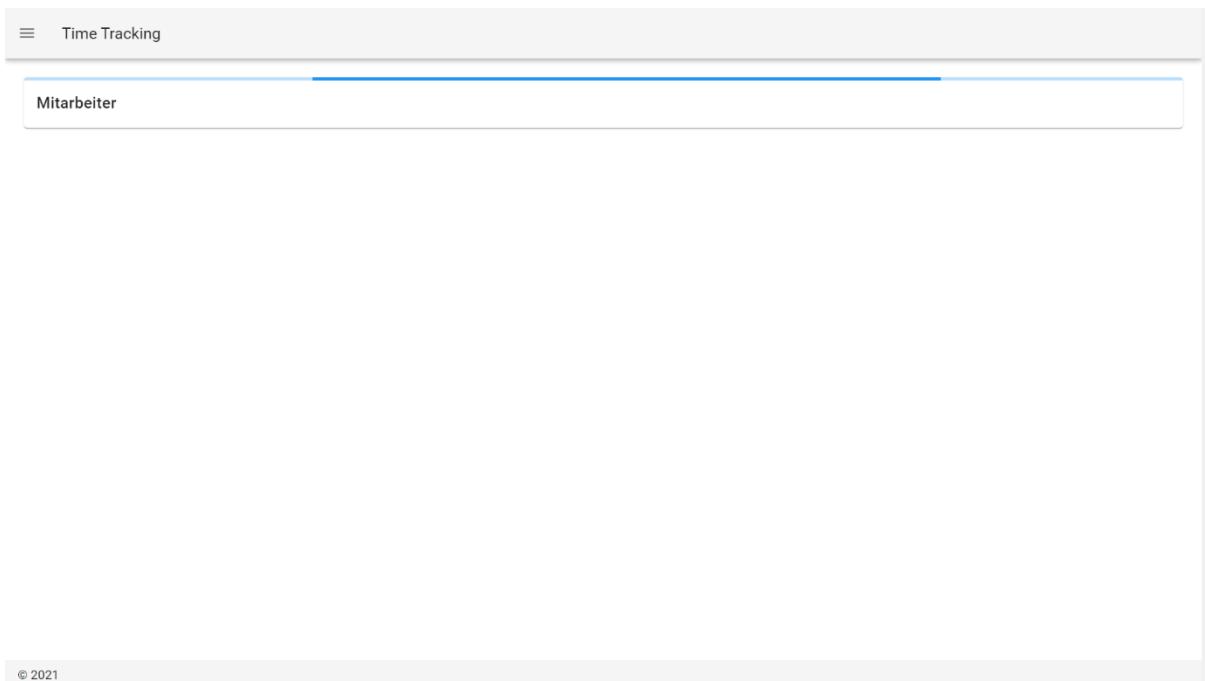


Abbildung 151: Mitarbeiterseite - ladend

7.8.5.4 Fehler

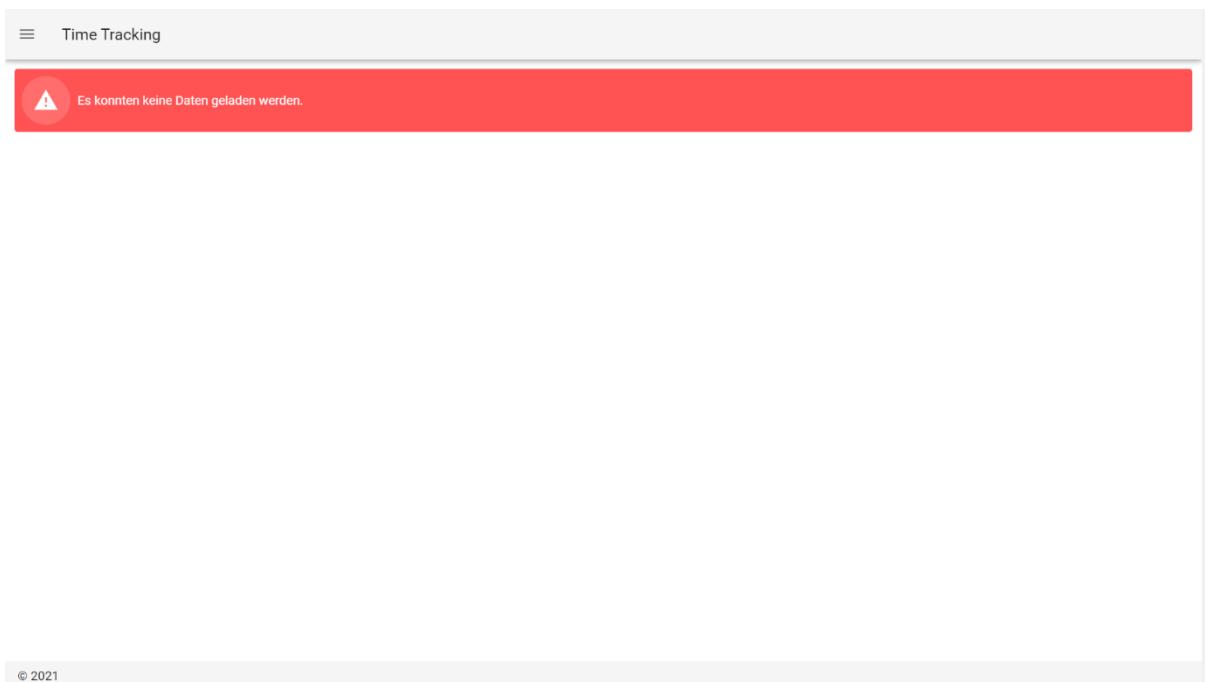


Abbildung 152: Mitarbeiterseite – Fehlerfall

7.8.6 Einstellungen

In den Einstellungen kann man das Theme der Applikation auswählen. Es ist auch möglich die Sprache zu verändern.

7.8.6.1 Einstellungen – Admin

Hier kann sich ein Admin einen Report erstellen lassen.

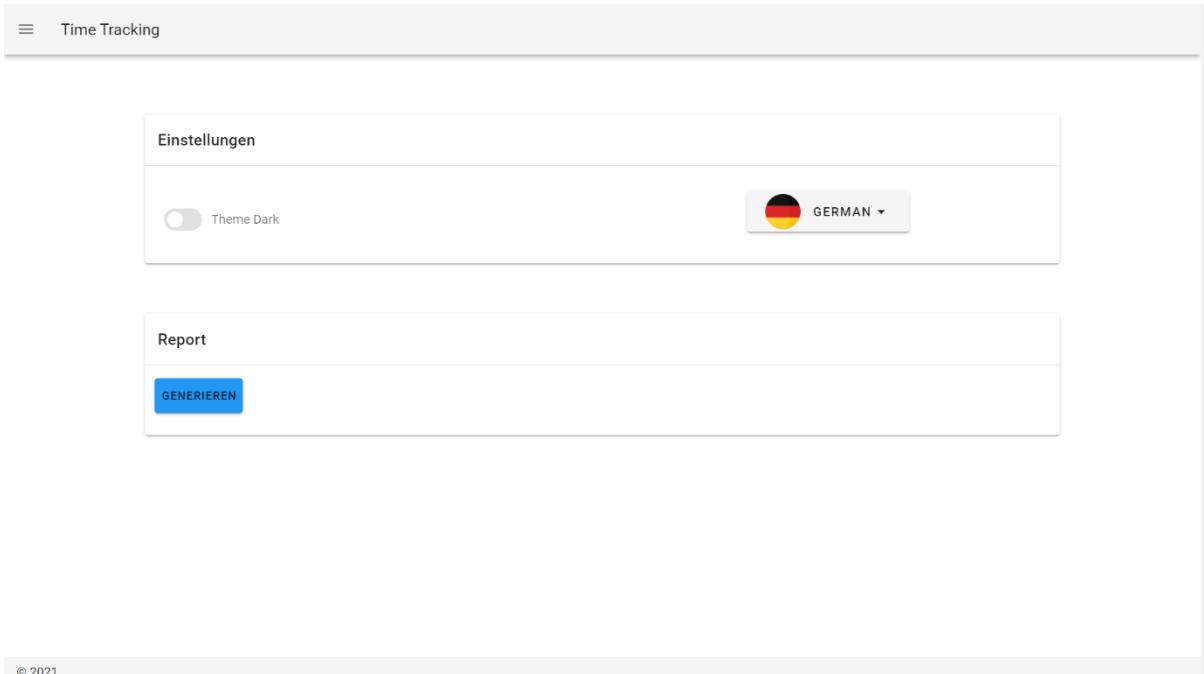


Abbildung 153: Einstellungsseite - Admin

7.8.6.2 Einstellungen – User

Bei einem User wird der Report-Abschnitt ausgeblendet.

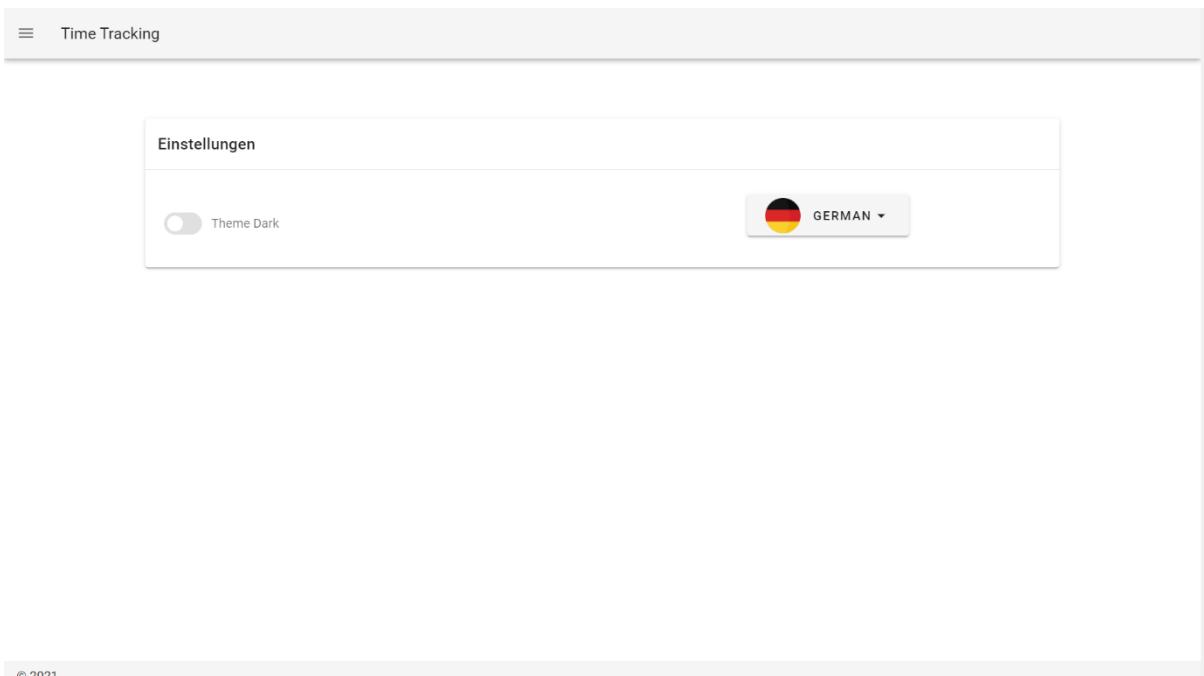


Abbildung 154: Einstellungsseite - User



7.8.6.3 Sprache

Die Sprache der App kann in Deutsch oder Englisch eingestellt werden.

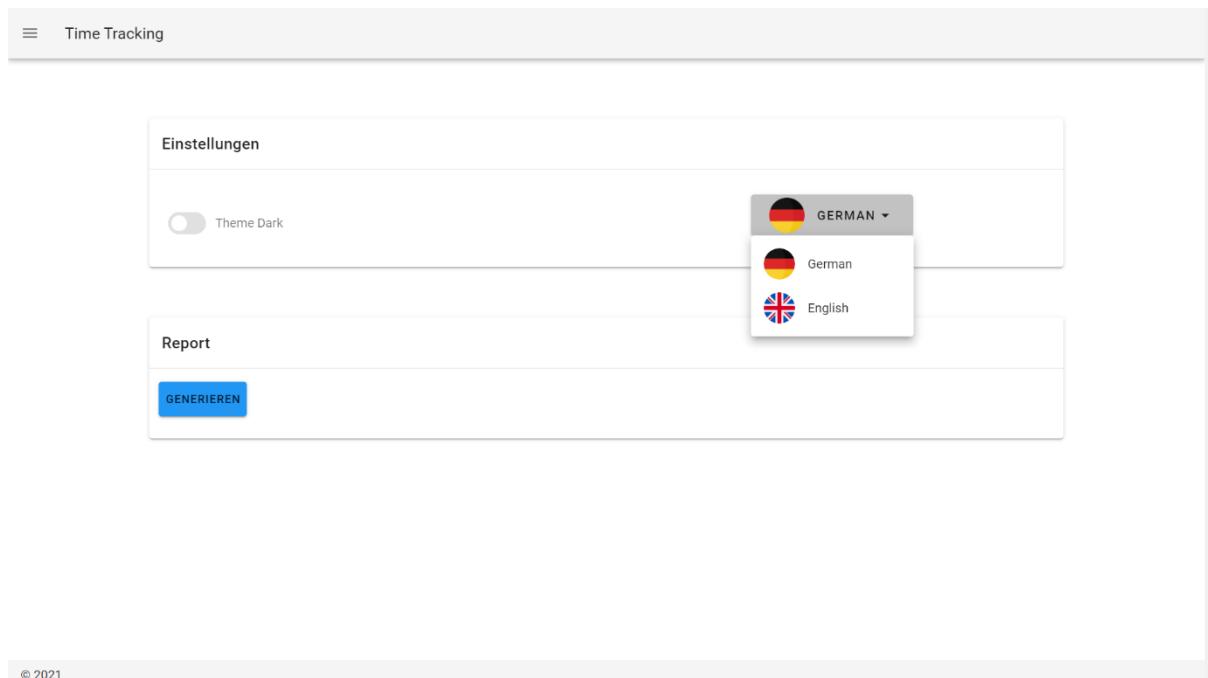


Abbildung 155: Einstellungsseite - Sprache auswählen

7.8.6.4 Report

Klickt man auf den Report-Knopf kann man das Jahr und den Monat des Reports bestimmen. Es wird folgendes Fenster angezeigt:

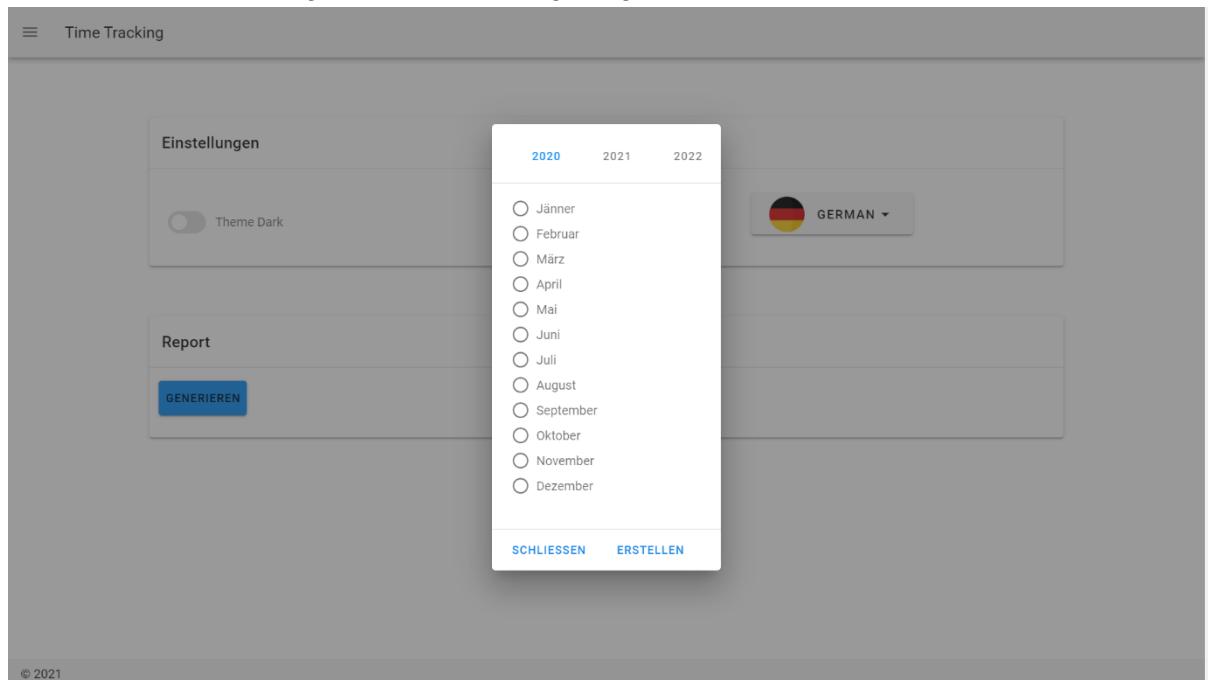


Abbildung 156: Einstellungsseite - Report erstellen

7.8.6.5 Fehler

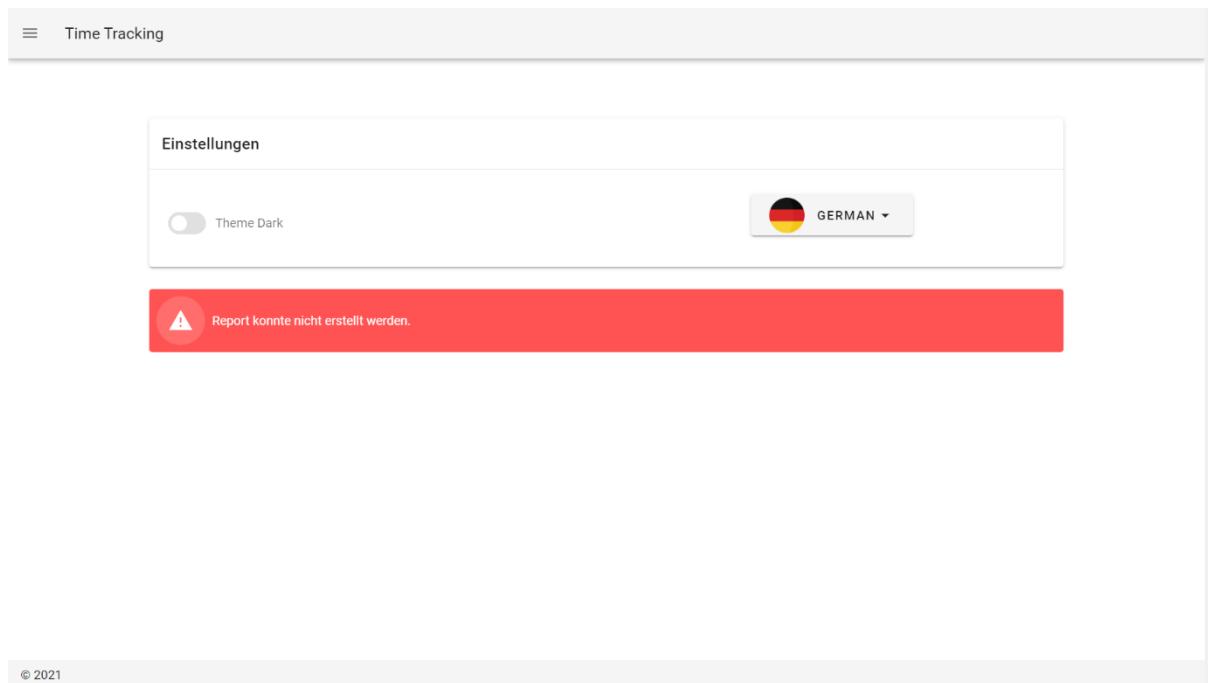


Abbildung 157: Einstellungsseite - Report Fehlerfall

7.9 Funktionsbeschreibung der App-Komponenten

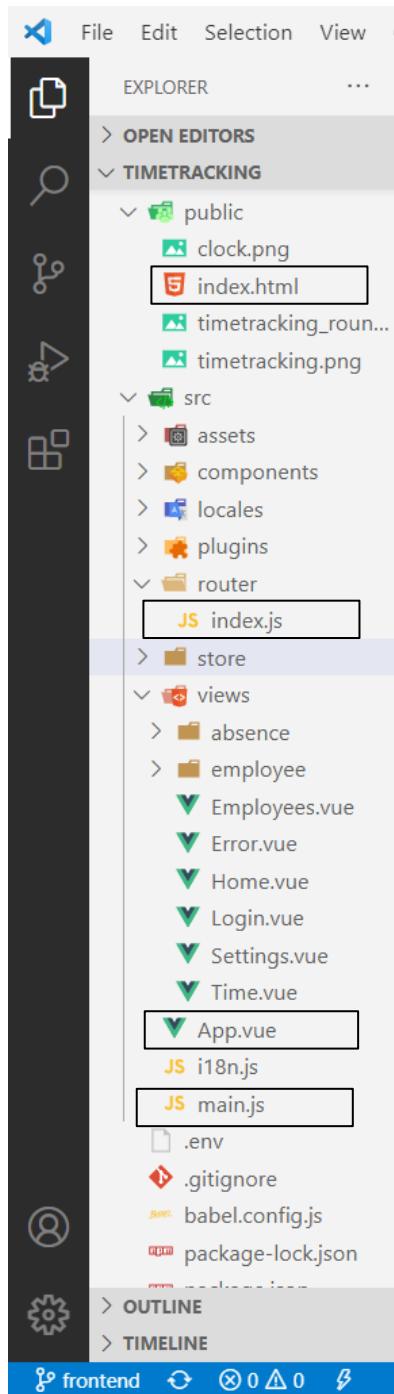


Abbildung 158: Ordnerverzeichnis 1

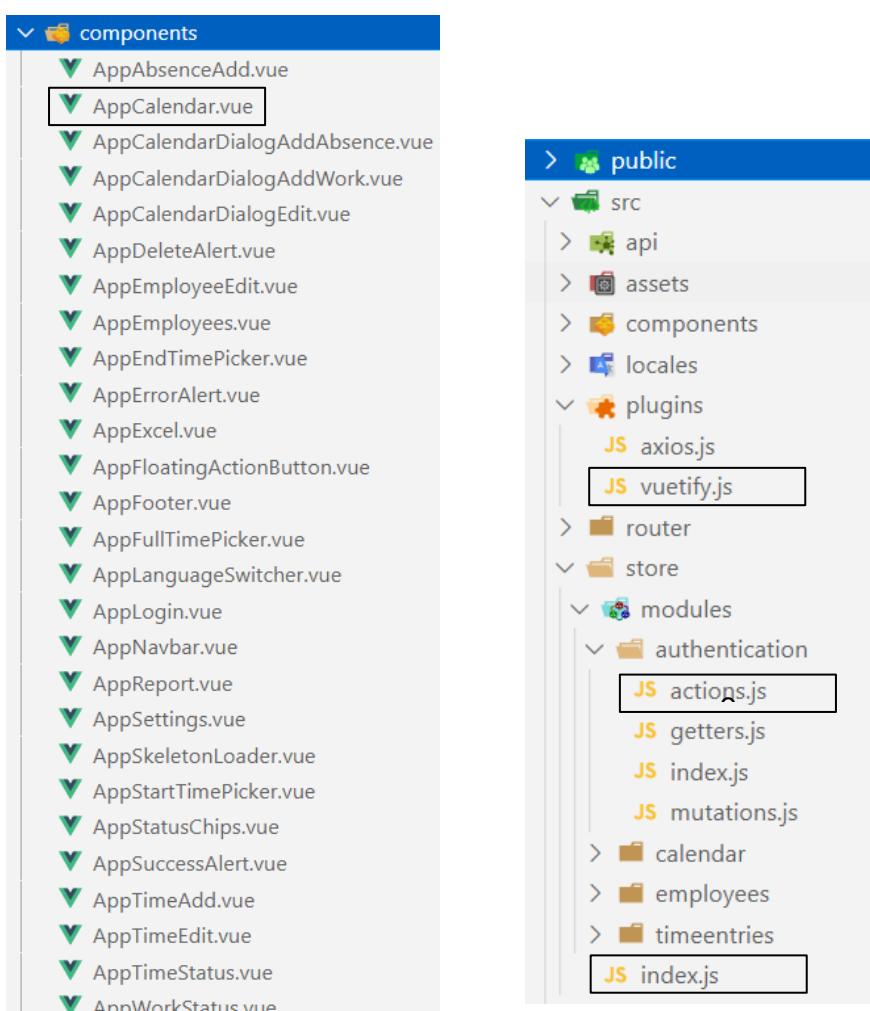


Abbildung 160: Ordnerverzeichnis 3

Abbildung 159: Ordnerverzeichnis 2

7.9.1 index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta http-equiv="Referer-Policy" content="no-referrer" />
    <meta http-equiv="Access-Control-Allow-Origin" content="*" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <link rel="icon" alt="TimeTracking logo" href="./timetracking_roun
d.png" />
    <title>Time Tracking</title>
    <link
      rel="stylesheet"
      href="https://fonts.googleapis.com/css?family=Roboto:100,300,4
00,500,700,900"
    />
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/@mdi/font@latest/css/materi
aldesignicons.min.css"
    />
    <link
      href="https://fonts.googleapis.com/css?family=Material+Icons"
      rel="stylesheet"
    />
  </head>
  <body>
    <noscript>
      <strong>
        We're sorry but <%= htmlWebpackPlugin.options.title %> does
        n't work
        properly without JavaScript enabled. Please enable it to
        continue.</strong>
      </strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

Listing 116: index.html-File



7.9.2 Mainskript – main.js

```
import Vue from "vue";
import "./plugins/axios"; // http library
import App from "./App.vue";
import router from "./router"; // vue router
import store from "./store"; // vuex store
import vuetify from "./plugins/vuetify"; // ui framework
import i18n from "./i18n"; // internationalisation

Vue.config.productionTip = true;

new Vue({
  router,
  store,
  vuetify,
  i18n,
  render: (h) => h(App),
}).$mount("#app");
```

Listing 117: main.js-File

7.9.3 Style-Konfiguration – vuetify.js

```
import Vue from "vue";
import Vuetify from "vuetify/lib";

Vue.use(Vuetify);

export default new Vuetify({
  // theme
  theme: {
    themes: {
      light: {
        primary: "#2196F3",
        secondary: "#424242",
        accent: "#82B1FF",
        error: "#FF5252",
        info: "#2196F3",
        success: "#4CAF50",
        warning: "#FFC107",
      },
      dark: {
        dark: {
          primary: "#1976D2",
          secondary: "#1976D2",
        },
      },
    },
  },
});
```

```

        },
        },
        },
        // icons
        icons: {
            iconfont: "mdi",
            values: {
                add: "mdi-add",
                work: "mdi-work",
                settings: "mdi-settings",
                logout: "mdi-exit-to-app",
                chevron_right: "mdi-chevron-right",
                chevron_left: "mdi-chevron-left",
                person: "mdi-person",
                delete: "mdi-delete",
            },
        },
    });
}
);

```

Listing 118: vuetify.js-File

7.9.4 Router – router / index.js

```

import Vue from "vue";
import VueRouter from "vue-router";
import Home from "../views/Home.vue";
import Time from "../views/Time.vue";
import Employees from "../views/Employees.vue";
import Settings from "../views/Settings.vue";

Vue.use(VueRouter);

const routes = [
{
    // Unknown routes
    path: "*",
    name: "Error",
    component: () =>
        import(/* webpackChunkName: "error" */ "../views/Error.vue")
},
{
    // Login
    path: "/",
    name: "Login",
    component: () =>
        import(/* webpackChunkName: "login" */ "../views/Login.vue")
},
{

```



```
// Fallback for login
path: "/login",
redirect: { name: "Login" }
},
{
// Dashboard
path: "/home",
name: "Home",
component: Home,
meta: { requiresAuth: true }
},
{
// Fallback for dashboard
path: "/dashboard",
redirect: { name: "Home" }
},
{
// Fallback for dashboard
path: "/time",
redirect: { name: "Login" }
},
{
// Time Calendar for myself
path: "/time/:userId",
name: "Time",
component: Time,
meta: { requiresAuth: true }
},
{
// employee list
path: "/employees",
name: "Employees",
component: Employees,
meta: { requiresAuth: true },
children: [
{
// employee edit
path: "edit/:userId",
name: "EmployeeEdit",
component: () =>
import(
/* webpackChunkName: "employeedit" */ "../views/employee/Edit.vue"
),
meta: { requiresAuth: true },
/* Only for Admins
```

```
beforeEnter: (to, from, next) => {
    if (!localStorage.getItem("isUserAdmin"))
        console.log("Adminroute rejected..."), next({ name: "Home" });
    else next();
}
],
/* Only for admins
beforeEnter: (to, from, next) => {
    if (!localStorage.getItem("isUserAdmin"))
        console.log("Adminroute rejected..."), next({ name: "Home" });
} ;
else next();
}
},
{
// Settings
path: "/settings",
name: "Settings",
component: Settings,
meta: { requiresAuth: true }
},
{
// Logout
path: "/logout",
name: "Logout",
redirect: { name: "Login" }
}
];
// Setting up router
const router = new VueRouter({
routes
});

/* Pre route guards
router.beforeEach((to, from, next) => {
// User must be authenticated for route
console.log("Authentifiziert: " + localStorage.getItem("isAuthenticated"));
if (to.meta.requiresAuth && !localStorage.getItem("isAuthenticated")) {
    console.log("Route rejected...");
    next({ name: "Login" });
} else {
    next();
}
```



```
}

});

export default router;
```

Listing 119: index.js-Routerfile

7.9.5 Maintemplate – App.vue

```
<template>
<div>
  <v-app id="inspire">
    <v-slide-y-transition mode="out-in" leave-absolute>
      <router-view></router-view>
    </v-slide-y-transition>
  </v-app>
</div>
</template>

<script>
export default {
  name: "App",
  created() {
    this.$store.dispatch("tryLogin"); // auto login after reload
  },
};
</script>
```

Listing 120: App.vue-Komponente

7.9.6 Vuex Store – store.js

```
import Vue from "vue";
import Vuex from "vuex";
import { authentication } from "@/store/modules/authentication/index.js";
import { calendar } from "@/store/modules/calendar/index.js";
import { employees } from "@/store/modules/employees/index.js";
import { timeentries } from "@/store/modules/timeentries/index.js";

Vue.use(Vuex);

export default new Vuex.Store({
  modules: {
    authentication,
```

```

        calendar,
        employees,
        timeentries
    }
}) ;

```

Listing 121: store.js-File

7.9.7 Authentifizierungs-Skript - authentication/actions.js

Das folgende Skript ist für die Authentifizierung mit Redmine und dem Backend verantwortlich. Zudem wurde ein Autologout eingebaut, die Dauer dieses Auslogtimers wird durch die Konstante expiresIn festgelegt. Der API-Key von Redmine wird im lokalen Browserspeicher gelagert. Damit kann sich der User bei einem Aktualisieren der Seite zeiteffizient bei Redmine authentifizieren.

```

import axios from "axios";

const expiresIn = 3600000; // logout time

const authenticationURL = "https://redmine.tailored-
apps.com"; // Redmine
const verificationURL = "https://wudi.serveminecraft.net:8080"; // Backend API

// actions
export const actions = {
    async authenticateUser({ commit, dispatch, state }) {
        commit("setLoadingAuth", true);
        console.log("Username: " + state.userCredentials.username);
        console.log("Password: " + state.userCredentials.password);

        // important use { object } for objects
        console.log(`Authenticating on: ${authenticationURL}/my/account.json`);
        axios
            .get(`${authenticationURL}/my/account.json`, {
                auth: {
                    username: state.userCredentials.username,
                    password: state.userCredentials.password,
                },
                headers: { "Access-Control-Allow-
Origin": "*" }, // Allow request to site
            })
            .then((response) => {
                commit("setAuthentication", true);
                commit("setUserData", response.data.user);

                //todo make this faster
            })
    }
}

```



```
dispatch("storeUserData"); /* Stores User data local

dispatch("verifyUser");
console.log(
    "%c User with credentials logged in",
    "background: #ff5722;"
);
console.log(response);
})

.catch((error) => {
    commit("setErrorAuth", true);
    console.error("Authenticating with credentials: " + error);
})
.finally(() => commit("setLoadingAuth", false)); // result
},

// Reauthenticate User when reloading with his api key
async authUserApiKey({ commit, dispatch }, api_key) {
    commit("setLoadingAuth", true);
    console.log("API: " + api_key);
    axios
        .get(`$authenticationURL}/my/account.json`, {
            auth: {
                key: api_key,
            },
            headers: {
                "Access-Control-Allow-Origin": "*",
                "X-Redmine-API-Key": api_key,
            },
        })
        .then((response) => {
            commit("setAuthentication", true); // user is authenticated
            commit("setUserData", response.data.user);
            dispatch("storeUserData"); /* Stores new User data local
            console.log(
                "%c User authenticated with API Key",
                "background: #ff5722;"
            );
            console.log(response);
        })
        .catch((error) => {
            commit("setErrorAuth", true);
            console.error("Authentication with API_key: " + error);
        })
        .finally(() => commit("setLoadingAuth", false));
},
```

```
// Handle reloads of the website
async tryLogin({ dispatch }) {
    const isAuthenticated = localStorage.getItem("isAuthenticated");
    const api_key = localStorage.getItem("api_key");
    const userId = localStorage.getItem("userId");

    // look for prior logins
    if (isAuthenticated && api_key && userId) {
        dispatch("authUserApiKey", api_key); // authenticate with api_
key

        setTimeout(() => {
            dispatch("logout");
            console.log(
                "%c You were logged out...",
                "background: #222; color: #bada55"
            );
            }, expiresIn);
    }
},
/* registeres the user in the backend
async registerUser({ commit, state }) {
    commit("setLoadingAuth", true);
    console.log("Registration on: " + `${verificationURL}/regUser`);

    // send userdata with backend password
    const object = {
        idUser: "608",
        firstname: "Marceldwad",
        lastname: state.userData.lastname,
        email: state.userData.mail,
        joined: state.userData.created_on.slice(0, 10),
        apikey: state.userData.api_key,
        pass:
            "example",
    };
    console.table("idUser: " + object.idUser);
    console.log("firstname: " + object.firstname);
    console.log("lastname: " + object.lastname);
    console.log("Email: " + object.email);
    console.log("joined: " + object.joined);

    axios
        .post(` ${verificationURL}/regUser`, object)
        .then((response) =>
            console.log("%c User registered by Backend", "background: #f

```

```

f5722;");

    console.log("Response:" + response.data);
}

.catch((error) => {
    commit("setErrorAuth", true);
    console.error("Registration on Backend: " + error);
})
.finally(() => commit("setLoadingAuth", false));
}

/* verifies the user in the backend
async verifyUser({ commit, dispatch, state }) {
    commit("setLoadingAuth", true);
    console.log("id User:" + state.userData.id);
    console.log("Verification on: " + `${verificationURL}/verifyUser`);
}

// send userdata with backend password
//state.userData.id
const object = {
    idUser: "608",
    pass:
        "example",
};

console.log("Sent data: " + object);
axios
    .post(` ${verificationURL}/verifyUser`, object)
    .then((response) => {
        console.log("%c User verified by Backend", "background: #ff5722;");

        console.log("Existing: " + response.data.existing);
        // is user already registered?
        if (response.data.existing == false) {
            // user unknown, register user
            dispatch("registerUser");
        }
    })
    .catch((error) => {
        commit("setErrorAuth", true);
        console.error("Verification on Backend: " + error);
    })
    .finally(() => commit("setLoadingAuth", false));
}

// Stores user information in local storage
async storeUserData({ commit }) {
    commit("storeAuthentication");
    commit("storeUserId");
}

```

```

        commit("storeAdminPrivilage");
        commit("storeApiKey");
    },
    // Deletes Data and storage
    async logout({ commit }) {
        commit("setUserCredentials", "null"); // deletes user data
        commit("setAuthentication", false); // deletes authentication state
        commit("clearLocalStorage"); // delete local store
        commit("setErrorAuth", false); // reset login form validation
    },
}

```

Listing 122: actions.js-Authentifizierung

7.9.8 Kalender-Skript – AppCalendar.vue

Das folgende Skript ist für die Anzeige und Filterung der Einträge am Kalender zuständig. Dafür werden die Events vom Vuex Store hergenommen und durch Verzweigungen geschickt. Aus den Datensätzen werden Einträge generiert, die der Kalender darstellen kann.

```

/* updates events in the calendar
updateRange({start, end}) {
    this.start = start.date;
    this.end = end.date;
    console.log("Woche start: " + start.date);
    console.log("Woche end: " + end.date);
    this.storedEvents = this.$store.getters.getCalendarEntries;

    var events = [];
    var storedEvents = this.storedEvents;
    var nameEvent = "";
    var startEvent = "";
    var endEvent = "";
    var date = "";
    var detailEvent = "";
    var colorEvent = "";
    var allDay = false;

    // Search all events
    for (let i = 0; i < storedEvents.length - 1; i++) {
        /* make sure the are not NULL
        // valid entries
        if (storedEvents[i].stamp && storedEvents[i].type) {
            /* is there an start stamp?
            // No stop, can create all entries without stop
            if (storedEvents[i].type != "Stop") {
                // no event filter

```



```
if (storedEvents[i].type == "Start") {
    // name event
    nameEvent = this.names[0]; // Arbeit
} else {
    nameEvent = storedEvents[i].type;
// Abwesenheit or Pause
    nameEvent = this.translateName(nameEvent);
// translate
}
startEvent = storedEvents[i].stamp;
// Start timestamp for event
detailEvent = storedEvents[i].comment;
// Comment for event
    // Search for an stop timestamp
    //? go to the new entry i + 1
    //! search entries with the same date
if (storedEvents[i + 1].date == storedEvents[i].date) {
    endEvent = storedEvents[i + 1].stamp;
    date = storedEvents[i].date;
    const id = storedEvents[i].idDaily;
    const idStop = storedEvents[i + 1].idDaily;
    const first = new Date(` ${date}T${startEvent}`);
    const second = new Date(` ${date}T${endEvent}`);
    colorEvent = this.colors[this.rnd(0, this.colors.length - 1)];
}

events.push({
    id: id, // id of the event
    idStop: idStop, // id of stop stamp
    name: nameEvent, // name of the event
    start: first, // start timestamp
    end: second, // end timestamp
    details: detailEvent, // comments
    color: colorEvent, // color
    timed: !allDay, // fullday
}) ;
}

} else {
    console.error("Error invalid entry");
}
}

this.bufferEvents = events; //? prepare for possible filter
this.events = this.bufferEvents;
this.updateFilter();
},
/* filters entries in calendar
updateFilter() {
```

```

var showEvents = [];

console.log("Filter updated:" + this.filter + "\n");
if (!this.filter) {
    this.events = this.bufferEvents;
    return;
}
if (this.filter == this.$t("dashboard.absence")) {
    // Filter by absence
    for (let i = 0; i < this.bufferEvents.length; i++) {
        // search all elements in buffer
        for (let grund = 0; grund < this.abwesenheitsgrund.length;
grund++) {
            // search for all absences
            if (this.bufferEvents[i].name == this.abwesenheitsgrund[
grund]) {
                showEvents.push(this.bufferEvents[i]);
                console.log("filter bei Absence: " + this.bufferEvents
[i].name);
            }
        }
    }
    console.table("Show: " + showEvents);
}

this.events = showEvents;

return;
} else if (this.filter == this.$t("time.work") ||
           this.filter == this.$t("time.pause")) {
    // Filter by pause or work
    for (let i = 0; i < this.bufferEvents.length; i++) {
        if (this.bufferEvents[i].name == this.filter) {
            console.log("filter bei arbeit oder Pause: " +
this.bufferEvents[i].name);
            showEvents.push(this.bufferEvents[i]);
        }
    }
    console.table("Show: " + showEvents);
    this.events = showEvents;

return;
} else { console.error("No matching filter..."); }
},
console.log("Delete Event id: " + id);
this.$store.dispatch("deleteCalendarEntry", id);
},

```

Listing 123: AppCalendar.vue-Skript

7.10 Views

7.10.1 Login

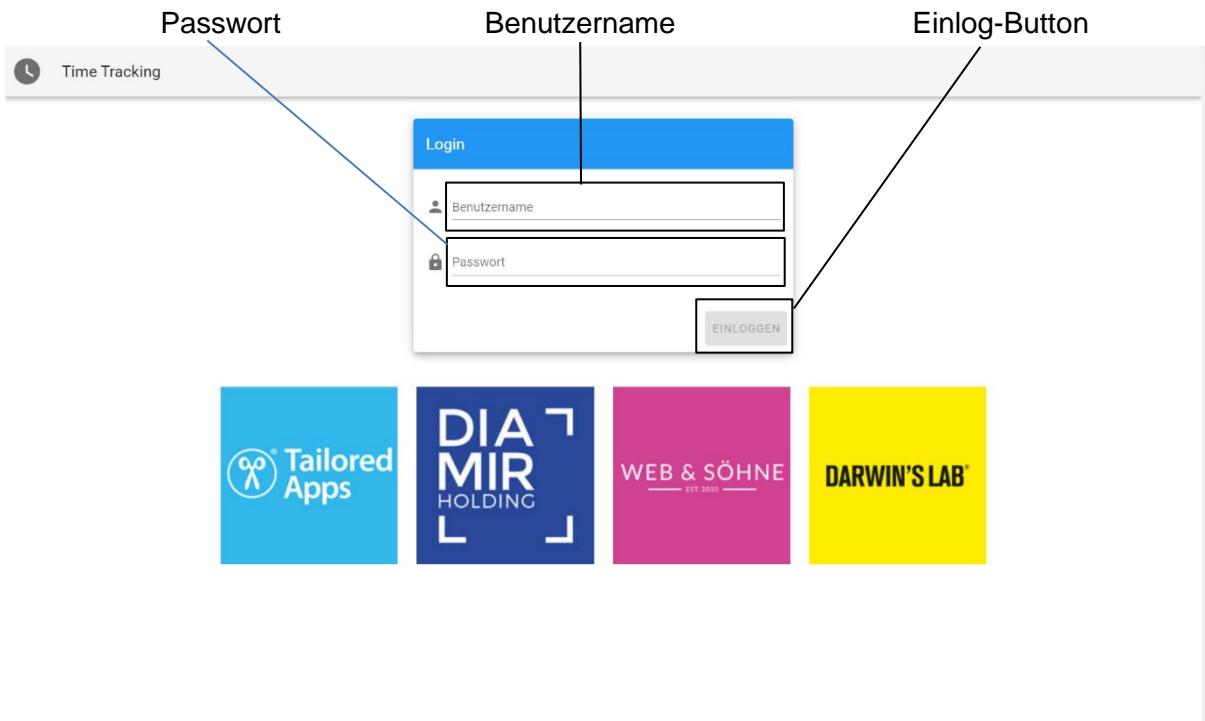
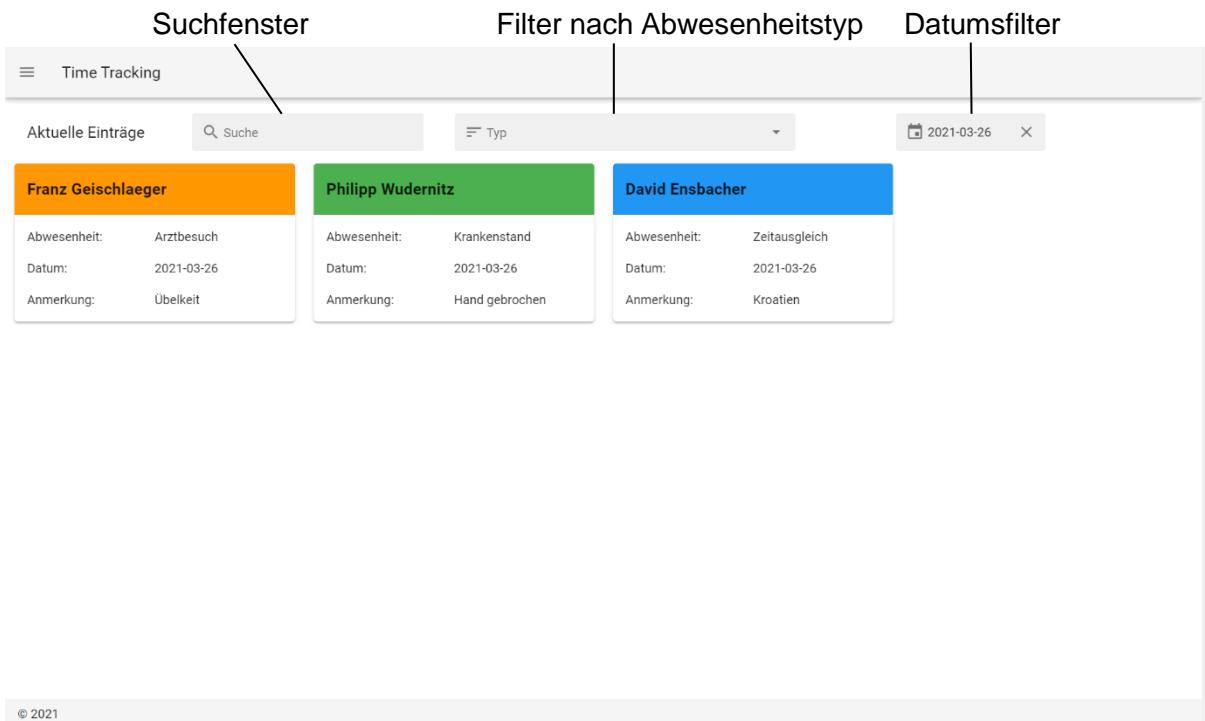


Abbildung 161: Loginseite - Erklärung

7.10.2 Dashboard



The screenshot displays a dashboard interface with a search bar at the top. Annotations point to three specific sections: 'Suchfenster' (Search Window) pointing to the search bar, 'Filter nach Abwesenheitstyp' (Filter by absence type) pointing to a dropdown menu, and 'Datumsfilter' (Date filter) pointing to a date input field set to '2021-03-26'. Below the search bar, there are three cards showing absence details for Franz Geischlaeger, Philipp Wudernitz, and David Ensbacher. Each card includes columns for 'Abwesenheit' (Absence Type), 'Datum' (Date), and 'Anmerkung' (Note). Franz Geischlaeger's entry is 'Arztbesuch' on 2021-03-26 with note 'Übelkeit'. Philipp Wudernitz's entry is 'Krankenstand' on 2021-03-26 with note 'Hand gebrochen'. David Ensbacher's entry is 'Zeitausgleich' on 2021-03-26 with note 'Kroatien'. At the bottom left, there is a copyright notice: '© 2021'.

Abbildung 162: Übersichtsseite - Erklärung

Name des Mitarbeiters



Abbildung 163: Karte - Erklärung

7.10.3 Kalender

Filter nach Eintragstyp (Arbeit, Pause, Abwesenheit)

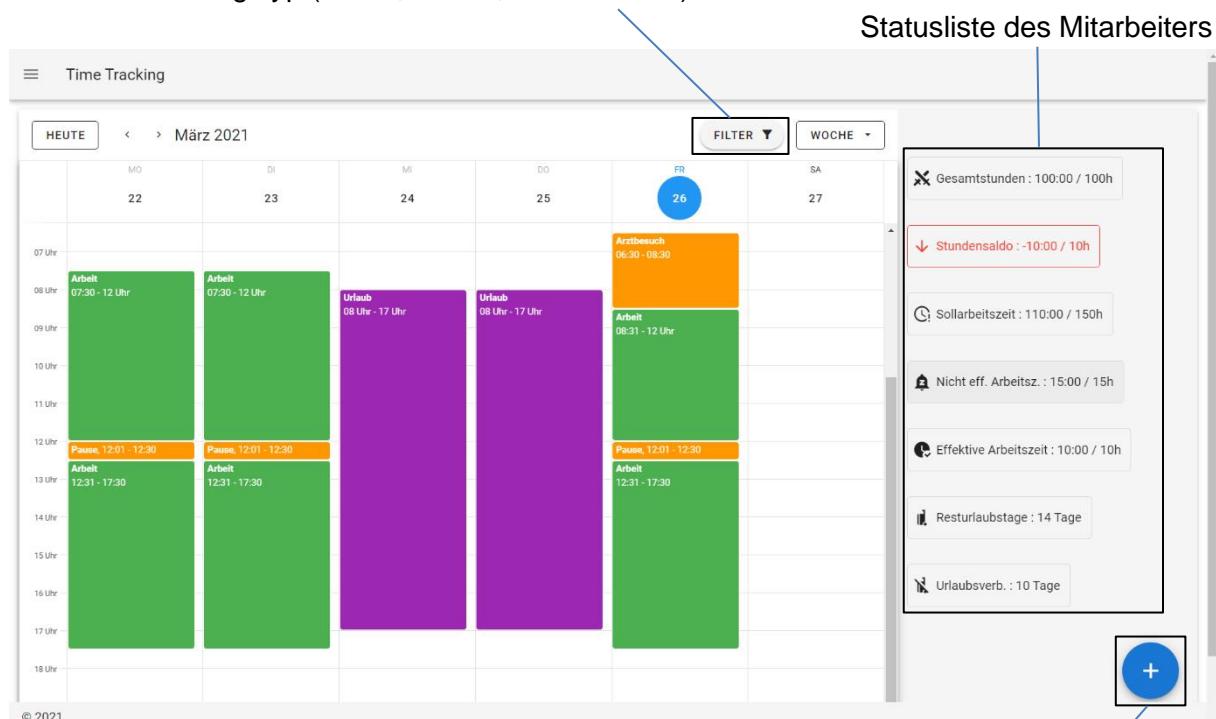


Abbildung 164: Kalenderseite - Erklärung

Floating-Action-Button um Einträge zu erstellen

Nach einem Klick auf den Floating-Action-Button:



Abbildung 165: Kalenderseite - Floating Action Button

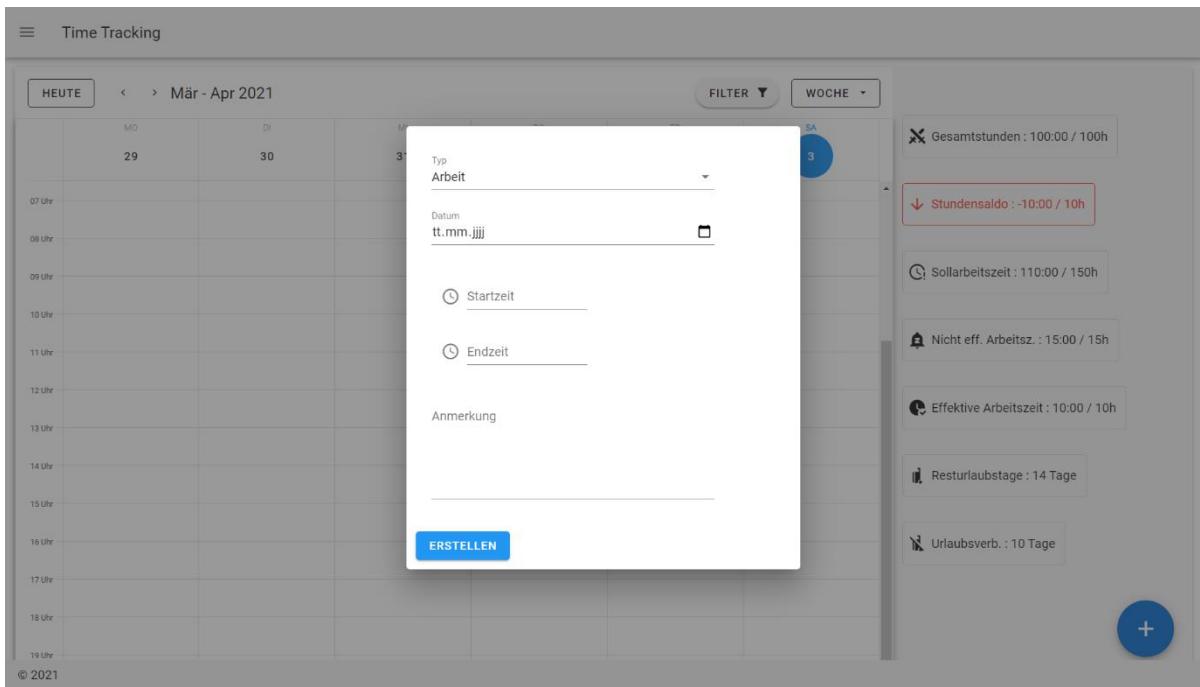
Zeit:


Abbildung 166: Kalenderseite - Zeiteintrag erstellen

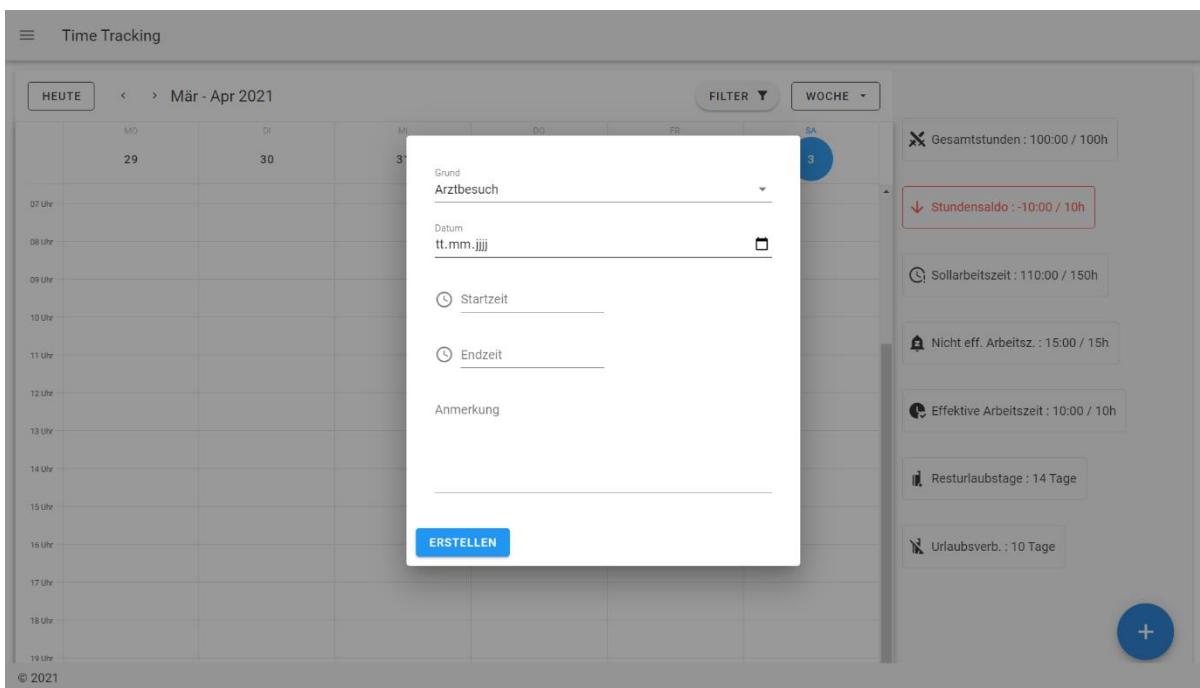
Abwesenheit:


Abbildung 167: Kalenderseite - Abwesenheit erstellen

Eintrag auswählen:

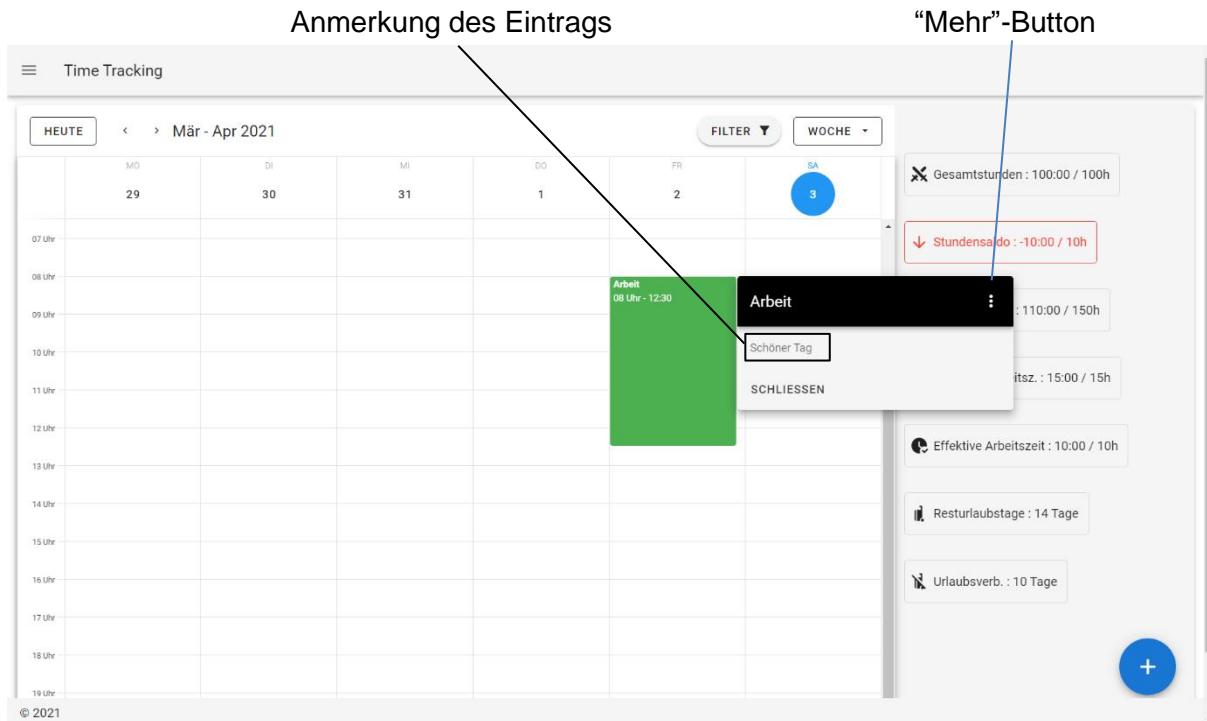


Abbildung 168: Kalenderseite - Eintrag auswählen

Klickt man auf einen Eintrag, so kann man dessen Bezeichnung und Anmerkung sehen.

Auf das "Mehr"-Icon gedrückt:

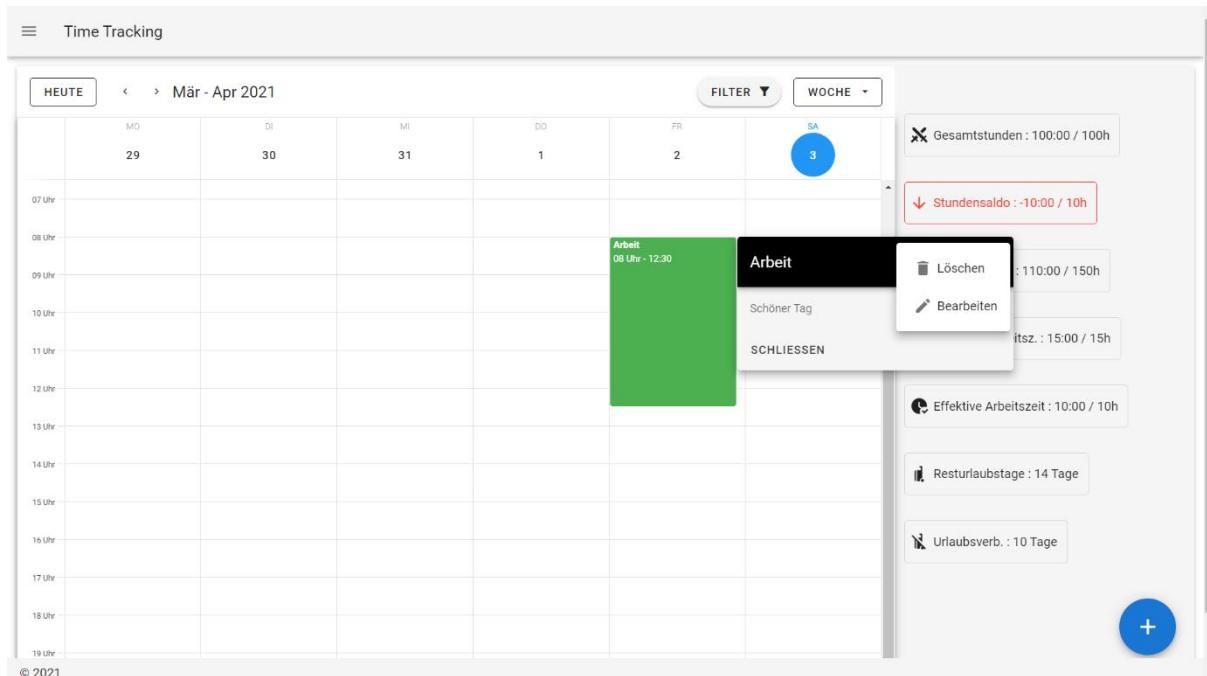


Abbildung 169: Kalenderseite - Eintrag Aktionen

Nun kann man den Eintrag bearbeiten oder löschen.

Eintrag bearbeiten:

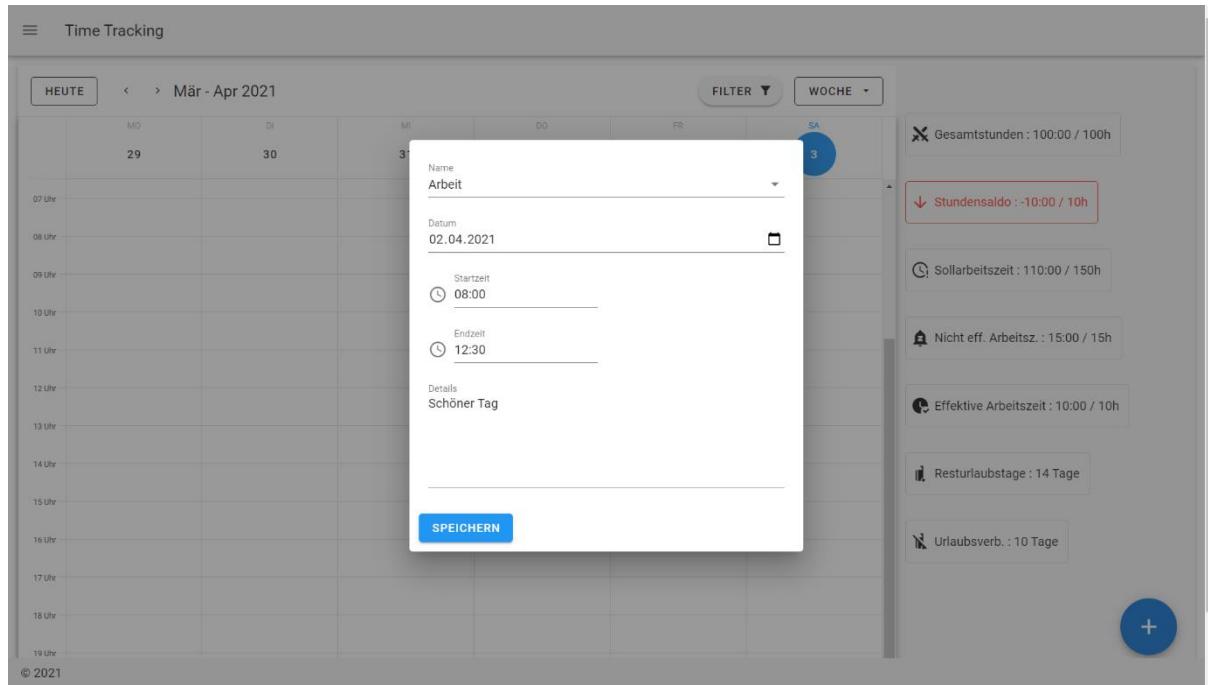


Abbildung 170: Kalenderseite - Eintrag bearbeiten

Der Eintrag lässt sich einfach bearbeiten und aktualisieren.

Eintrag löschen:

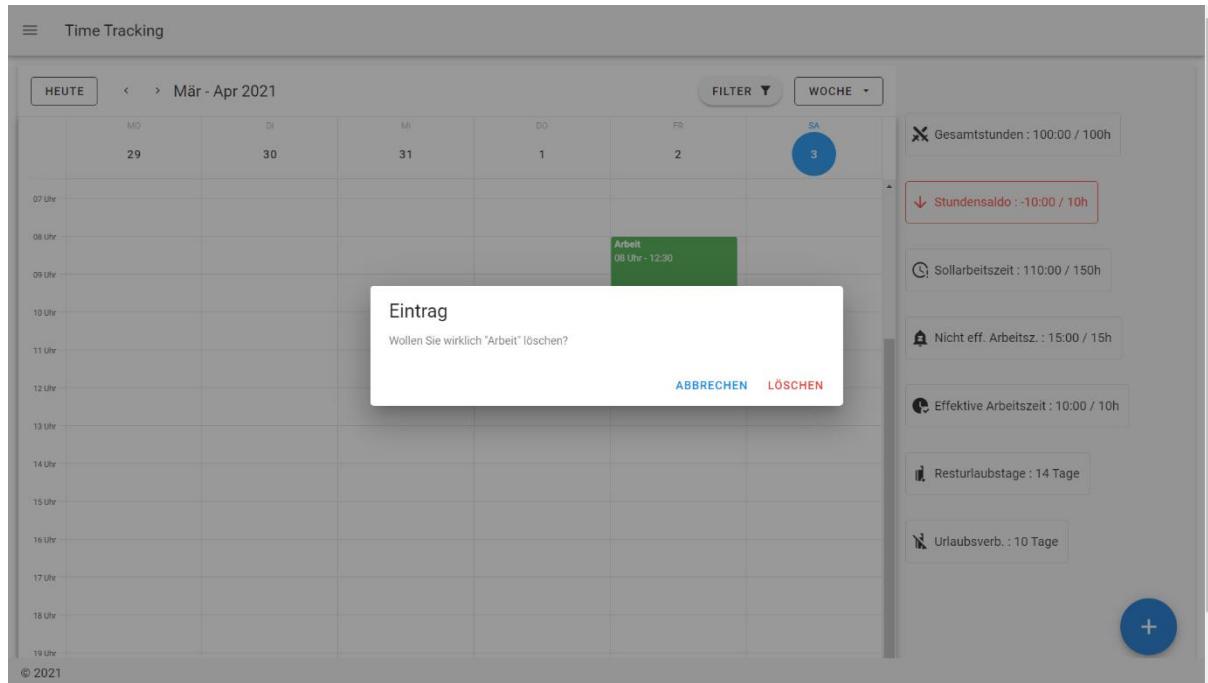


Abbildung 171: Kalenderseite - Eintrag löschen

Vor dem Löschen eines Eintrages erscheint ein Alert, der verhindert, dass man einen Eintrag versehentlich entfernt. Klickt man nun auf „Löschen“, so wird der Eintrag endgültig gelöscht.

7.10.4 Mitarbeiter (Ausschließlich für Admins)

Name	Email	Bearbeiten
 Time Tracking		
Mitarbeiter		
 Geischlaeger Franz	Franz.Geischlaeger@htl-hl.ac.at	
 Zeitlberger Thomas	thomas@zeitlberger.com	
 Hacker Marcel	Marcel@Hacker.at	
 Wudernitz Philipp	Philipp@Wudernitz.at	
 Ensbacher David	David@Ensbacher.at	

Abbildung 172: Mitarbeiterseite - Erklärung

Kalender des Mitarbeiters aufrufen

Bearbeiten:

Mitarbeiter bearbeiten				
Startdatum  2021-03-26	Stunden/Woche 53.5	Montag 11.5	Dienstag 13.5	Mittwoch 9.5
Startzeit Mo 06:30	Startzeit Di 07:30	Startzeit Mi 08:30	Startzeit Do 07:30	Startzeit Fr 07:30
Endzeit Mo 18:30	Endzeit Di 21:30	Endzeit Mi 18:30	Endzeit Do 17:30	Endzeit Fr 17:30
SPEICHERN		ZURÜCKSETZEN		
Historie Eintrittsdatum: 2020-06-30				

7.10.5 Settings

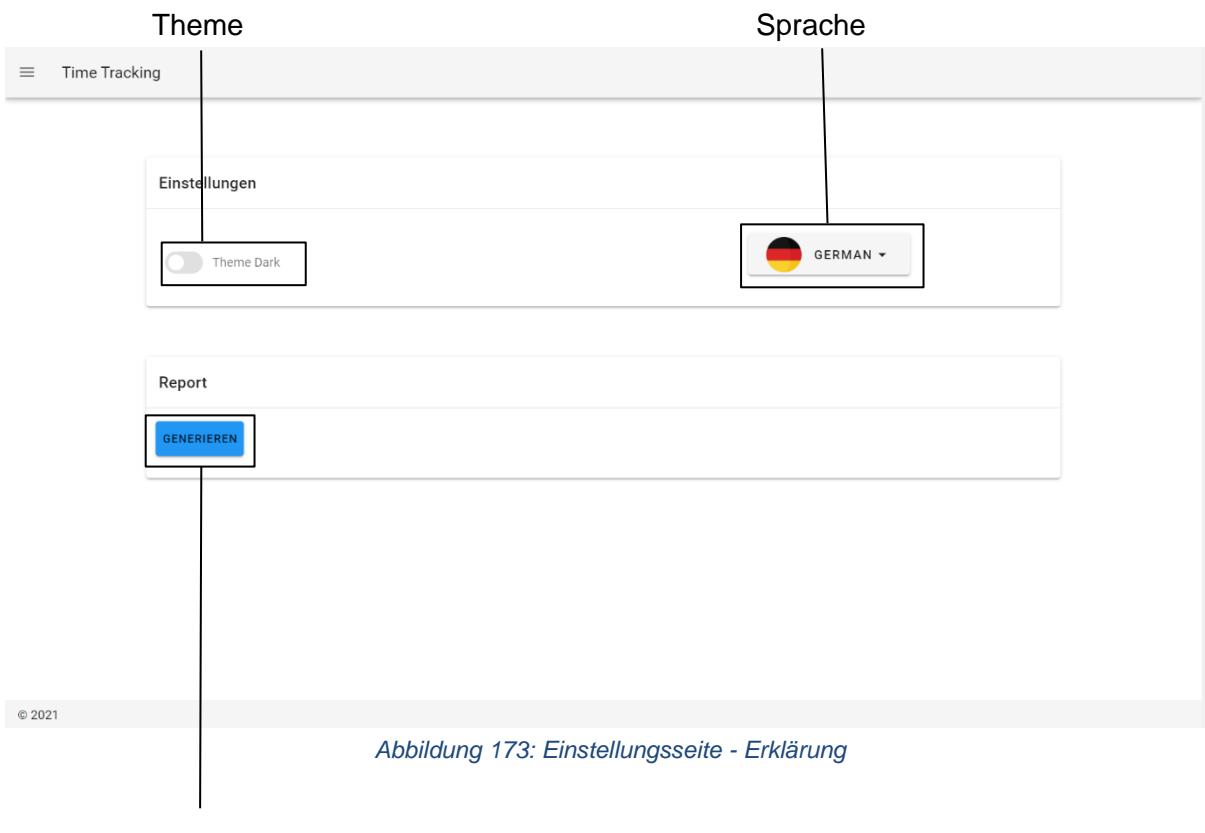


Abbildung 173: Einstellungsseite - Erklärung

Report

7.10.6 Logout

Wird bei der Sidebar auf „Ausloggen“ gedrückt, so werden die Userdaten vom lokalen Speicher gelöscht und der User aus der App ausgeloggt. Der User wird danach auf die Loginseite weitergeleitet.

7.11 Deployen der Vue-App

Um die Vue-App nun zu deployen muss man folgenden Befehl in die Konsole eingeben:

npm run build

Dieser Vorgang sieht folgendermaßen aus:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/pscore6

PS C:\Users\ihack\Google Drive\DA\timetracking> npm run build
```

Abbildung 174: App mit npm builden

Nachdem der Befehl abgeschlossen ist, erscheinen folgende Nachrichten:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
File Size Gzipped
dist\js\chunk-vendors.2c3835da.js 582.67 KiB 176.20 KiB
dist\js\app.af2a5ac3.js 88.08 KiB 24.38 KiB
dist\js\employeedit.daccd0f3.js 12.79 KiB 3.58 KiB
dist\js\absence.f510a53d.js 6.08 KiB 1.89 KiB
dist\js\login.24c4c369.js 4.31 KiB 1.65 KiB
dist\js\error.ac4fa657.js 0.83 KiB 0.55 KiB
dist\css\chunk-vendors.1e5c489d.css 438.93 KiB 53.48 KiB
dist\css\app.ff339118.css 0.97 KiB 0.32 KiB

Images and other types of assets omitted.

[DONE] Build complete. The dist directory is ready to be deployed.
[INFO] Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html

PS C:\Users\ihack\Google Drive\DA\timetracking>
```

Abbildung 175: Buildingprozess abgeschlossen

Im Projektverzeichnis befindet sich nun ein Ordner namens „dist“.

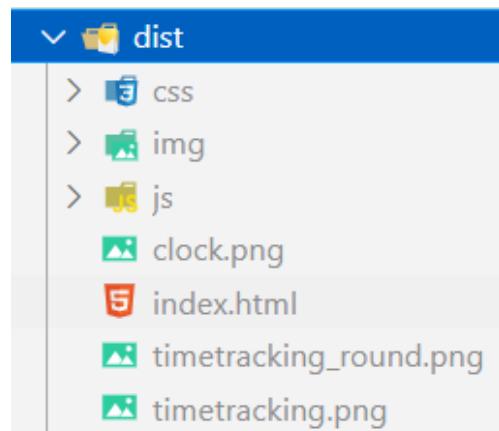


Abbildung 176: dist-Ordner

Dieser beinhaltet nun alle wichtigen Daten, um die App auf einen Web-Server zu hosten.

8 Entwicklung des Backends

8.1 MySQL

MySQL ist eine Cross-Plattform-fähige, Open-Source, sowie kommerzielle Datenbankverwaltungssoftware, welche von großen Firmen, wie Adobe, Facebook und Google, eingesetzt wird. Der häufigste Anwendungsbereich ist die Speicherung der Daten für Webservices.

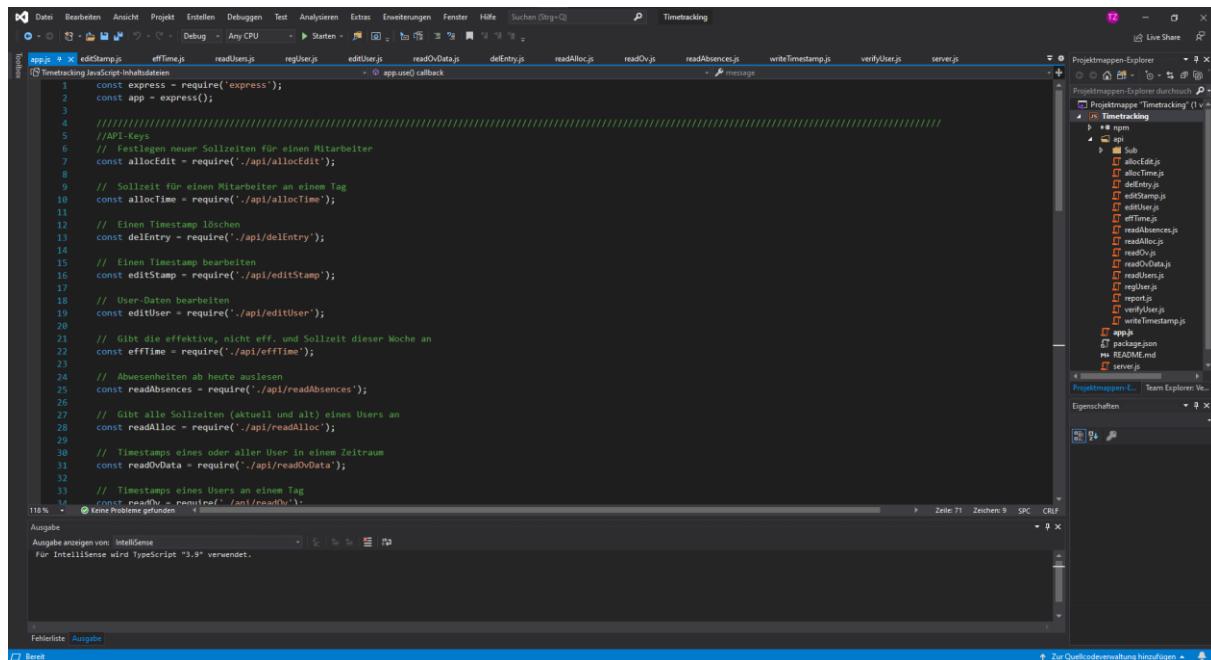
In dieser Diplomarbeit wurde MySQL in Verbindung mit ApacheWebserver eingesetzt. Man hat sich für MySQL entschieden, da bereits Erfahrungen im Unterricht damit gesammelt werden konnten.

8.2 Entwicklungsumgebung und Programme

8.2.1 Visual Studio Community 2019

Visual Studio ist eine kostenlose IDE, welche mit unzähligen Erweiterungen bestückt werden kann. Des Weiteren ist es sehr flexibel und es können unter anderem Applikationen für iOS, Android und Windows, sowie Webanwendungen und Cloud-Dienste entwickelt werden. Es werden viele Sprachen, wie C#, Visual Basic, F#, C++, HTML, JavaScript, TypeScript, Python und mehr, unterstützt.

In dieser Diplomarbeit wurde VS zur Entwicklung des node.js Backends verwendet.



The screenshot shows the Visual Studio interface with the following details:

- Project Explorer:** Shows the project structure for "Timetracking". It includes a "Sub" folder containing "allocEdit.js", "allocTime.js", "delEntry.js", "editStamp.js", "effTime.js", "readAlloc.js", "readAbsences.js", "readData.js", "readEntries.js", "readUser.js", "readV.js", and "writeTimestamp.js". There is also a "server.js" file under the main project folder.
- Code Editor:** Displays the content of the "editStamp.js" file. The code uses Node.js modules like express and fs to handle API requests for stamping time entries.
- Status Bar:** Shows the current zoom level (118%), the number of problems found (0), and the current file (editStamp.js).
- Bottom Navigation:** Includes tabs for "Fehlerliste" (Errors) and "Ausgabe" (Output).

Abbildung 177: VS UI

8.2.2 MySQL Workbench

MySQL Workbench ist eine graphische Entwicklungsumgebung und bietet Datenmodellierung, SQL-Entwicklung und umfassende Administrationswerkzeuge für Serverkonfiguration, Benutzerverwaltung, Backup und ähnliches.

Die Datenbank für diese Diplomarbeit, wurde mithilfe der MySQL Workbench graphisch designt.

Insgesamt befinden sich fünf Tabellen in der Datenbank, Daily, User, Type, Allocated, BT und GF.

In der Daily-Tabelle liegen die Daten der Timestamps wie:

- fortlaufende ID (idDaily)
- die Uhrzeit (stamp)
- das Datum (date)
- ein Kommentar (comment)
- die Auswahl, ob eine Abwesenheit weitergeführt werden soll bis zum nächsten Tracking (fulltime)
- die zugehörige User-ID (User_idUser)
- die ID des Typen (Typen_idType).

In der User-Tabelle befinden sich die User-Daten, welche von Redmine zur Verfügung gestellt werden:

- die User-ID (idUser)
- Vorname (firstname)
- Nachname (lastname)
- E-Mail (email)
- wann der Account angelegt wurde (joined)
- der Api-Key welcher zur Verifizierung verwendet wird (apikey)

Die Typen-Tabelle verbindet Typen-ID mit dem Typen als Wort.

Zur Speicherung der Sollzeiten wird die Allocated-Tabelle verwendet:

- ID der Sollzeiten (idAllocated)
- die zugehörige User-ID (User_idUser)
- Startzeit jedes Wochentags (mon- ...friststart)
- Stoppzeit jedes Wochentags (mon- ...fristop)
- Datum ab wann diese Zeiten gelten (date)

In der BT-Tabelle befinden sich die Daten der zugelassenen Bluetooth-Beacons:

- ID des Eintrags (idBT)
- Identifikation des Beacons (UUID)
- Unterteilung der Beacons
 - minor
 - major
- Beaconradius (radius)



In der GF-Tabelle befinden sich die Daten der Geofences:

- ID des Geofences (idGF)
- Identifikation des Geofences (Identifier)
- Koordinaten des Geofence-Mittelpunktes
 - Breitengrad (latitude)
 - Längengrad (longitude)
- Geofenceradius (radius)

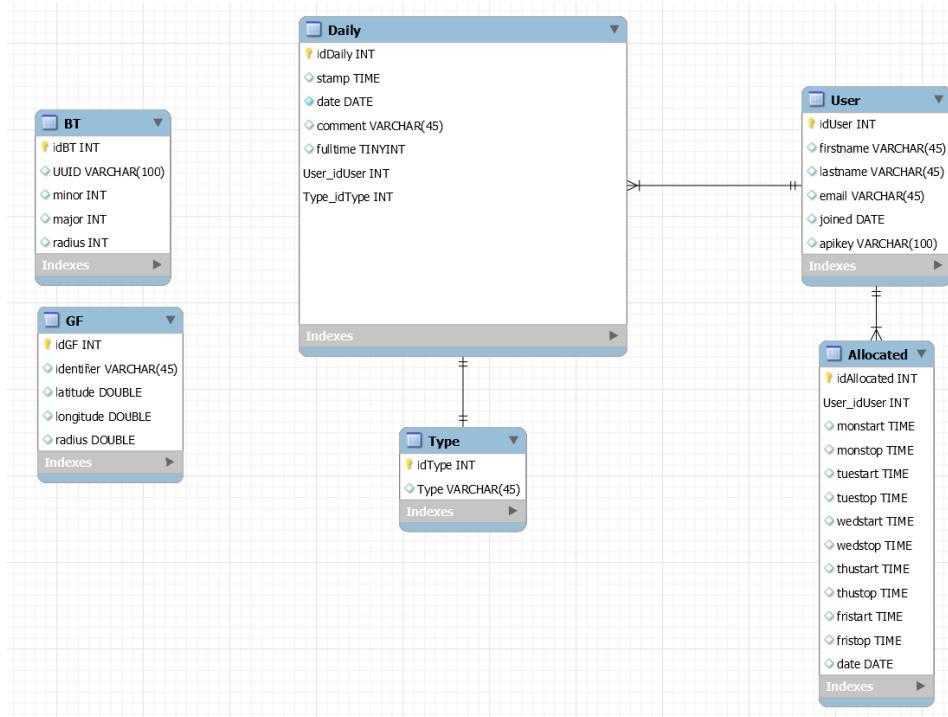


Abbildung 178: MySQL Workbench Editor

Anschließend wurde das graphische Design als Code exportiert.

```
1 -- MySQL Script generated by MySQL Workbench
2 -- Thu Mar 23 18:18:13 2017
3 -- Model: New Model  Version: 1.0
4 -- MySQL Workbench Forward Engineering
5
6 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
7 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
8 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
9
10 -----
11 -- Schema mydb
12 -----
13 --
14 --
15 -- Schema mydb
16 -----
17 CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
18 USE `mydb` ;
19 --
20 --
21 -- Table 'mydb`.`User'
22 -----
23 CREATE TABLE IF NOT EXISTS `mydb`.`User` (
24   `idUser` INT NOT NULL AUTO_INCREMENT,
25   `firstname` VARCHAR(45) NULL,
26   `lastname` VARCHAR(45) NULL,
27   `email` VARCHAR(45) NULL,
28   `joined` DATE NULL,
29   `api_key` VARCHAR(100) NULL,
30   PRIMARY KEY (`idUser`),
31   ENGINE = InnoDB;
32
33
34 --
35 -- Table 'mydb`.`Type'
36 -----
```

Abbildung 179: MySQL Code-Export

8.2.3 XAMPP

XAMPP ist ein Cross-Plattform-fähiges, Open-Source Programm, welches von ApacheFriends entwickelt wurde.

Es dient der einfachen Installation und Konfiguration von Apache-Webservern mit der Datenbank MariaDB (früher MySQL) bzw. SQLite.

In der Diplomarbeit wurde XAMPP für das lokale Hosting des Datenbank-Servers verwendet, damit dessen Verbindung zum Backend und die Funktionalität getestet werden konnte, bevor die Backend-Files auf den Server hochgeladen wurden.

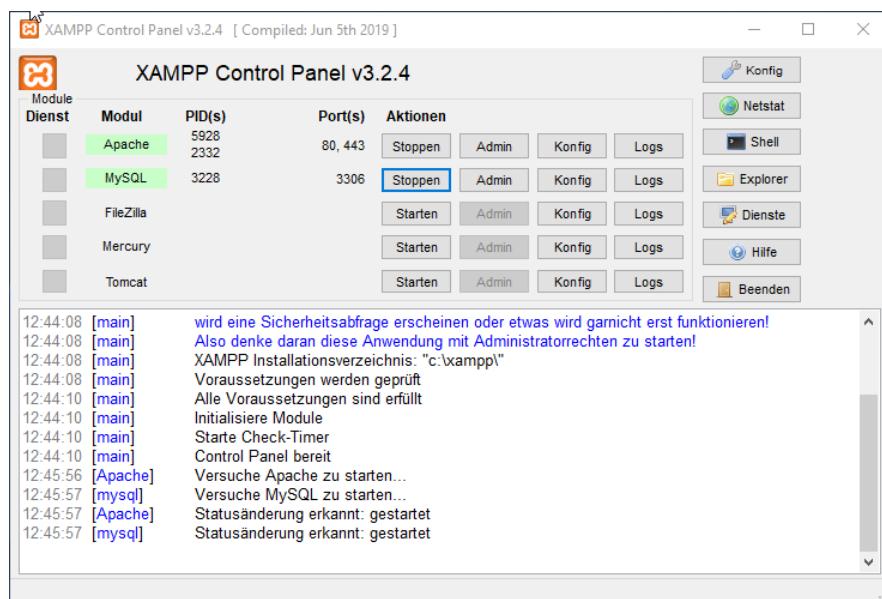


Abbildung 180: XAMPP UI

8.2.4 Postman

Postman ist ein Tool, welches für die API-Entwicklung verwendet wird. Es kann dazu genutzt werden um REST, SOAP und GraphQL Requests zu senden und die API damit zu testen. Des Weiteren kann man mit Postman eine API bauen, bevor man mit dem Entwickeln des Codes beginnt.

Mit Postman wurden Anfragen an das Backend gesendet und die API-Keys auf deren Funktionalität und Fehler getestet.

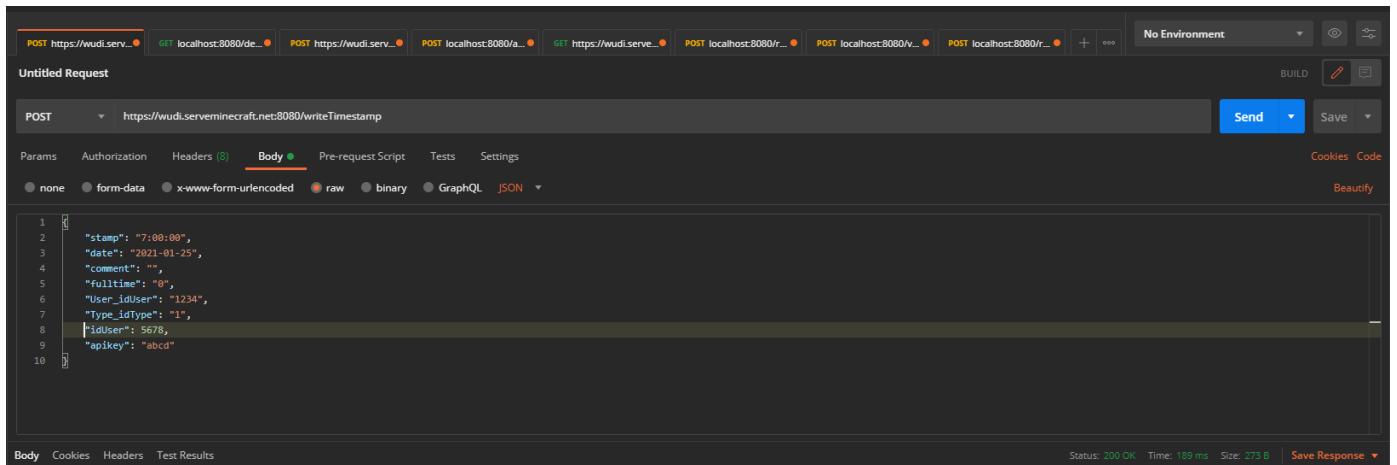


Abbildung 181: Postman UI

8.2.5 FileZilla

FileZilla ist eine freie, Open-Source und Cross-Plattform-fähige Client- Server- Software, welche zur Datenübertragung genutzt wird. Für die Verbindung zwischen Client und Server werden FTP und SFTP eingesetzt und darüber hinaus können damit Dateien hoch und runtergeladen werden. Des Weiteren kann man Dateien, welche sich auf dem Server befinden, lokal im Texteditor öffnen und bearbeiten und anschließend wieder mit dem Server synchronisieren.

Mithilfe von Filezilla wird auf die Daten des Diplomarbeits-Servers, auf welchem das Backend, der Apache Server die MySQL Datenbank und die Web-Applikation laufen, zugegriffen.

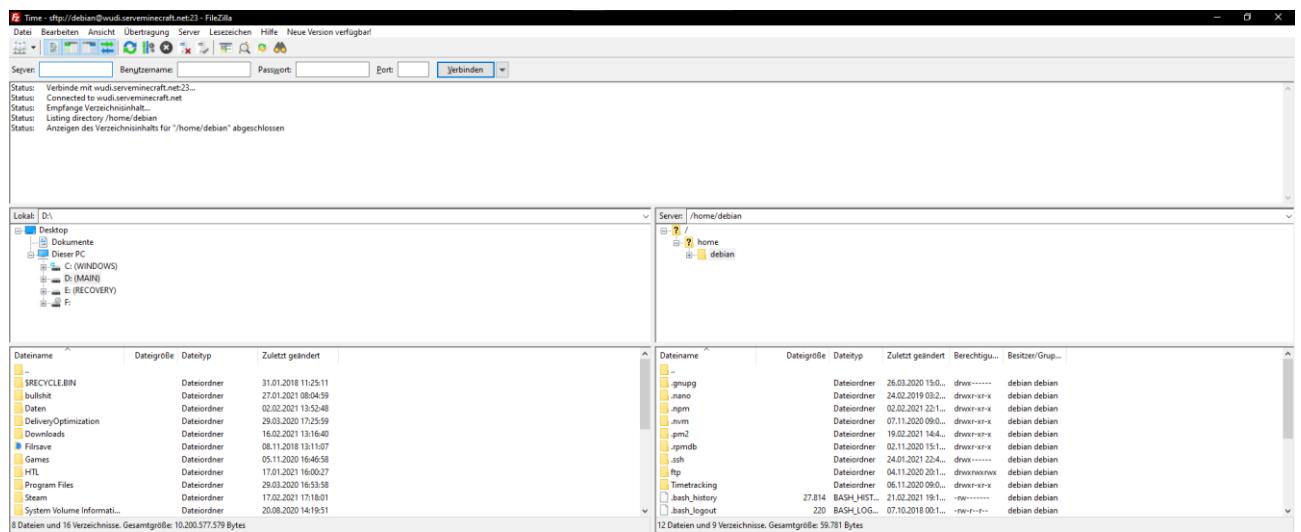


Abbildung 182: FileZilla UI

8.2.6 PuTTY

PuTTY ist ein freies, Open-Source und Cross-Plattform-fähiges Programm, welches genutzt wird, um eine Verbindung zu einem Gerät, Server o.ä. über SSH (Secure Shell), Telnet, Remote login oder serieller Schnittstelle aufzubauen. Die Sitzung wird im Terminal dargestellt, über welches Befehle an das verbundene Gerät geschickt und ausgeführt werden können.

PuTTY wird verwendet, um eine sichere Verbindung zum Diplomarbeits-Server aufzubauen und diesen über die Konsole mit dem Root-User zu konfigurieren.

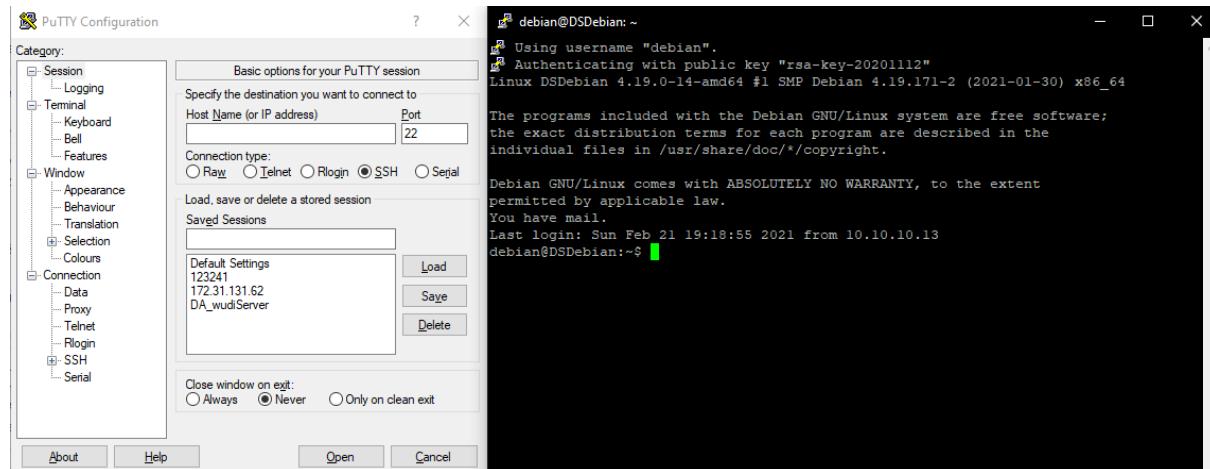


Abbildung 183: PuTTY Config und CMD

8.3 Programmiersprachen und Plattformen

8.3.1 Node.js

Node.js ist ein Cross-Plattform-fähiges, Open-Source-Runtime-Environment und das Framework, mit welchem das Backend dieser Diplomarbeit entwickelt wurde.

Mit Node.js ist es möglich JavaScript-Code außerhalb eines Browsers auszuführen und kann dazu eingesetzt werden, um einen Web-Server zu betreiben. Des Weiteren wird mit der Installation von Node.js auch ein Package Manager mitgeliefert, Node Package Manager (npm).

Für jedes Projekt muss ein Package mithilfe des Commands

```
npm init
```

generiert werden und es müssen die Credentials eingetragen werden.

Folgende Pakete wurden im Backend benötigt und mithilfe von npm installiert:



Wird genutzt zur einfachen Übertragung von JSON-Objekten:

```
npm install body-parser
```

Wird genutzt, um eine Web-Applikation zu generieren:

```
npm install express
```

Vereinfacht die Verbindung zur MySQL-Datenbank:

```
npm install mysql
```

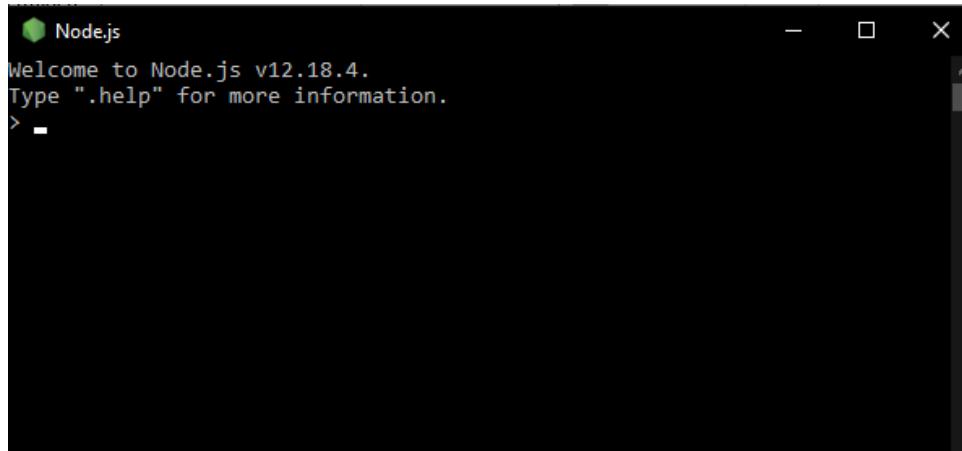


Abbildung 184: Node.js Command Line

8.3.2 JSON

JSON, kurz für Javascript Object Notation, ist ein kompaktes Datenformat, welches eine, für Menschen einfach lesbare Textform verwendet. Der Hauptzweck von JSON ist die Vereinfachung des Datenaustausches zwischen verschiedenen Anwendungen und ist von keiner Programmiersprache abhängig

In dieser Diplomarbeit wurde JSON für den Datenaustausch zwischen den Frontend-Applikationen und dem Backend verwendet.

Beispiel:

```
{  
    "array": [1, 2, 3, 4, 5],  
    "boolean": true,  
    "color": "#f51a69",  
    "number": 123,  
    "object": {  
        "firstname": "John",  
        "lastname": "Doe",  
        "email": "johndoe@email.com"  
    },  
}
```

```

    "string": "Hello World"
}

```

Listing 124: JSON Beispiel

8.4 Server

Der PC auf welchem die Webseite und die Backend-Server gehostet werden, ist ein Debian-System auf welches über PuTTY (Command Line) und FileZilla (Filesystem) zugegriffen wurde.

8.4.1 Apache Web-Server

Apache ist eine freie, Open-Source und Cross-Plattform fähige Web-Server-Software. Dieses Paket kann genutzt werden, um einen Computer als HTTP-Server zu verwenden und das Web-Frontend zu hosten.

Installation:

Zu Beginn sollte man die lokale Paket-Datenbank aktualisieren:

```
sudo apt-get update
```

Danach kann das Apache-Paket installiert werden:

```
sudo apt-get install apache2
```

Anschließend wird vom System nach Erlaubnis gefragt:

```

dejan@dejan-VirtualBox:~$ sudo apt-get install apache2
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  bridge-utils ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap
0 upgraded, 8 newly installed, 0 to remove and 257 not upgraded.
Need to get 1,604 kB of archives.
After this operation, 6,493 kB of additional disk space will be used.
Do you want to continue? [Y/n] y

```

Abbildung 185: Apache Installation

Indem man die lokale Adresse in einen Browser eingibt, kann man überprüfen, ob die Installation erfolgreich war.

<http://local.server.ip>

Mit folgendem Command kann man die lokale Server-Adresse ausgeben:

```
hostname -I | awk '{print $1}'
```

Schlussendlich muss noch der richtige Port in der Firewall freigegeben werden. Mit diesem Command wird normaler Internetverkehr für den Port 80 freigegeben:

```
sudo ufw allow 'Apache'
```

[47]

8.4.2 MySQL Server

Um auf die Datenbank zugreifen zu können muss diese auf einem MySQL-Server laufen. In dieser Diplomarbeit lief der Server auf einem Debian-Server und wurde wie folgt aufgesetzt:

Command zum Downloaden des MySQL APT Repositories:

```
wget http://repo.mysql.com/mysql-apt-config_0.8.13-1_all.deb
```

Command zur Installation des Paketes:

```
sudo apt install ./mysql-apt-config_0.8.13-1_all.deb
```

Anschließend kann man, wenn man möchte die MySQL-Version anpassen, welche man installieren möchte und drückt dann auf <Ok>:

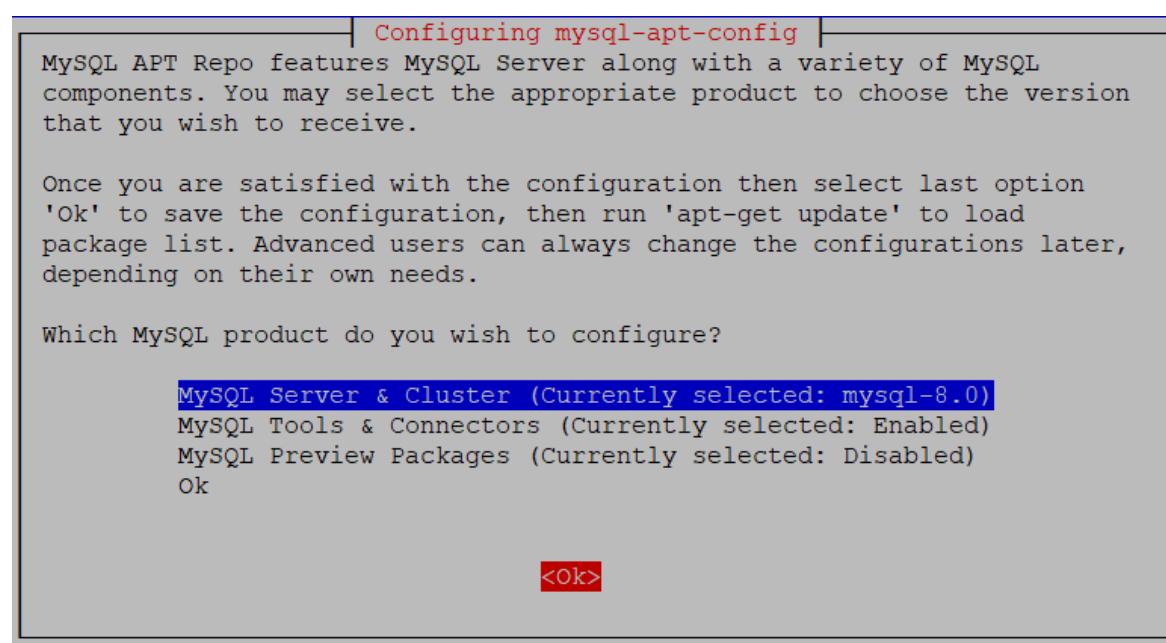


Abbildung 186: MySQL Configuration

Das Repository ist nun angelegt und es ist an der Zeit den MySQL-Server zu installieren. Zuerst muss die package list upgedatet werden und anschließend kann mit der Installation begonnen werden:

```
sudo apt update  
sudo apt install mysql-server
```

Als nächstes scheint eine Meldung auf, in welcher man auswählen muss, ob man die neue MySQL 8.0 Authentication haben möchte. Dieses Plugin muss auch von der Applikation unterstützt werden. In dieser DA wurde diese nicht verwendet.

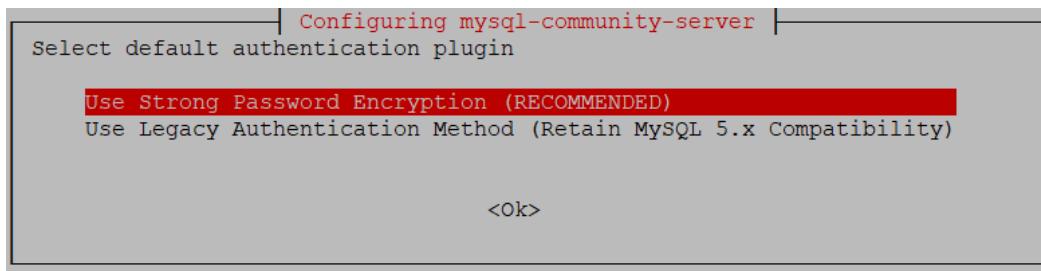


Abbildung 187: MySQL Encryption

Die Installation ist nun abgeschlossen und der MySQL-Server startet automatisch. Dies kann man mit folgendem Command überprüfen:

```
sudo systemctl status mysql
```

8.4.3 Server mit Node.js

Mit Node.js wird ein https-Server gehostet, auf den von außen über eine API zugegriffen werden kann.

Um https auf einem Server anbieten zu können wird ein SSL-Zertifikat benötigt. Dieses wurde mithilfe von „Let's Encrypt“ generiert und eingebunden.

Um Let's Encrypt zu installieren muss zuerst der Apache HTTP Server installiert werden. Anschließend kann das Certbot-Paket installiert werden.

```
sudo apt install python3-certbot-apache
```

Mit dem Certbot-Tool kann man SSL-Zertifikate für mehrere Domains generieren.

```
sudo certbot --apache
```

Danach wird nach einer gültigen E-Mail-Adresse gefragt, der Annahme der Geschäftsbedingungen und ob man den Newsletter abonnieren möchte.

Man erhält eine E-Mail, bevor das Zertifikat ausläuft.

[49]



Code:

```
// Einbindung von Packages und Libraries
var fs = require('fs');
var path = require('path');
var https = require('https');
const app = require('./app');
const port = 8080;
const host = '10.10.10.40';

// https-Zertifikate
const options ={
    key: fs.readFileSync('/etc/.../privkey.pem'),
    cert: fs.readFileSync('/etc/.../cert.pem')
};

const server = https.createServer(options, app);
server.listen(port, host, () => {
    console.log(`Server is running on https:// ${host} : ${port}`);
});
```

Listing 125: Node Server

8.5 API

Um die Daten vom Backend zu den Frontends zu transferieren wird eine Schnittstelle benötigt. Diese wird in der Form einer API, einem Application Programming Interface, entwickelt. Die API definiert die möglichen Calls oder Request, wie diese ausgeführt werden müssen bzw. welche Daten, in welchem Format, übergeben werden müssen

8.5.1 RESTful API

Eine der wichtigsten Eigenschaften einer REST-API ist die Einheitlichkeit der Schnittstelle, wodurch eine einfache Nutzung gewährleistet wird.

Die Nachrichten der REST-API enthält alle Informationen, welche benötigt werden, um den Inhalt der Nachrichten zu verstehen. Des Weiteren sollen weder der Server noch die Anwendung Zustandsinformationen zwischen zwei Nachrichten speichern. [48]

8.5.2 API-Keys

8.5.2.1 Authentifizierung jedes Calls

Damit von außen nicht jeder auf die Datenbank über die API-Keys zugreifen kann, muss man sich für jeden Call authentifizieren. Ausnahme für diese Regel sind die Calls zur Registrierung der User und zur Überprüfung, ob der User bereits existiert.

Um sich zu authentifizieren muss, neben den Daten, welche für den Call benötigt werden, auch die User-ID und der dazugehörige Key übergeben werden. Diese erhält man am Beginn, wenn man sich einloggt von der Redmine-API und wird später mit den Daten aus der Datenbank verglichen. Falls die Daten übereinstimmen, wird das Programm fortgesetzt, ansonst wird ein JSON zurückgegeben, in welchem steht, dass man nicht autorisiert sei.

JSON-Übergabe:

```
{
  .
  .
  .
  "LoginUser": 1234,
  "apikey": "7w3gcrb8i34hi"
}
```

Listing 126: Authentifizierung Übergabe

8.5.2.2 Auslesen der Timestamps

readOv.js:

Um sich einen Tag genauer im Detail ansehen zu können, brauchen die Apps die zu diesem Tag zugehörigen Daten. Dieser Key verlangt vom Frontend die User-ID und ein Datum und im Gegenzug bekommt er die Timestamp-IDs, sowie die Uhrzeit des Events, den passenden Typ (z.B. Start einer Arbeitszeit) und die gesamte Arbeitszeit dieses Tages.

URL: <https://wudi.serveminecraft.net:8080/readOv/>

Beispiel:

JSON-Übergabe:

```
{
  "idUser": 1234,
  "date": "2021-01-01"
}
```

Listing 127: readOv Übergabe

JSON-Rückgabe:

```
{
  "id": [1, 2, 3, 4, 5],
  "events": ["7:30:00", "11:30:00", "12:00:00", "17:30:00"],
  "type": ["Start", "Pause", "Start", "Stop"],
  "time": ""
}
```

Listing 128: readOv Rückgabe



Code:

```
//Auslesen der Timestamps eines bestimmten Datums aus der DB
mclient.query("SELECT ...", function (err, results) {
    if (err) throw err;

    // Timestamps in ein Objekt schreiben
    var time = JSON.parse(JSON.stringify(results));

    // Timestamps aussortieren
    for (var i = 0; i < time.length; i++) {

        // Ersten Timestamp ins Objekt schreiben
        if (i == 0) {
            data.id[y] = time[i].idDaily;
            data.events[y] = time[i].stamp;
            data.type[y] = time[i].type;
            y++;
        } // Sich Wiederholende Timestamps werden ignoriert
        else if (time[i].type != time[i - 1].type) {
            data.id[y] = time[i].idDaily;
            data.events[y] = time[i].stamp;
            data.type[y] = time[i].type;
            y++;
        }

        // Startzeit wird festgelegt
        if ((token == 0) && (time[i].type == "Start")) {
            var a = (time[i].stamp).split(':');
            start = (+a[0])* 60* 60 + (+a[1]) * 60 + (+a[2]);
            token = 1;
        } // Zeitdifferenz wird errechnet und addiert
        else if ((token == 1) && (time[i].type != "Start")) {
            var a = (time[i].stamp).split(':');
            stop = (+a[0])* 60* 60 + (+a[1]) * 60 + (+a[2]);

            if (count == 0) {
                out = stop - start;
            }
            else {
                out = out + stop - start;
            }
            token = 0;
            count++;
        }
    } //Errechnete Zeit wird ins Objekt geschrieben
    if (count != 0) {
```

```

        data.time = out;
    }

    // Rückgabe des Objekts
    res.status(200).json(
        data
    );

```

Listing 129: Code "readOv"

readOvData.js:

Die Web-Applikation hat einen Kalender in welchem die Arbeitszeiten des Users angezeigt werden. Wenn das Frontend die Anfrage für die Daten an den API-Call sendet, erhält es die IDs der Timestamps, den zugehörigen Namen, sowie die Uhrzeit, das Datum, den Kommentar, die User-ID und den Typ des Stamps.

URL: <https://wudi.serveminecraft.net:8080/readOvData>

Beispiel:

JSON-Übergabe:

```
{
    "startdate": "2021-01-12",
    "stopdate": "2021-01-18",
    "idUser": 1234 oder "all" //Alle User mit all
}
```

Listing 130: readOvData Übergabe

JSON-Rückgabe:

```
{
    "idDaily": 1,
    "name": "Thomas Zeitlberger",
    "stamp": "7:30:00",
    "date": "2021-03-25",
    "comment": "",
    "userID": 1234,
    "type": "Start"
}
```

Listing 131: readOvData Rückgabe

Code:

```

// Routine, ob Timestamps aller Nutzer oder von einem
// ausgegeben werden müssen
if (dataIn.idUser == "all") {
    var ifUser = "";
}
else {

```



```
var ifUser =" Daily.User_idUser = "+dataIn.idUser+" AND ";
}

mysqlLib.getConnection(function (err, mclient) {
    // Auslesen der Timestamps aus der Datenbank
    mclient.query("SELECT ... ", function (err, results) {
        if (err) throw err;

        res.status(200).json(
            results
        );
    });
    // Schließen der Datenbankverbindung
    mclient.release();
});
```

Listing 132: Code „readOvData“

8.5.2.3 Auslesen der Mitarbeiterinformation

readUsers.js:

Das Frontend verwendet diesen Call, um die Daten der User anzuzeigen.

URL: <https://wudi.serveminecraft.net:8080/readUsers>

Beispiel:

JSON-Übergabe:

Bei diesem Key werden nur die Authenfizierungs-Daten (9.5.2.1 Authentifizierung jedes Calls) benötigt.

JSON-Rückgabe:

```
{
    "idUser": 1234,
    "firstname": "John",
    "lastname": "Doe",
    "email": "johndoe@email.com",
    "joined": "2016-08-01",
    "apikey": "7w3gcrb8i34hi"
}
```

Listing 133: readUsers Rückgabe

Code:

```
// Auslesen der User aus der Datenbank
mysqlLib.getConnection(function (err, mclient) {
    mclient.query("SELECT * from User", function (err, results) {
        if (err) throw err;

        res.status(200).json(
            results
        );
    });
    mclient.release();
});
```

Listing 134: Code „readUsers“

8.5.2.4 Auslesen der Abwesenheiten

readAbsences.js:

Die Apps haben eine Anzeige, in welcher sie die aktuellen bzw. die zukünftigen Abwesenheiten des Nutzers anzeigen können. Dieser Key verarbeitet nur die User-ID und liefert die Information, ob die Abwesenheit ganztägig ist bzw. bis zum nächsten Stamp dauert, den Typen, falls vorhanden den Kommentar und die Start- und Stoppzeit.

Beispiel:

JSON-Übergabe:

```
{
    "idUser": 1234
}
```

Listing 135: readAbsences Übergabe

JSON-Rückgabe:

```
{
    {
        "fulltime": false,
        "type": "Arztbesuch",
        "comment": "Optiker",
        "start": "2021-03-18 14:30:00",
        "end": "2021-03-18 16:30:00"
    },
    {
        "fulltime": true,
        "type": "Urlaub",
        "comment": "Kroatien",
        "start": "2021-03-30 07:30:00",
        "end": ""
    }
}
```

Listing 136: readAbsences Rückgabe



Code:

```
// Heutiges Datum wird umformatiert
var dt = date.getFullYear() + "-" + (date.getMonth() +
1).toLocaleString('en-US', { minimumIntegerDigits: 2 }) + "-" +
date.getDate().toLocaleString('en-US', { minimumIntegerDigits: 2 });

// aktuelle Uhrzeit formatieren
var time = date.getHours().toLocaleString('en-US', {
minimumIntegerDigits: 2 }) + ":" +
date.getMinutes().toLocaleString('en-US', { minimumIntegerDigits: 2 });

// Alle Abwesenheiten werden aus der Datenbank ausgelesen
mysqlLib.getConnection(function (err, mclient) {
    mclient.query("SELECT ...", function (err, results) {
        if (err) throw err;

        // Alle Abwesenheiten die älter sind werden aussortiert
        for (var i = 0; i < results.length; i++) {
            if (results[i].stamp > time || results[i].date > date) {
                if (i != 0) {
                    z = i - 1;
                }
                break;
            }
        }

        for (z; z < results.length; z++) {

            // Abwesenheitsanfang bestimmen
            if (results[z].type != 'Start' && results[z].type != 'Stop' && results[z].type != 'Pause') {

                if (count == 0) {
                    // Falls noch keine Abwesenheit davor begonnen hat
                    out.fulltime = results[z].fulltime;
                    out.type = results[z].type;
                    out.start =
                        formatDateLib.formatDate(results[z].date) + " "
                    + results[z].stamp;
                    out.comment = results[z].comment;
                }
            }
        }
    });
});
```

```

        count++;
    }
    else {
        // Beenden der begonnenen Abwesenheit und Anfangen
        // einer Neuen
        out.end = results[z].date + " " + results[z].stamp;

        fout[fout.length] = JSON.parse(JSON.stringify(out));
        // array.push erstellt nur referenz kein neues
        // Objekt (Shallow Copy)
        // JSON.parse(JSON.stringify(out)) macht Kopie vom
        // Objekt (Deep Copy)

        out.fulltime = results[z].fulltime;
        out.type = results[z].type;
        out.start = formatDateLib.formatDate(results[z].date)
        + " " + results[z].stamp;
        out.comment = results[z].comment;
    }
}
else if (count != 0) {
    // Beenden der begonnenen Abwesenheit
    out.end = formatDateLib.formatDate(results[z].date)
    + " " + results[z].stamp;
    fout[fout.length] = JSON.parse(JSON.stringify(out));
    count = 0;
}
if ((out.start != null) && (fout[fout.length - 1] != out) &&
(count != 0)) {

    // Falls die Abwesenheit noch nicht beendet wurde
    out.end = "";
    fout[fout.length] =
    JSON.parse(JSON.stringify(out));
}

```

Listing 137: Code „readAbsences“

8.5.2.5 Auslesen von Sollzeiten

allocTime.js:

Die Apps zeigen auf dem Homescreen die Sollarbeitszeit, welche an diesem Tag zu leisten ist. Dieser Key benutzt die User-ID und das Datum des anzuzeigenden Tages und liefert im Gegenzug die Start- und Stopnzeiten, die Differenz in Stunden sowie in Sekunden.

URL: <https://wudi.serveminecraft.net:8080/allocTime>



Beispiel:

JSON-Übergabe:

```
{  
    "date": "2021-03-19",  
    "idUser": 1234  
}
```

Listing 138: allocTime Übergabe

JSON-Rückgabe:

```
{  
    "start": "7:30:00",  
    "stop": "17:30:00",  
    "diff": "10:00:00",  
    "diff_sec": 36000  
}
```

Listing 139: allocTime Rückgabe

Code:

```
// Erkennen ob es ein Werktag ist  
if ((day != "sat") && (day != "sun")) {  
    // Variable für die gewollten Daten  
    var wanteddata = day + "start start, " + day + "stop stop,  
    TIMEDIFF(" + day + "stop," + day + "start) diff,  
    TIME_TO_SEC(TIMEDIFF(" + day + "stop," + day + "start))  
    diff_sec";  
  
    mysqlLib.getConnection(function (err, mclient) {  
        // Auslesen der Sollzeiten  
        mclient.query("SELECT " + wanteddata + " FROM Allocated  
        ...", function(err, results) {  
            if (err) throw err;  
  
            // Schreiben der Daten in ein Objekt  
            data.start = results[0].start;  
            data.stop = results[0].stop;  
            data.diff = results[0].diff;  
            data.diff_sec = results[0].diff_sec;  
  
            // Rückgabe der Daten über JSON  
            res.status(200).json(  
                data  
            );  
        });  
        mclient.release();  
    });
```

```

    });
}

else { // Falls am Wochenende abgefragt wird
    res.status(200).json({
        message: "Weekend"
    });
}

```

Listing 140: Code „allocTime“

readAlloc.js:

Das Web-Frontend nutzt diesen Key, um die Sollzeiten eines bestimmten Users zu erhalten und darzustellen. Es werden nicht nur die aktuellen Sollzeiten übertragen, sondern auch die Arbeitsstunden von früheren Sollzeiten sowie das Datum ab welchem diese Zeiten gültig waren. Diese werden in einer Historie angezeigt.

URL: <https://wudi.serveminecraft.net:8080/readAlloc/>

Beispiel:

JSON-Übergabe:

```
{
    "idUser": 1234
}
```

Listing 141: readAlloc Übergabe

JSON-Rückgabe:

```
{
    "start": [""],
    "stop": [""],
    "stunden": [42, 30, 35],
    "datum": ["2020-12-20", "2020-11-20", "2020-10-20"]
}
```

Listing 142: readAlloc Rückgabe

Code:

```
//Verbindung zu Datenbank
mysqlLib.getConnection(function (err, mclient) {
    // Auslesen der Sollzeiten aus der Datenbank
    mclient.query("SELECT * from Allocated ...",
function (err, results) {
    if (err) throw err;

    var h = 0, m = 0, s = 0;
    var results = JSON.parse(JSON.stringify(results));
```



```
//neueste Zeiten holen, welche dann auch ausgegeben werden
data.start[0] = results[0].monstart;
data.stop[0] = results[0].monstop;
data.start[1] = results[0].tuestart;
data.stop[1] = results[0].tuestop;
data.start[2] = results[0].wedstart;
data.stop[2] = results[0].wedstop;
data.start[3] = results[0].thustart;
data.stop[3] = results[0].thustop;
data.start[4] = results[0].fristart;
data.stop[4] = results[0].fristop;

for (var i = 0; i < results.length; i++) {

// Sollzeit der Wochen berechnen
time =
allocDiff(results[i].monstart,results[i].monstop) +
allocDiff(results[i].tuestart,results[i].tuestop) +
allocDiff(results[i].wedstart,results[i].wedstop) +
allocDiff(results[i].thustart, results[i].thustop)+

allocDiff(results[i].fristart, results[i].fristop);

//Umrechnung von Sekunden in hh:mm:ss
h = Math.floor(time / 3600);
m = Math.floor((time % 3600) / 60);
s = (time % 3600) % 60;
data.hours[i] = bigger10(h) + ":" + bigger10(m) + ":" +
bigger10(s);
data.date[i] = formatDate(results[i].date);
}

// Rückgabe des JSON
res.status(200).json(
  data
);
```

Listing 143: Code „readAlloc“

8.5.2.6 Bearbeiten von Sollzeiten

allocEdit.js:

Dieser Call wird vom Web-Frontend verwendet, um die Sollarbeitszeiten der Mitarbeiter zu editieren. Das Web-Frontend muss für jeden Tag, Uhrzeiten übertragen. Diese werden anschließend mit dem aktuellen Datum und der korrekten User-ID in die Datenbank eingetragen. Das Datum dient dazu, um eine Historie der Sollarbeitszeiten zu erstellen.

URL: <https://wudi.serveminecraft.net:8080/allocEdit/>

Beispiel:

JSON-Übergabe:

```
{
  "start": ["07:30:00", "07:30:00", "08:00:00", "07:30:00", "07:30:00"],
  "stop": ["17:30:00", "17:30:00", "17:30:00", "17:30:00", "17:30:00"],
  "idUser": 1234
}
```

Listing 144: allocEdit Übergabe

JSON-Rückgabe:

```
{
  "message": "Writing was successful"
}
```

Listing 145: allocEdit Rückgabe

Code:

```
// Startzeiten
var datastart = '""' + dataIn.start[0] + '", "' + dataIn.start[1] +
  '", "' + dataIn.start[2] + '", "' + dataIn.start[3] +
  '", "' + dataIn.start[4];

// Stopzeiten
var datastop = '", "' + dataIn.stop[0] + '", "' + dataIn.stop[1] +
  '", "' + dataIn.stop[2] + '", "' + dataIn.stop[3] +
  '", "' + dataIn.stop[4] + '",';

// Aktuelles Datum
date = new Date();
date = formatDateLib.formatDate(date);

// Editieren der Daten in der Datenbank
mysqlLib.getConnection(function (err, mclient) {
  mclient.query("INSERT INTO Allocated ...",
    function (err, results) {
      if (err) throw err;

      res.status(200).json({
        "message": "Success!"
      });
    });
  mclient.release();
});
```

Listing 146: Code „allocEdit“



8.5.2.7 Bearbeiten von Timestamps

editStamp.js:

Falls beim Erstellen eines Timestamps etwas schief geht, besteht die Möglichkeit dies im Kalender der Web-App zu beheben. Die Daten, welche verändert werden sollen, werden an den Key gesendet und dafür bekommt man eine Message, wenn das Schreiben funktioniert hat.

URL: <https://wudi.serveminecraft.net:8080/editStamp/>

Beispiel:

JSON-Übergabe:

```
{  
    "idDaily": 34,  
    "stamp": "09:15:00",  
    "date": "2020-11-01",  
    "comment": "",  
    "fulltime": 1,  
    "User_idUser": 1234,  
    "Type_idType": 2  
}
```

Listing 147: editStamp Übergabe

JSON-Rückgabe:

```
{  
    "message": "Writing was successful",  
}
```

Listing 148: editStamp Rückgabe

Code:

```
// ID in eine Variable speichern und  
// aus dem Objekt entfernen  
var idDaily = dataIn.idDaily;  
delete dataIn.idDaily;  
  
// fulltime bool konvertieren  
dataIn.fulltime = dataIn.fulltime ? 1 : 0;  
var keys = Object.keys(dataIn);  
var val = Object.values(dataIn);  
  
mysqlLib.getConnection(function (err, mclient) {  
  
    // Ändern der Werte in der Datenbank  
    for (var i = 0; i < keys.length; i++) {
```

```

mclient.query("UPDATE Daily SET " + keys[i] + " = '" +
val[i] + "' where idDaily = '" + idDaily + "'",
function (err, results) {
    if (err) throw err;
}) ;
}

```

Listing 149: Code „editStamp“

8.5.2.8 Bearbeiten von Userdaten

editUser.js:

Mit diesem Key können die User-Daten bearbeitet werden.

URL: <https://wudi.serveminecraft.net:8080/editUser/>

Beispiel:

JSON-Übergabe:

```
{
    "idUser": 1234,
    "firstname": "John",
    "lastname": "Doe",
    "email": "johndoe@email.com",
    "joined": "2016-08-01"
}
```

Listing 150: editUser Übergabe

JSON-Rückgabe:

```
{
    "message": "Writing was successful",
}
```

Listing 151: editUser Rückgabe

Code:

```

mysqlLib.getConnection(function (err, mclient) {

    for (var i = 0; i < keys.length; i++) {
        // Checken ob Daten Zahl oder String sind,
        // und nach dem wird formatiert
        if (Number.isInteger(val[i])) {
            wanteddata = keys[i] + " = " + val[i];
        }
        else {
            wanteddata = keys[i] + " = '" + val[i] + "'";
        }
    }
})

```



```
// Überschreiben der User-Daten in der Datenbank
mclient.query("UPDATE User SET " + wanteddata + " where
idUser = '" + idUser + "'", function (err, results) {
    if (err) throw err;
}) ;
}
```

Listing 152: Code „editUser“

8.5.2.9 Berechnung diverser Zeiten

effTime.js:

Dieser Key liefert die Sollzeit, die bereits erledigte Arbeitszeit und die nicht-effiziente Arbeitszeit in einer Woche, in Sekunden.

URL: <https://wudi.serveminecraft.net:8080/effTime/>

Beispiel:

JSON-Übergabe:

```
{
    "idUser": 1234
}
```

Listing 153: effTime Übergabe

JSON-Rückgabe:

```
{
    "allocTime": 136800,
    "effTime": 98000,
    "neffTime": 14400
}
```

Listing 154: effTime Rückgabe

Code:

```
var date = new Date();
var day = date.getDay();

while (day != 1) {
    date.setDate(date.getDate() - 1);
    day = date.getDay();
}

var date2 = new Date();
date2.setDate(date.getDate() + 4); //Startdatum
dt = formatDateLib.formatDate(date); //Startdatum
dt2 = formatDateLib.formatDate(date2); //Stopptdatum

//*****
```

```
mysqlLib.getConnection(function (err, mclient) {
    // Auslesen der Sollzeiten
    mclient.query("SELECT * from Allocated ...",
    function (err, results) {
        if (err) throw err;

        // Sollzeit der Woche berechnen
        time = allocDiff(results[0].monstart, results[0].monstop) +
        allocDiff(results[0].tuestart, results[0].tuestop) +
        allocDiff(results[0].wedstart, results[0].wedstop) +
        allocDiff(results[0].thustart, results[0].thustop) +
        allocDiff(results[0].fristart, results[0].fristop);
        data.allocTime = time;

    });

    // Auslesen der Timestamps aus der Datenbank
    mclient.query("SELECT " + wanteddata + " FROM Daily ...",
    function (err, results) {
        if (err) throw err;
        var time = JSON.parse(JSON.stringify(results));

        // Berechnen der effektiven und nicht eff. Arbeitszeit
        for (var i = 0; i < time.length; i++) {

            // Startzeit der effektiven Arbeitszeit festlegen und in
            // Sekunden umrechnen
            if (token == 0 && (time[i].type == "Start")) {
                a = (time[i].stamp).split(':');
                start = (+a[0]) * 60 * 60 + (+a[1]) * 60 + (+a[2]);
                token = 1;
            } // effektive Arbeitszeit errechnen
            else if ((token == 1) && time[i].type != "Start") {
                a = (time[i].stamp).split(':');
                stop = (+a[0]) * 60 * 60 + (+a[1]) * 60 + (+a[2]);
                out = out + stop - start;
                token = 0;
            } // nicht effektive Arbeitszeit berechnen
            else if ((token == 2) && (time[i].type == "Start" ||
            time[i].type == "Stop" || time[i].type == "Pause")) {
                a = (time[i].stamp).split(':');
                stop = (+a[0]) * 60 * 60 + (+a[1]) * 60 + (+a[2]);
                nout = nout + stop - start;
                token = 0;
            }
        }
    });
});
```



```
// Startzeit der nicht effektiven Arbeitszeit festlegen
// und in Sekunden umrechnen
if (token != 2 && time[i].type != "Start" &&
time[i].type != "Stop" && time[i].type != "Pause") {
    a = (time[i].stamp).split(':');
    start = (+a[0]) * 60 * 60 + (+a[1]) * 60 + (+a[2]);
    token = 2;
}
data.effTime = out;
data.neffTime = nout;
res.status(200).json(
    data
);
} );
mclient.release();

}) ;
```

Listing 155: Code „effTime“

8.5.2.10 Erzeugen des Reports

report.js:

Dieser Key dient der Erstellung eines Reports über die Mitarbeiter. Er war schon backendseitig fertig, es traten jedoch Komplikation beim Download über das Frontend auf, wodurch dieser Key nicht vervollständigt werden konnte.

URL: <https://wudi.serveminecraft.net:8080/report/>

8.5.2.11 Löschen von Einträgen

delEntry.js:

Um bestimmte Timestamps zu löschen, wird die Timestamp-ID an diesen Key gesendet, welcher anschließend den Eintrag mit dieser ID entfernt.

URL: <https://wudi.serveminecraft.net:8080/delEntry/>

Beispiel:

JSON-Übergabe:

```
{
    "idDaily": 123
}
```

Listing 156: delEntry Übergabe

JSON-Rückgabe:

```
{
  "message": "Deletion was successful",
}
```

Listing 157: delEntry Rückgabe

Code:

```
// Datenbankaufruf zum Löschen des Stamps
mclient.query("DELETE FROM Daily WHERE idDaily = " + dataIn.idDaily,
function (err, results) {
  if (err) throw err;

  res.status(200).json(
    results
  );
}) ;
```

Listing 158: Code „delEntry“

8.5.2.12 Schreiben von Timestamps

writeTimestamp.js:

Bei diesem Key erhält das Backend Daten von den Frontends und trägt diese dann als Timestamps in die Datenbank ein.

URL: <https://wudi.serveminecraft.net:8080/writeTimestamp/>

Beispiel:

JSON-Übergabe:

```
{
  "User_idUser": 1234,
  "stamp": "7:30:00",
  "date": "2021-03-31",
  "comment": "",
  "fulltime": false,
  "Type_idType": 1
}
```

Listing 159: writeTimestamp Übergabe



Die Typen-ID (Type_idType) sagt aus, um welche Art von Stamp es sich handelt.

- 1.....Start
- 2.....Arztbesuch
- 3.....Amtsweg
- 4.....Dienstverhinderung
- 5.....Krankenstand
- 6.....Urlaub
- 7.....Pflegefreistellung
- 8.....Zeitausgleich
- 9.....Pause
- 10....Stop

JSON-Rückgabe:

```
{  
  "message": "Writing was successful",  
}
```

Listing 160: writeTimestamp Rückgabe

Code:

```
// Schreiben des Timestamps  
mclient.query("INSERT INTO Daily ...", (err, results2) => {  
  if (err) throw err;  
});  
  
// Lesen des Timestamps davor  
mclient.query("SELECT ... FROM Daily ...", (err, results)  
=> {  
  if (err) throw err;  
  
  // Überprüfen ob die vorherige Abwesenheit  
  // bis zum nächsten Stamp zu tracken ist  
  if (results[results.length - 1] != undefined) {  
    if ((results[results.length - 1].fulltime == 1) &&  
        ((results[results.length - 1].Type_idType > 1) &&  
        (results[results.length - 1].Type_idType < 9))) {  
      date = results[results.length - 1].date;  
      type = results[results.length - 1].Type_idType;  
    }  
  }  
  
  // Falls bis zum nächsten Stamp zu tracken war  
  if (type != 0) {  
    var weekday = ["sun", "mon", "tue", "wed", "thu",  
      "fri", "sat"];  
  }  
});
```

```
while (date != dataIn.date) {  
    var day = weekday[new Date(date).getDay()];  
  
    if ((day != "sat") && (day != "sun")) {  
        var wanteddata = day + "start start, " +  
            day + "stop stop";  
  
        dt[test3] = date;  
        test3++;  
  
        // Auslesen der Sollzeiten für jeden Tag  
        mclient.query("SELECT ... FROM Allocated  
        ...", (err, results2) => {  
            if (err) throw err;  
  
            // Überspringen der ersten Startzeit  
            if (counter != 0) {  
  
                // Schreiben der Startzeit der  
                // Abwesenheiten für jeden Tag  
                mclient.query("INSERT INTO Daily  
                ...", (err, results3) => {  
                    if (err) throw err;  
  
                }) ;  
            }  
  
            // Schreiben der Stopzeit der  
            // Abwesenheiten für jeden Tag  
            mclient.query("INSERT INTO Daily ...",  
                (err, results4s) => {  
                    if (err) throw err;  
                }) ;  
  
            counter++;  
        }) ;  
    }  
  
    // Datum um einen Tag erhöhen  
    var test = new Date(date);  
    test.setDate(test.getDate() + 1);  
    date = formatDateLib.formatDate(test)  
}
```

Listing 161: Code „writeTimestamp“



8.5.2.13 Registrierung neuer User

reqUser.js:

Falls ein User noch nicht in der Datenbank eingetragen wurde, kann man diesen mit diesem Key registrieren. Es wird auch automatisch eine Sollzeit festgelegt, welche empfohlen wird zu ändern

URL: <https://wudi.serveminecraft.net:8080/regUser/>

Beispiel:

JSON-Übergabe:

```
{  
    "idUser": 1234,  
    "firstname": "John",  
    "lastname": "Doe",  
    "email": "johndoe@email.com",  
    "joined": "2",  
    "apikey": "7w3gcrb8i34hi",  
    "pass": "jzj5s124egr847hti235ruzti74snu"  
}
```

Listing 162: regUser Übergabe

JSON-Rückgabe:

```
{  
    "message": "Writing was successful"  
}
```

Listing 163: regUser Rückgabe

Code:

```
// Schreiben der User-Daten in die Datenbank  
mcclient.query("INSERT INTO User " + keys + " VALUES " + val, (err,  
results) => {  
    if (err) throw err;  
  
});  
  
// Setzen von Standard-Sollzeiten, welche zu ändern sind  
var data = '"00:00:00","00:00:00","00:00:00","00:00:00","00:00:00"';  
var date = new Date();  
date.setDate(date.getDate() + 1);  
date = formatDateLib.formatDate(date);  
  
// Schreiben der Standard-Sollzeiten in die Datenbank  
mcclient.query("INSERT INTO Allocated ...", function (err, results) {  
    if (err) throw err;
```

```

        res.status(200).json({
            message: 'Writing was successful!'
        });
    });
mclient.release();

```

Listing 164: Code „regUser“

8.5.2.14 Lesen der Tracking-Methoden

readBT.js:

Mit diesem Key können die Daten, der in der Datenbank eingetragenen Bluetooth-Beacons, abgefragt werden.

URL: <https://wudi.serveminecraft.net:8080/readBT/>

Beispiel:

JSON-Übergabe:

Bei diesem Key werden nur die Authentifizierungs-Daten benötigt.

JSON-Rückgabe:

```
{
    "idBT": 1,
    "UUID": "E2C56DB5-DFFB-48D2-B060-D0F5A71096E0",
    "minor": 1,
    "major": 1,
    "radius": 100
}
```

Listing 165: readBT Rückgabe

Code:

```

mysqlLib.getConnection(function (err, mclient) {
    // Auslesen der Bluetooth-Beacon-Daten aus der Datenbank
    mclient.query("SELECT * from BT", function (err, results) {
        if (err) throw err;

        //Rückgabe des JSON
        res.status(200).json(
            results
        );
    });
    mclient.release();
});

```

Listing 166: Code „readBT“

**readGF.js:**

Mit diesem Key können die Daten der in der Datenbank eingetragenen Geofences abgefragt werden.

URL: <https://wudi.serveminecraft.net:8080/readBT/>

Beispiel:

JSON-Übergabe:

Bei diesem Key werden nur die Authentifizierungs-Daten benötigt.

JSON-Rückgabe:

```
{  
    "idGF": 1,  
    "identifier": "Ernstbrunn",  
    "latitude": 48.5259898,  
    "longitude": 16.3594227,  
    "radius": 100  
}
```

Listing 167: readGF Rückgabe

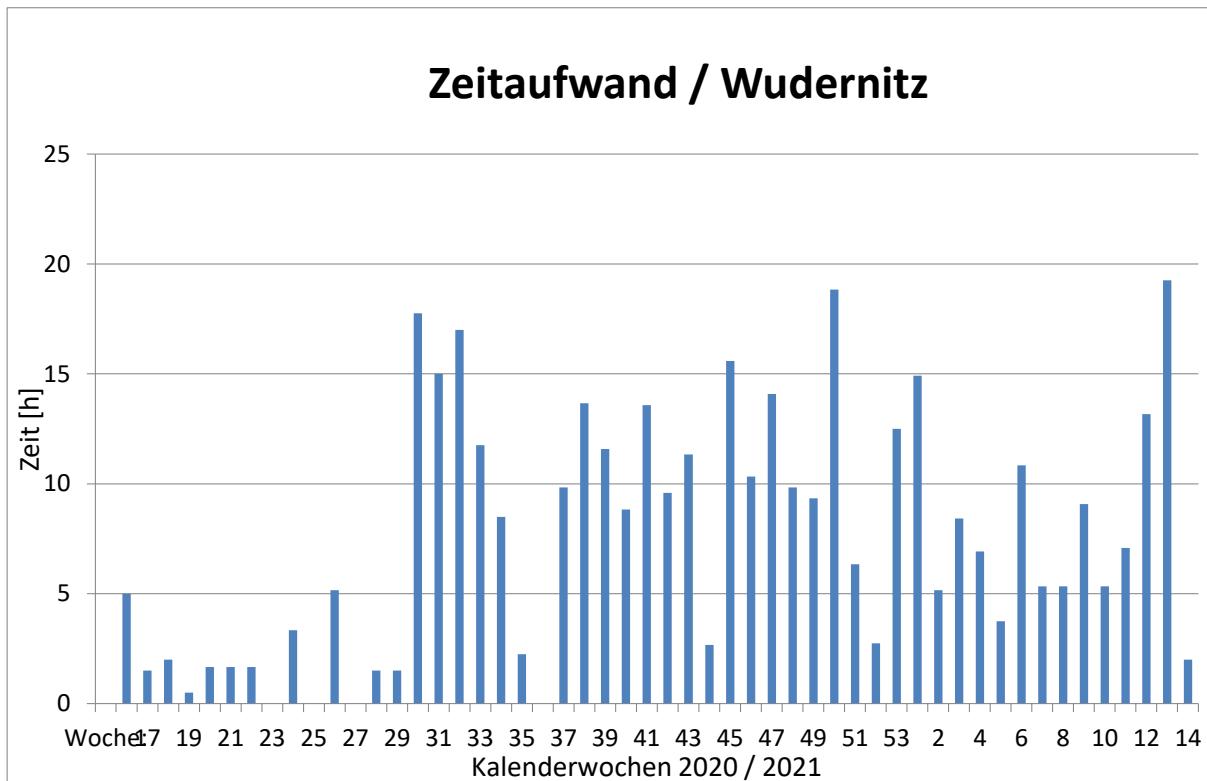
Code:

```
mysqlLib.getConnection(function (err, mclient) {  
    // Auslesen der Geofence-Daten aus der Datenbank  
    mclient.query("SELECT * from GF", function (err, results) {  
        if (err) throw err;  
  
        //Rückgabe des JSON  
        res.status(200).json(  
            results  
        );  
    });  
    mclient.release();  
});
```

Listing 168: Code „readGF“

9 Zeiterfassung

9.1 Wudernitz

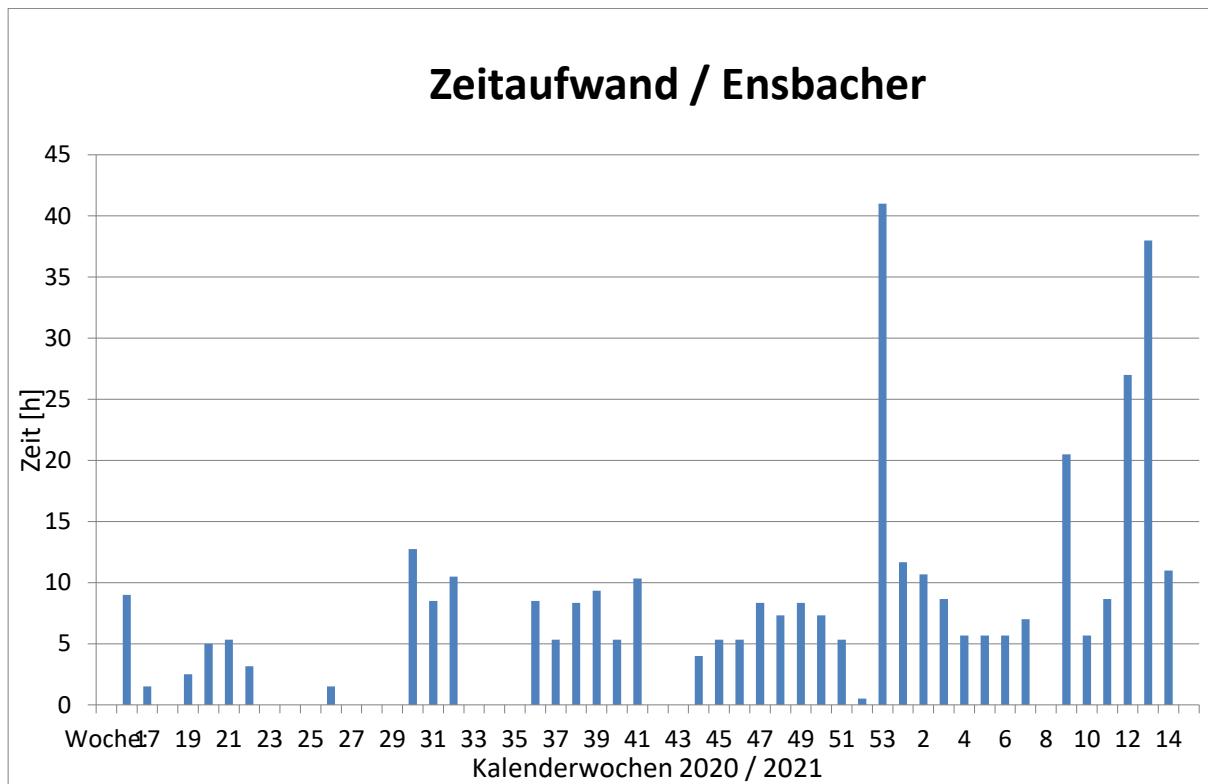


Schule: 119,10 h

Freizeit: 286,89 h

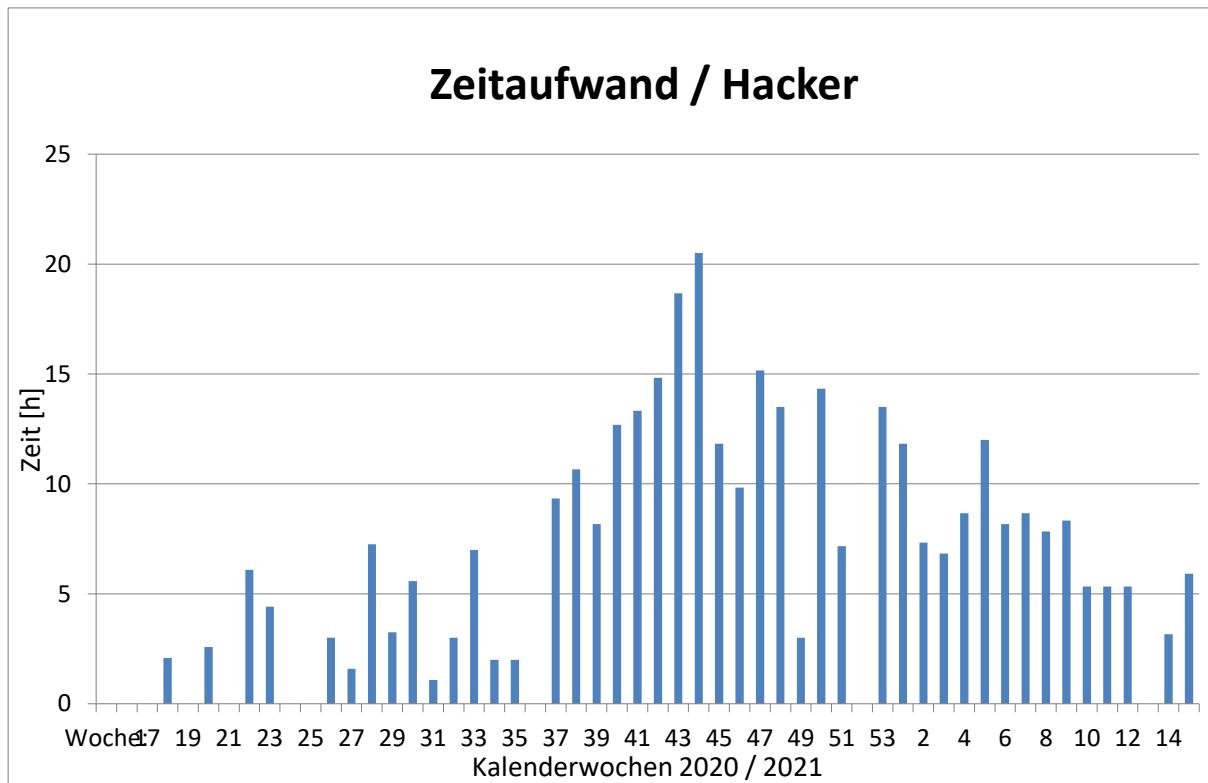
Gesamt: 405,99 h

9.2 Ensbacher



Schule: 109,27 h
Freizeit: 256,31 h
Gesamt: 365,58 h

9.3 Hacker

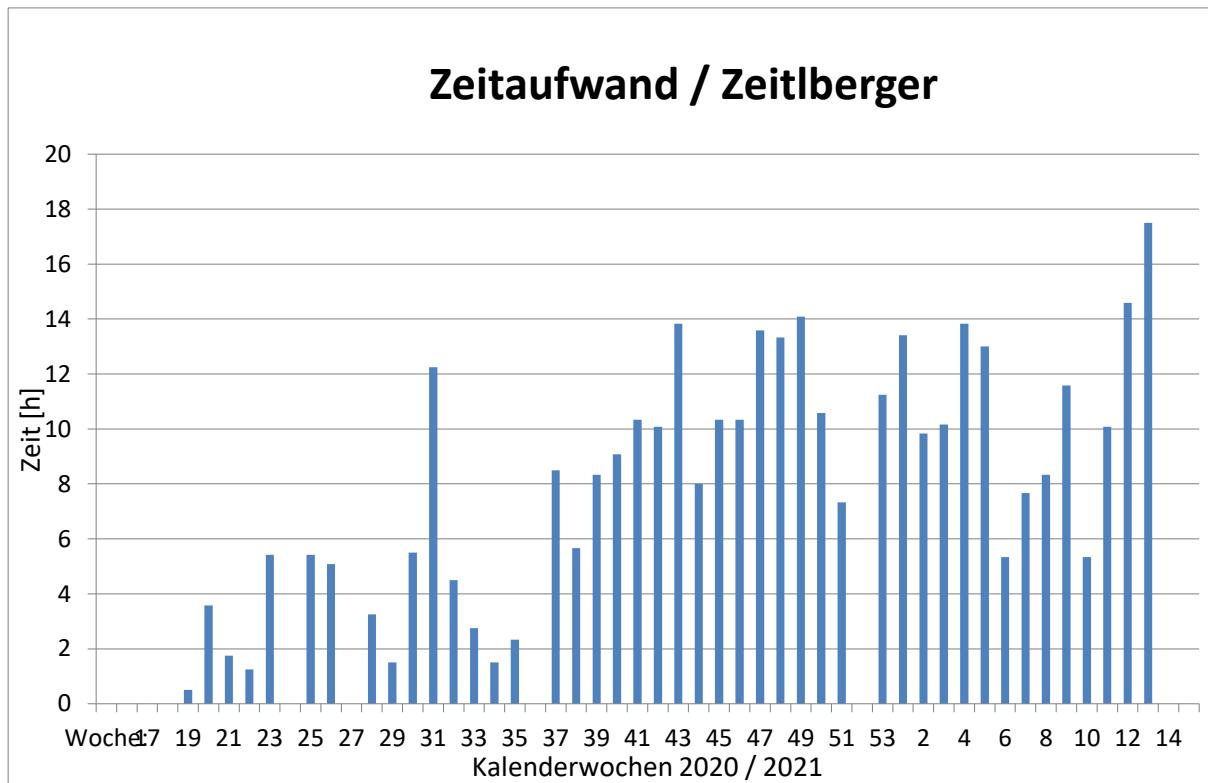


Schule: 106,35 h

Freizeit: 241,42 h

Gesamt: 352,18 h

9.4 Zeitberger



Schule: 108,35 h

Freizeit: 253,56 h

Gesamt: 361,92 h

10 Literaturverzeichnis

[1] Apple ID erstellen

<https://support.apple.com/de-at/HT204316>

(Letzter Aufruf: 09.02.2021)

[2] Apples Features (Capabilities)

<https://developer.apple.com/de/support/app-capabilities/>

[3] Apple Developer Account erstellen

<https://developer.apple.com/programs/enroll/>

(Letzter Aufruf: 09.02.2021)

[4] Xcode

<https://developer.apple.com/xcode/>

(Letzter Aufruf: 09.02.2021)

[5] Objective-C

<https://de.wikipedia.org/wiki/Objective-C>

(Letzter Aufruf: 09.02.2021)

[6] Lebenszyklus einer iOS-App

<https://hackernoon.com/application-life-cycle-in-ios-12b6ba6af78b>

(Letzter Aufruf: 11.02.2021)

[7] AppDelegate

<https://www.hackingwithswift.com/example-code/language/what-does-the-appdelegate-class-do>

(Letzter Aufruf: 11.02.2021)

[8] SceneDelegate

<https://medium.com/@kalyan.parise/understanding-scene-delegate-app-delegate-7503d48c5445>

(Letzter Aufruf: 11.02.2021)

[9] Model-View-Controller

<https://www.raywenderlich.com/1000705-model-view-controller-mvc-in-ios-a-modern-approach>

(Letzter Aufruf: 11.02.2021)

[10] ViewController

https://developer.apple.com/documentation/uikit/view_controllers

(Letzter Aufruf: 11.02.2021)

[11] Storyboards vs. Programmatically

<https://medium.com/@matlyles/storyboards-vs-programmatic-ios-development-40c5b5907f85>

(Letzter Aufruf: 11.02.2021)

[12] UI-Elemente

https://www.tutorialspoint.com/ios/ios_ui_elements.htm

(Letzter Aufruf: 11.02.2021)

[13] Swift

<https://developer.apple.com/swift/>

(Letzter Aufruf: 11.02.2021)

[14] Vorteile Swift

<https://mlsdev.com/blog/swift-vs-objective-c>

(Letzter Aufruf: 11.02.2021)



[15] Auto Layout

<https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutoLayoutPG/index.html>

(Letzter Aufruf: 12.02.2021)

[16] UIKit

<https://developer.apple.com/documentation/uikit>

(Letzter Aufruf: 12.02.2021)

[17] SwiftUI

<https://developer.apple.com/xcode/swiftui/>

(Letzter Aufruf: 12.02.2021)

[18] Funktionen

<https://docs.swift.org/swift-book/LanguageGuide/Functions.html>

(Letzter Aufruf: 12.02.2021)

[19] API-Calls

<https://www.freecodecamp.org/news/how-to-make-your-first-api-call-in-swift/>

(Letzter Aufruf: 12.02.2021)

[20] NFC Tools

<https://apps.apple.com/us/app/nfc-tools/id1252962749>

(Letzter Aufruf: 19.02.2021)

[21] Activity Lifecycle

<https://developer.android.com/guide/components/activities/activity-lifecycle#alc>

(Letzter Aufruf: 25.03.2021)

[22] Lebenszyklus einer iOS-App

https://hackernoon.com/images/1*Iwe53lfhwwPO3_K1K1Gliq.gif

(Letzter Aufruf: 25.03.2021)

[23] Node.js

<https://nodejs.org/de>

(Letzter Aufruf: 25.03.2021)

[24] Vue CLI

<https://cli.vuejs.org/>

(Letzter Aufruf: 25.03.2021)

[25] Vue Lebenszyklus

<https://vuejs.org/v2/guide/instance.html>

(Letzter Aufruf: 25.03.2021)

[26] Vue.js

<https://vuejs.org/v2/guide>

(Letzter Aufruf: 25.03.2021)

[27] Vuetify Erklärung

<https://vuetifyjs.com/en/introduction/why-vuetify>

(Letzter Aufruf: 25.03.2021)

[28] Vuetify Bild

<https://vuetifyjs.com/de>

(Letzter Aufruf: 25.03.2021)

[29] Vuex Store

<https://vuex.vuejs.org>

(Letzter Aufruf: 25.03.2021)

[30] Vuei18n

<https://kazupon.github.io/vue-i18n>

(Letzter Aufruf: 25.03.2021)

[31] CORS Bild

<https://developer.mozilla.org/de/docs/Web/HTTP/CORS>

(Letzter Aufruf: 25.03.2021)

[32] Valet (Library)

<https://github.com/square/Valet>

(Letzter Aufruf: 29.03.2021)

[33] TinyConstraints (Library)

<https://github.com/roberthein/TinyConstraints>

(Letzter Aufruf: 29.03.2021)

[34] Charts (Library)

<https://github.com/danielgindi/Charts>

(Letzter Aufruf: 29.03.2021)

[35] Cocoapods

<https://cocoapods.org/>

(Letzter Aufruf: 29.03.2021)

[36] AppDelegate

<https://learnappmaking.com/scene-delegate-app-delegate-xcode-11-ios-13/>

(Letzter Aufruf: 30.03.2021)

[37] Swift Lazy

<https://www.hackingwithswift.com/example-code/language/what-are-lazy-variables>

(Letzter Aufruf: 30.03.2021)

[38] SceneDelegate

<https://learnappmaking.com/scene-delegate-app-delegate-xcode-11-ios-13/>

(Letzter Aufruf: 31.03.2021)

[39] RootViewController

<https://developer.apple.com/documentation/uikit/uiwindow/1621581-rootviewcontroller>

(Letzter Aufruf: 31.03.2021)

[40] UserDefaults

<https://developer.apple.com/documentation/foundation/userdefaults>

(Letzter Aufruf: 31.03.2021)

[41] Keychain

https://developer.apple.com/documentation/security/keychain_services

(Letzter Aufruf: 31.03.2021)

[42] Bluetooth Beacons Beispiel

<https://kontakt.io/wp-content/uploads/2014/08/aug-24-1-1024x614.jpg>

(Letzter Aufruf: 02.04.2021)

[43] Bluetooth Beacons Parameter

<https://kontakt.io/blog/beacon-id-strategy-guide-quick-deployment/>

(Letzter Aufruf: 02.04.2021)

[44] SpringBoard

<https://en.wikipedia.org/wiki/SpringBoard>

(Letzter Aufruf: 04.04.2021)



[45] Google Maps

<https://www.google.at/maps>

(Letzter Aufruf: 04.04.2021)

[46] Apple Human Interface Guidelines

<https://developer.apple.com/design/human-interface-guidelines/>

(Letzter Aufruf: 04.04.2021)

[47] How to Install Apache

<https://phoenixnap.com/kb/how-to-install-apache-web-server-on-ubuntu-18-04>

(Letzter Aufruf: 04.04.2021)

[48] RESTful-API

https://de.wikipedia.org/wiki/Representational_State_Transfer

(Letzter Aufruf: 04.04.2021)

[49] Let's Encrypt

<https://www.pragmaticlinux.com/2020/10/install-a-free-lets-encrypt-ssl-certificate-on-debian-10/>

(Letzter Aufruf: 04.04.2021)

[50] Android Build Prozess

<https://developer.android.com/studio/build#build-process>

(Letzter Aufruf: 05.04.2021)

[51] Type Annotation

<https://www.hackingwithswift.com/sixty/1/7/type-annotations>

(Letzter Aufruf: 05.04.2021)

[52] CoreData Context

<https://developer.apple.com/documentation/coredata/nsmanagedobjectcontext>

(Letzter Aufruf: 05.04.2021)

[53] NFC-Tools Android

https://play.google.com/store/apps/details?id=com.wakdev.wdnfc&hl=de_AT&gl=US

(Letzter Aufruf: 05.04.2021)

[301] Balsamiq-Logo

<https://www.capterra.com.de/software/145723/balsamiq-mockups>

(Letzter Aufruf: 06.04.2021)

[302] Adobe Xd

<https://www.adobe.com/at/products/xd.html>

(Letzter Aufruf: 06.04.2021)

[303] Sketch

<https://www.sketch.com/>

(Letzter Aufruf: 06.04.2021)

[304] Overflow

<https://overflow.io/>

(Letzter Aufruf: 06.04.2021)

[305] Visual Studio Code

<https://code.visualstudio.com/>

(Letzter Aufruf: 06.04.2021)

[306] Visual Studio Community

<https://visualstudio.microsoft.com/de/downloads/>

(Letzter Aufruf: 06.04.2021)

[307] Xcode

<https://developer.apple.com/xcode/>

(Letzter Aufruf: 06.04.2021)

[308] Android Studio

<https://developer.android.com/studio>

(Letzter Aufruf: 06.04.2021)

[309] Git

<https://git-scm.com/>

(Letzter Aufruf: 06.04.2021)

[310] GitHub-Desktop

<https://desktop.github.com/>

(Letzter Aufruf: 06.04.2021)

[311] Tortoise-Git

<https://tortoisegit.org/>

(Letzter Aufruf: 06.04.2021)

[312] node.js

<https://nodejs.org/en/>

(Letzter Aufruf: 06.04.2021)

[313] Google Drive

<https://www.google.at/drive/about.html>

(Letzter Aufruf: 06.04.2021)

[314] Google Backup & Sync

(Letzter Aufruf: 06.04.2021)

[315] Gimp

<https://www.gimp.org/>

(Letzter Aufruf: 06.04.2021)

[316] App Icon Resizer

<https://apps.apple.com/us/app/app-icon-resizer-air/id1013592989?mt=12>

(Letzter Aufruf: 06.04.2021)

[317] Google Chrome

<https://www.google.com/intl/de/chrome/>

(Letzter Aufruf: 06.04.2021)

[318] Mozilla Firefox

<https://www.mozilla.org/de/firefox/new/>

(Letzter Aufruf: 06.04.2021)

[319] Microsoft Office

<https://www.microsoft.com/de-at/microsoft-365/free-office-online-for-the-web>

(Letzter Aufruf: 06.04.2021)



11 Abbildungsverzeichnis

Abbildung 1: Balsamiq [301]	31
Abbildung 2: Adobe Xd [302]	31
Abbildung 3: Sketch [303]	31
Abbildung 4: Overflow [304]	31
Abbildung 5: Visual Studio Code [305]	31
Abbildung 6: Visual Studio Community [306]	31
Abbildung 7: Xcode [307]	32
Abbildung 8: Android Studio [308]	32
Abbildung 9: Git [309]	32
Abbildung 10: Github Desktop [310]	32
Abbildung 11: Tortoise Git [311]	32
Abbildung 12: node.js [312]	33
Abbildung 13: Google Drive [313]	33
Abbildung 14: Google Backup & Sync [314]	33
Abbildung 15: Gimp [315]	33
Abbildung 16: App Icon Resizer [316]	33
Abbildung 17: Google Chrome [317]	34
Abbildung 18: Firefox [318]	34
Abbildung 19: Microsoft Excel [319]	34
Abbildung 20: Microsoft Word [319]	34
Abbildung 21: Microsoft PowerPoint [319]	34
Abbildung 22: Projektplan	35
Abbildung 23: Lebenszyklus einer iOS-App [22]	36
Abbildung 24: Xcode Startscreen	45
Abbildung 25: Xcode Auswahl iOS-App	46
Abbildung 26: Xcode Projektoptionen	46
Abbildung 27: Xcode Projekt speichern	47
Abbildung 28: Xcode neues Projekt fertig	47
Abbildung 29: Xcode Hauptansicht	48
Abbildung 30: Xcode Projekteigenschaften	49
Abbildung 31: Xcode Capabilities	49
Abbildung 32: Xcode Project-Info	50
Abbildung 33: Xcode Info.plist	50
Abbildung 34: iOS-App mit einem Button in der Mitte des Views	52
Abbildung 35: App deployen	52
Abbildung 36: Auswahl Gerät zum Deployen	52
Abbildung 37: Xcode Capabilities	56
Abbildung 38: Xcode NFC Capability	57
Abbildung 39: NFC Property (Info.plist-File)	57
Abbildung 40: Bluetooth Beacons Parameter Beispiel [42]	58
Abbildung 41: Bluetooth-Beacons Properties (Info.plist-Datei)	59
Abbildung 42: Geofencing Properties (Info.plist-Datei)	61
Abbildung 43: Xcode CoreData	63
Abbildung 44: Xcode CoreData Model-File	64

Abbildung 45: CoreData Codegen.....	64
Abbildung 46: Xcode NSManagedObject erstellen	65
Abbildung 47: Xcode CoreData Model-Files speichern.....	65
Abbildung 48: Xcode CoreData 2 neue Files	66
Abbildung 49: Launch Screen.....	70
Abbildung 50: Login Screen	70
Abbildung 51: Home Screen.....	71
Abbildung 52: Abwesenheiten (Liste)	71
Abbildung 53: Neue Abwesenheit hinzufügen	72
Abbildung 54: Wochenüberblick	72
Abbildung 55: Tagesüberblick	73
Abbildung 56: NFC Reader	73
Abbildung 57: Tracking Verlauf.....	74
Abbildung 58: Einstellungen	74
Abbildung 59: Valet Clone Code Link	75
Abbildung 60: Swift Package Manager	76
Abbildung 61: Swift Package Dependecy hinzufügen	76
Abbildung 62: Valet als Swift Package Dependecy.....	77
Abbildung 63: Terminal in den Projektordner navigieren.....	77
Abbildung 64: Terminal Pod-File erstellen	77
Abbildung 65: erfolgreich erstelltes Pod-File im Projektverzeichnis	77
Abbildung 66: TabBar der iOS-App	81
Abbildung 67: Username-Text Field angepasst	81
Abbildung 68: Login Spinner.....	82
Abbildung 69: Sollarbeitszeit und Wochensollarbeitszeit	85
Abbildung 70: Arbeits- und Pausezeit.....	87
Abbildung 71: Abwesenheit (Bis Timetracking).....	88
Abbildung 72: Abwesenheit (Ganztägig).....	88
Abbildung 73: Abwesenheit (Geplant)	89
Abbildung 74: Abwesenheitsgrund auswählen	90
Abbildung 75: PopUp im Überblick	93
Abbildung 76: Löschen eines Events im Überblick	94
Abbildung 77: ActionSheet im Tracking Verlauf.....	95
Abbildung 78: Löschen eines Eintrags im Tracking Verlauf	97
Abbildung 79: Notification Beim Eintritt in den Geofencing Bereich	102
Abbildung 80: Activity Lifecycle [21].....	106
Abbildung 81: Android Studio Start Screen.....	111
Abbildung 82: Activity Template Auswahl	112
Abbildung 83: Projekt Parameter Fenster	113
Abbildung 84: Android API-Aufteilung.....	114
Abbildung 85: Android Studio IDE	115
Abbildung 86: Android Studio SDK Manager	116
Abbildung 87: Android Studio AVD Manager	117
Abbildung 88: Layout Editor	118
Abbildung 89: Android Gradle Files	123
Abbildung 90: Android - Build Prozess [50].....	126



Abbildung 91: Android Emulator Toolbar	127
Abbildung 92: Android Emulator	127
Abbildung 93: Erweiterte Toolbar des Android Emulators.....	128
Abbildung 94: Login Screen	137
Abbildung 95: Home Screen.....	137
Abbildung 96: Abwesenheiten Dialog	138
Abbildung 97: NFC Lese Dialog	138
Abbildung 98: NFC Screen.....	138
Abbildung 99: Android Bottom Navigationbar	145
Abbildung 100: cmd.....	157
Abbildung 101: VSCode	158
Abbildung 102: VSCode Konsole	159
Abbildung 103: VSCode Erweiterungen	160
Abbildung 104: Suchfeld.....	160
Abbildung 105: Vetur	161
Abbildung 106: Vue VSCode Snippets	161
Abbildung 107: Prettier	162
Abbildung 108: Highligth Line	162
Abbildung 109: Bracket Pair Colorizer 2	163
Abbildung 110: Better Comment.....	163
Abbildung 111: IntelliSense CSS	164
Abbildung 112: JavaScript Snippets	164
Abbildung 113: VSCode-Icons.....	165
Abbildung 114: Flat UI	165
Abbildung 115: Lebenszyklus Vue [25].....	166
Abbildung 116: Vuetify [28].....	171
Abbildung 117: Vuex [29]	172
Abbildung 118: Vue I18n [30]	173
Abbildung 119: VUE CLI Projekterstellung	174
Abbildung 120: Projektauswahl	174
Abbildung 121: Featureauswahl	174
Abbildung 122: Vue Versionsauswahl	175
Abbildung 123: History mode.....	175
Abbildung 124: Linter und Formatter-Auswahl.....	175
Abbildung 125: Lintingauswahl	175
Abbildung 126: Einstellungsfile.....	175
Abbildung 127: Einstellungen speichern.....	175
Abbildung 128: Projektgenerierung.....	176
Abbildung 129: Projektgenerierung abgeschlossen	177
Abbildung 130: npm run serve	178
Abbildung 131: CORS [31]	179
Abbildung 132: CORS Policy Browser.....	179
Abbildung 133: CORS Policy	180
Abbildung 134: cmd Chrome öffnen	180
Abbildung 135: Sidebar - Admin	182

Abbildung 136: Sidebar - User.....	182
Abbildung 137: Loginseite	183
Abbildung 138: Loginseite - ladend	183
Abbildung 139: Loginseite - Fehlerfall.....	184
Abbildung 140: Übersichtsseite - Admin	184
Abbildung 141: Übersichtsseite - User.....	185
Abbildung 142: Übersichtsseite - ladend	185
Abbildung 143: Übersichtsseite - Fehlerfall.....	186
Abbildung 144: Kalenderseite.....	186
Abbildung 145: Kalenderseite - ladend	187
Abbildung 146: Kalenderseite - Fehlerfall	187
Abbildung 147: Mitarbeiterseite	188
Abbildung 148: Mitarbeiter - bearbeiten.....	188
Abbildung 149: Mitarbeiters - Historie.....	189
Abbildung 150: Mitarbeiterseite - gespeichert.....	189
Abbildung 151: Mitarbeiterseite - ladend	190
Abbildung 152: Mitarbeiterseite – Fehlerfall.....	190
Abbildung 153: Einstellungsseite - Admin.....	191
Abbildung 154: Einstellungsseite - User	191
Abbildung 155: Einstellungsseite - Sprache auswählen.....	192
Abbildung 156: Einstellungsseite - Report erstellen	192
Abbildung 157: Einstellungsseite - Report Fehlerfall.....	193
Abbildung 158: Ordnerverzeichnis 1.....	194
Abbildung 159: Ordnerverzeichnis 2.....	194
Abbildung 160: Ordnerverzeichnis 3.....	194
Abbildung 161: Loginseite - Erklärung	208
Abbildung 162: Übersichtsseite - Erklärung	208
Abbildung 163: Karte - Erklärung.....	209
Abbildung 164: Kalenderseite - Erklärung.....	209
Abbildung 165: Kalenderseite - Floating Action Button	209
Abbildung 166: Kalenderseite - Zeiteintrag erstellen.....	210
Abbildung 167: Kalenderseite - Abwesenheit erstellen	210
Abbildung 168: Kalenderseite - Eintrag auswählen.....	211
Abbildung 169: Kalenderseite - Eintrag Aktionen.....	211
Abbildung 170: Kalenderseite - Eintrag bearbeiten.....	212
Abbildung 171: Kalenderseite - Eintrag löschen	212
Abbildung 172: Mitarbeiterseite - Erklärung	213
Abbildung 173: Einstellungsseite - Erklärung.....	214
Abbildung 174: App mit npm builden	215
Abbildung 175: Buildingprozess abgeschlossen	215
Abbildung 176: dist-Ordner.....	215
Abbildung 177: VS UI	216
Abbildung 178: MySQL Workbench Editor.....	218
Abbildung 179: MySQL Code-Export.....	218
Abbildung 180: XAMPP UI	219
Abbildung 181: Postman UI.....	220



Abbildung 182: FileZilla UI.....	220
Abbildung 183: PuTTY Config und CMD	221
Abbildung 184: Node.js Command Line	222
Abbildung 185: Apache Installation	223
Abbildung 186: MySQL Configuration.....	224
Abbildung 187: MySQL Encryption.....	225

12 Programm-Listing

Listing 1: Button Constraints.....	39
Listing 2: Swift Variablen	40
Listing 3: Swift Konstanten	41
Listing 4: Swift If-Bedingung	41
Listing 5: Swift for-Schleife	41
Listing 6: Swift while-Schleife	42
Listing 7: Swift repeat-Schleife	42
Listing 8: Swift Struktur.....	42
Listing 9: Swift Klasse	43
Listing 10: Swift Funktion.....	43
Listing 11: Swift Optionals	43
Listing 12: Swift Optionals unwrappen.....	44
Listing 13: Swift Optionals sicher unwrappen	44
Listing 14: Einfaches UI.....	51
Listing 15: Model Struktur.....	53
Listing 16: API-Call GET.....	54
Listing 17: API-Call POST	55
Listing 18: NFC setup	58
Listing 19: Bluetooth-Beacons Setup.....	60
Listing 20: Geofencing Setup.....	62
Listing 21: User Defaults erstellen	62
Listing 22: TrackingHistory+CoreDataClass.swift	66
Listing 23: TrackingHistory+CoreDataProperties.swift	66
Listing 24: AppDelegate.swift persistentContainer initialisieren	66
Listing 25: AppDelegate.swift saveContext()	67
Listing 26: AppDelegate.swift fetchTrackingHistory()	67
Listing 27: Erstellen des Context (CoreData)	67
Listing 28: Löschen eines Elements aus dem CoreData Stack.....	68
Listing 29: Zugriff auf saveContext()	68
Listing 30: Zugriff auf fetchTrackingHistory()	68
Listing 31: Keychain Valet erstellen	68
Listing 32: Keychain.swift iCloud Valet erstellen	68
Listing 33: Schreiben auf Valet	69
Listing 34: Lesen von Valet	69
Listing 35: Podfile	78

Listing 36: AppDelegate.swift Notifications aktivieren	78
Listing 37: SceneDelegate.swift scene()	79
Listing 38: Extensions.swift confTabBar()	80
Listing 39: LoginViewController.swift Username Text Field anpassen in setupLogin()	81
Listing 40: LoginViewController.swift Konfiguration des Username Text Fields in setupLogin()	81
.....	
Listing 41: Spinner konfigurieren	82
Listing 42: Spinner starten und stoppen	82
Listing 43: UserService.swift setPassword()	83
Listing 44: UserService.swift checkRedmineAuth()	84
Listing 45: HomeViewController.swift AllocTime und EffTime Service in viewWillAppear()	85
Listing 46: Timer.swift startTimeTracking()	85
Listing 47: Timer.swift stopTimeTracking()	86
Listing 48: Timer.swift convWorkTime()	86
Listing 49: Timer.swift convPauseTime()	86
Listing 50: HomeViewController.swift confUpdateLabelsByTimer()	87
Listing 51: HomeViewController.swift chooseTimerModeToDisplay()	88
Listing 52: ListAbsenceViewController.swift Abwesenheiten laden in viewWillAppear()	88
Listing 53: ListAbsenceTableViewController.swift layoutSubviews()	90
Listing 54: AbsenceViewController.swift Abwesenheitsgründe Array	90
Listing 55: AbsenceViewController.swift saveAbsence()	91
Listing 56: AbsenceViewController.swift addAbsence()	92
Listing 57: OverviewController.swift Überblicksdaten des ausgewählten Tages abfragen in viewDidLoad()	93
Listing 58: OverviewDataTableViewController.swift Einträge aus Tagesüberblick löschen	94
Listing 59: Globals.swift resultNFC Struktur	94
Listing 60: NFC.swift storeNFCScanInCoreData()	95
Listing 61: TrackingHistoryViewController.swift showActionsheet()	96
Listing 62: TrackingHistoryTableViewController.swift Element aus CoreData Stack löschen	96
Listing 63: SettingsViewController.swift tappedLogout()	97
Listing 64: SettingsViewController.swift logout()	97
Listing 65: Globals.swift Schlüssel für User Defaults	98
Listing 66: SettingsViewController.swift Switch Tag zuweisen	98
Listing 67: SettingsViewController.swift User Defaults je nach Switch Tag setzen	98
Listing 68: SettingsViewController.swift Zuweisung der Werte für die Switches in Einstellungen	98
Listing 69: HomeViewController.swift confTrackingMethods()	99
Listing 70: HomeViewController.swift locationManager initialisieren	99
Listing 71: BluetoothBeacons.swift confBluetoothBeacons()	99
Listing 72: BluetoothBeacons.swift Bluetooth Beacons finden	100
Listing 73: Geofence.swift confGeofence()	101
Listing 74: Geofence.swift Geofencing Bereich betreten	101
Listing 75: Geofence.swift Geofencing Bereich verlassen	102
Listing 76: Extensions.swift setLoginTextFieldStyle()	103
Listing 77: Extensions.swift addLoadingScreen()	103
Listing 78: Loading Spinner erstellen, starten und stoppen	103



Listing 79: Extensions.swift showNotification()	104
Listing 80: Aufruf showNotification()	104
Listing 81: Linear Layout XML File	108
Listing 82: Constraint Layout XML File	109
Listing 83: Kotlin Variablen Beispiel	110
Listing 84: Kotlin Konstanten Beispiel	111
Listing 85: Menü Item Farb wähler XML File	119
Listing 86: Navigation zwischen den Einzelnen Fragments der App XML File	122
Listing 87: Projekt Gradle File	124
Listing 88: Module Gradle File	125
Listing 89: Entwickleroptionen USB-Debugging	129
Listing 90: API-Call Call Data Klasse Beispiel	130
Listing 91: API-Call Response Data Klasse Beispiel	130
Listing 92: Asynchroner API-Call	131
Listing 93: Interface und Verbindungsstartfunktionen	132
Listing 94: SSL Zertifikat CA vertrauenswürdig machen	133
Listing 95: Foreground Dispatch: MainActivity.kt	134
Listing 96: Verarbeitung des Gescannten NFC Tags: MainActivity.kt	136
Listing 97: Einstellungen Screen	139
Listing 98: Login Activity Login.kt	143
Listing 99: Bottom Navigationbar Elements bottom_nav_menu.xml	144
Listing 100: Bottom Navigationbar erzeugen MainActivity.kt	145
Listing 101: Android Timetracking Fragment Layout fragment_timetracking.xml	150
Listing 102: Variablen definition TimeTrackingFragment.kt	150
Listing 103: Buttonhandler TimeTrackingFragment.kt	151
Listing 104: Absence Button TimeTrackingFragment.kt	151
Listing 105: Trackingdatenspeicherung TimeTrackingFragment.kt	152
Listing 106: Picker Dialog AbsenceDialog.kt	155
Listing 107: Asynch & await-Code	167
Listing 108: Callbacks in JavaScript	168
Listing 109: Zeitbedingter Code	168
Listing 110: This-Zeiger	169
Listing 111: Vue-Komponente	169
Listing 112: Double Curly Syntax	170
Listing 113: Single File Komponenten	170
Listing 114: Navigation Guards	171
Listing 115: Vuex Store	173
Listing 116: index.html-File	195
Listing 117: main.js-File	196
Listing 118: vuetify.js-File	197
Listing 119: index.js-Routerfile	200
Listing 120: App.vue-Komponente	200
Listing 121: store.js-File	201
Listing 122: actions.js-Authentifizierung	205
Listing 123: AppCalendar.vue-Skript	207

Listing 124: JSON Beispiel	223
Listing 125: Node Server	226
Listing 126: Authentifizierung Übergabe	227
Listing 127: readOv Übergabe	227
Listing 128: readOv Rückgabe	227
Listing 129: Code "readOv"	229
Listing 130: readOvData Übergabe	229
Listing 131: readOvData Rückgabe	229
Listing 132: Code „readOvData“	230
Listing 133: readUsers Rückgabe	230
Listing 134: Code „readUsers“	231
Listing 135: readAbsences Übergabe	231
Listing 136: readAbsences Rückgabe	231
Listing 137: Code „readAbsences“	233
Listing 138: allocTime Übergabe	234
Listing 139: allocTime Rückgabe	234
Listing 140: Code „allocTime“	235
Listing 141: readAlloc Übergabe	235
Listing 142: readAlloc Rückgabe	235
Listing 143: Code „readAlloc“	236
Listing 144: allocEdit Übergabe	237
Listing 145: allocEdit Rückgabe	237
Listing 146: Code „allocEdit“	237
Listing 147: editStamp Übergabe	238
Listing 148: editStamp Rückgabe	238
Listing 149: Code „editStamp“	239
Listing 150: editUser Übergabe	239
Listing 151: editUser Rückgabe	239
Listing 152: Code „editUser“	240
Listing 153: effTime Übergabe	240
Listing 154: effTime Rückgabe	240
Listing 155: Code „effTime“	242
Listing 156: delEntry Übergabe	242
Listing 157: delEntry Rückgabe	243
Listing 158: Code „delEntry“	243
Listing 159: writeTimestamp Übergabe	243
Listing 160: writeTimestamp Rückgabe	244
Listing 161: Code „writeTimestamp“	245
Listing 162: regUser Übergabe	246
Listing 163: regUser Rückgabe	246
Listing 164: Code „regUser“	247
Listing 165: readBT Rückgabe	247
Listing 166: Code „readBT“	247
Listing 167: readGF Rückgabe	248
Listing 168: Code „readGF“	248