KERNASPEKTE EFFIZIENTER CROSS PLATFORM MOBILE APP ENTWICKLUNG - UNTERSUCHT ANHAND EINER KONKRETEN HYBRID-BEISPIELAPPLIKATION

Master-Thesis zur Erlangung des akademischen Grades

Master of Science, MSc

im Universitätslehrgang Web and Mobile Media Design, MSc 2

eingereicht von

Michael Wagner

Department für Weiterbildungsforschung und Bildungstechnologien

an der Fakultät für Bildung, Kunst und Architektur der Donau-Universität Krems

Betreuer: Dr. Klausjürgen Heinrich

Krems im August 2020

EIDESSTATTLICHE ERKLÄRUNG

Ich, Michael Wagner, erkläre hiermit an Eides statt,

1. dass ich meine Master-Thesis selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfen bedient habe,

2. dass ich meine Master-Thesis oder wesentliche Teile daraus bisher weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe,

3. dass ich, falls die Master-Thesis mein Unternehmen oder einen externen Kooperationspartner betrifft, meinen Arbeitgeber über Titel, Form und Inhalt der Master-Thesis unterrichtet und sein Einverständnis eingeholt habe.

ABSTRACT

Die folgende Masterarbeit behandelt das Thema **Mobile App Development**. Sie wird einen groben Überblick schaffen, welche unterschiedlichen Ansätze der App Entwicklung im Jahr 2020 existieren, welche Vor- und Nachteile sich aus den einzelnen Möglichkeiten ergeben und wie die verschiedenen Methodiken und Submethodiken sich voneinander abgrenzen und unterscheiden.

Ein besonderes Augenmerk wird auf die aktuell weit verbreitete Methode des **Cross Platform Mobile App Developments** gelegt. Hier gibt es verschiedene Ansätze und Libraries, von welchen ich exemplarisch zwei weit verbreitete Exemplare vergleiche und beschreibe. Dabei wird das verwendete Programmierparadigma und die zugrundeliegende Programmiersprache behandelt.

Anhand einer einfachen Beispiel-App, einer Online Einkaufsliste namens *„Me Want That"*, wird in jedem der beiden Beispielframeworks eine Implementierung stattfinden, anhand welcher der *„ease of use"* für den Developer dargestellt wird. Es werden Unterschiede und Gemeinsamkeiten bezogen auf die Themen Programmiersprache, Entwicklungswerkzeuge, Paketverwaltung, Command Line Tools, Debugging und Building sichtbar gemacht.

Anhand der Beispiel Applikation wird exemplarisch der Unterschied zwischen den Libraries *Flutter* (als Kandidat für **Native Hybrid Frameworks**) und *Ionic* (als Kandidat für **Web Hybrid Frameworks**) herausgearbeitet.

Es wird für diese „Me Want That" Beispiel-App dieser Arbeit eine **REST Backend Schnittstelle** entworfen und mit dem PHP-Tool *Laravel* exemplarisch implementiert. Dadurch wird ein Licht auf Datenaustausch, Asynchronität, Sicherheit, Abhängigkeiten und Interaktionen mit einer REST API geworfen. Die benötigten Tools werden vorgestellt und beschrieben. Das Backend ist

wesentlicher Teil einer App-Entwicklung, da der größte Teil der Geschäftslogik, wie bei den meisten Applikationen, serverseitig stattfindet. Da diese Masterthese anhand eines exemplarischen Entwicklungsansatzes den gesamten technischen Entstehungsprozess einer App beleuchten möchte, wird das Web Backend beispielhaft implementiert und dem Thema REST API, Backend und Datenpersistierung ein Kapitel gewidmet.

Nicht-Ziel dieser Arbeit sind die, jeweils in sich wieder sehr ausufernden und wichtigen Themen Designprozess (Farben, Formen, Psychologie, Ikonografie, Typografie, Layout, Screendesign), Marketing und Bewerbung, Store Deployment und Verteilung sowie Copyright und Recht zu betrachten und aufzuzeigen.

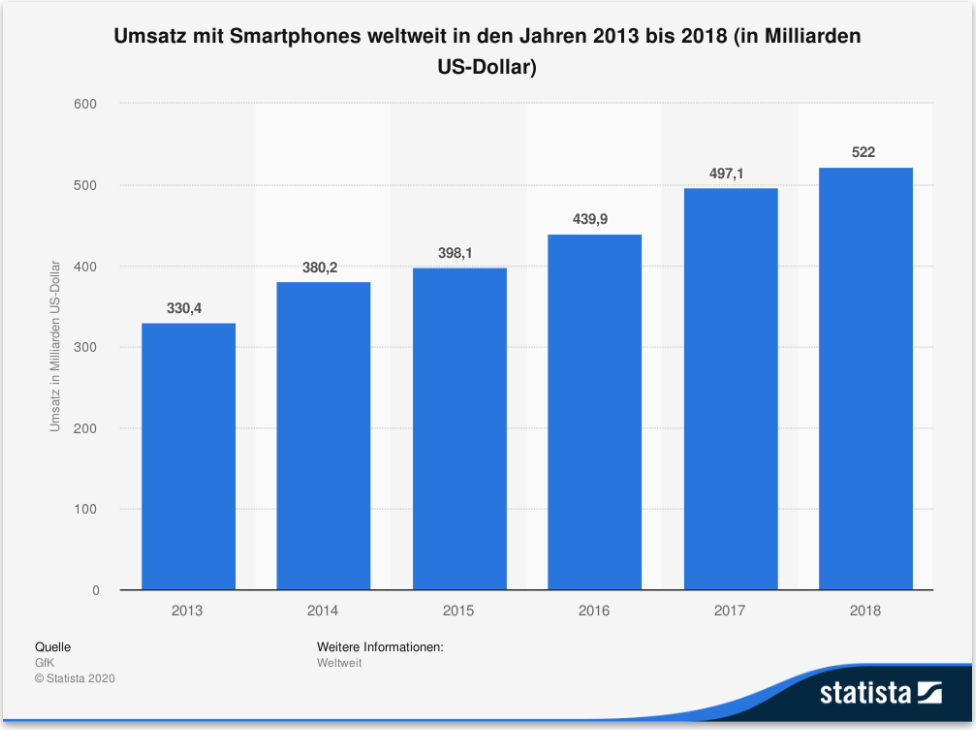
Außerdem sollte erwähnt sein, dass jedes der vorgestellten Frontend Frameworks einen Export der fertigen App sowohl als **Android**, **iOS** oder **HTML5** App zulässt. Die in dieser Arbeit dargestellten Implementierungen beschränken sich aus Gründen der Verfügbarkeit seitens des Autors auf die Android Versionen der jeweiligen Frameworks.

INHALTSVERZEICHNIS

1 Bedeutung von Mobile Applications in der Gesellschaft

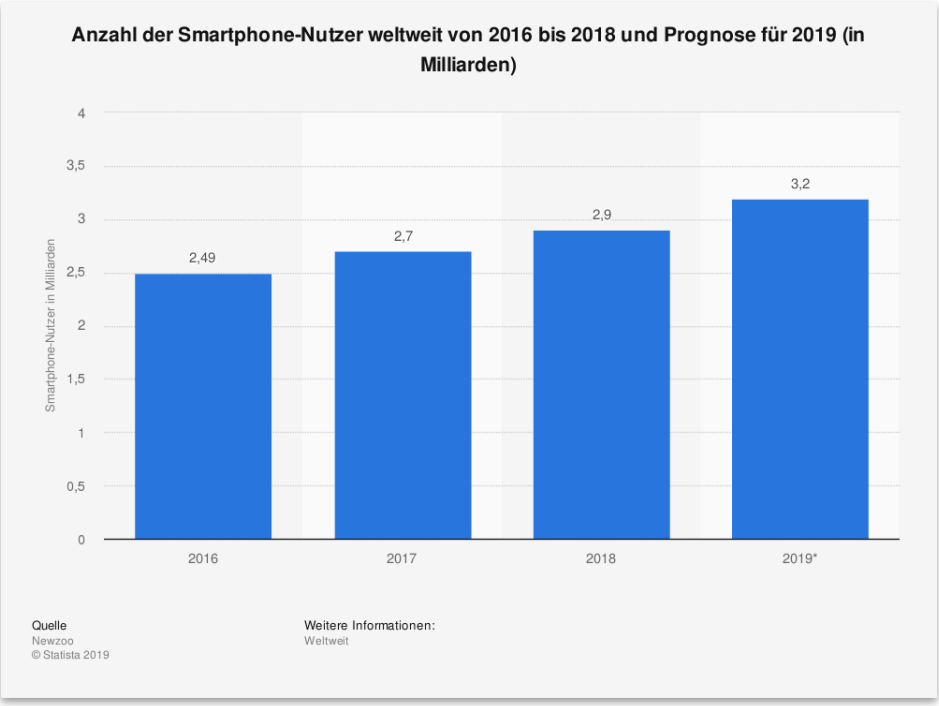
Es ist noch nicht besonders lange her, etwas mehr als eine Dekade vielleicht - da führte eine kleine technische Revolution unsere Gesellschaft auf einen neuen, aufregenden Pfad. Waren wir davor zwar bereits an das Internet gewöhnt, aber im mobilen Alltag trotzdem noch mehrheitlich offline in unserer Lebenszeit unterwegs, ermöglichte uns diese mobile Revolution, also der Besitz eines dieser neuen ***Smartphones*** mit dem richtigen Datenplan dahinter, das Abrufen des Wissens der gesamten Menschheit mit nur wenigen Gesten. Ob im Wald oder in zwanglosen Diskussionen am Abend mit Freunden.

Bei genauerem Hinsehen entpuppt sich diese *„Revolution"* allerdings, wie so oft, als Zusammenlegen von bereits bekannten Technologien kombiniert mit Vereinfachung der Bedienung, einer hohen Verfügbarkeit und Zugänglichkeit in der Gesellschaft. Plötzlich waren wir vernetzt. Plötzlich tauschten immer mehr Leute ihre *„Dumb Phones"* gegen diese neuen, vielversprechenden, sexy und trendy vermarkteten Geräte aus.



Quelle: GfK, Satista Austria 2020

Betrachtet man die weltweite Smartphone-Umsatzstatistik der letzten Jahre, sowie die Anzahl der Nutzer, bemerkt man sofort das Wachstum anhand der steilen Kurve nach oben. Selbst in den Jahren 2016 bis 2018, in denen gefühlt bereits jeder sein eigenes Smartphone besaß, war noch Raum für Wachstum.



Quelle: Newzoo, Satista Austria 2019

Wir schreiben das Jahr 2020. Das Smartphone ist in unserer Gesellschaft etabliert wie nie zuvor. Das liegt größtenteils daran, dass diese "Immer-Online-Recheneinheiten" mittlerweile so leistungsstark sind, dass sie sowohl problemlos aufwändige 3d Spiele darstellen können als sie auch allumfassend die meisten Individuen in unserer Gesellschaft mit Wissen versorgen oder sozial vernetzen.

Ein wichtiger Grund für den Erfolg dieser Devices ist, neben der intuitiven und symbolhaften Bedienung, die einfache, modulare Funktionserweiterung in Form kleiner, nachinstallierbarer Applications, kurz ***Apps***, sowie deren einfache und zentralisierte Verfügbarkeit über die jeweiligen App-Stores. Egal wohin man blickt, welche Problematik sich auftut: Es gibt bestimmt bereits eine App, welche die Problemstellung löst und dem Anwender das Leben erleichtert. Christian

Liebl erinnert sich in seinem Buch „Progressive Web Apps“ wie folgt an den Beginn der App-Erfolgsgeschichte zurück:

> *„There's an app for that“ („Dafür gibt es eine App“) – dieser 2009 in Apple-Werbeanzeigen geschaltete Slogan ging kurzzeitig in den täglichen Sprachgebrauch ein. Die App-Bewegung wurde insgesamt so stark, dass sie sich letztlich sogar auf Desktopbetriebssysteme durchschlug: Im Jahr 2010 veröffentlichte Apple mit dem Mac App Store einen ähnlichen Marktplatz für sein Betriebssystem macOS. Microsoft veröffentlichte im Jahr 2012 Windows 8, das neben der Windows Runtime (WinRT), einer neuen auf Apps zugeschnittenen Laufzeitumgegung, auch den Marktplatz Windows Store (heute Microsoft Store) enthielt.*
>
> *(Liebel, 2018, p. 33)*

Von Apps für Kochrezepte über Musik und Videostreaming bis hin zu Onlineshops, Wissensdatenbanken, Spiele für den Zeitvertreib oder Notenblätter für Alphornbläser. Kein Anwendungsfall scheint zu spezifisch, kein Problem unlösbar. Die meisten Nischen bieten immer noch genug Anwender, um wirtschaftlich erfolgreich sein zu können.

Die Alleskönner haben sich mit ihrer Rechenleistung, welche vor wenigen Jahren noch ausschließlich großen Serverlandschaften vorbehalten war und jetzt in jede Hosentasche passt, ihren Platz in unserer Gesellschaft verdient wie wenige andere Geräte in der Vergangenheit der Menschheitsgeschichte.

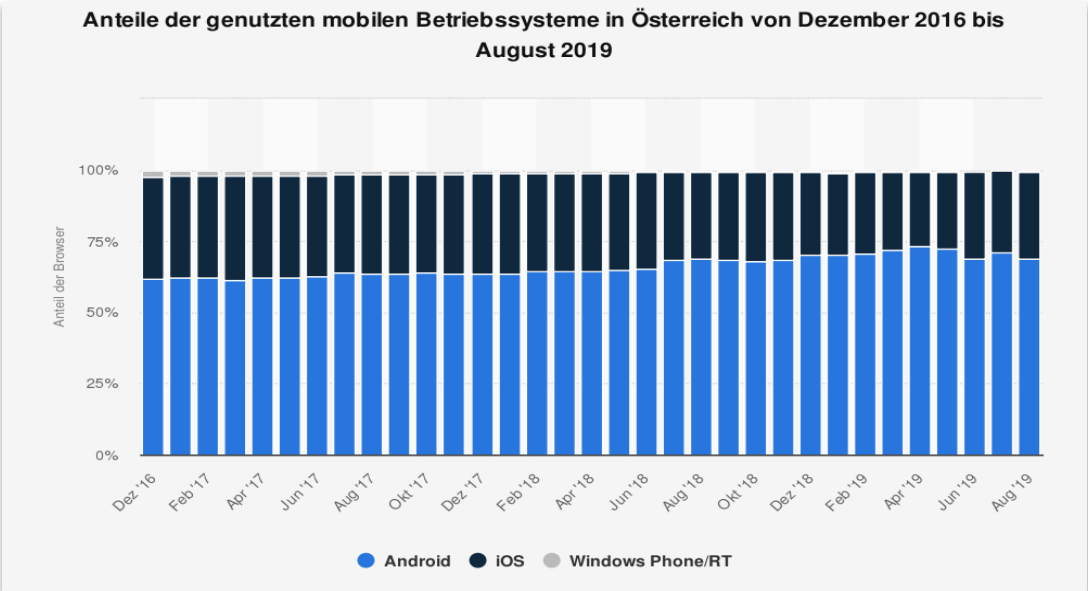
Nach wie vor intuitiv mit einfachen Gesten und Berührungen zu bedienen, aber mittlerweile zum größten Teil wasserfest, bruchsicher und über zahlreiche Schnittstellen wie Bluetooth, NFC, Infrarot, 5g oder WiFi mit dem Umfeld vernetzt, sind sie aus unserem modernen Leben für Jung und Alt kaum mehr wegzudenken. Losgetreten hat diesen Trend, mobile Telefone „smart“ zu machen das *iPhone* im Jahr 2007.

> *Mit Einführung des iPhones 2007 war schnell klar, dass dieses Gerät den Markt mobiler Endgeräte revolutionieren würde. Und tatsächlich*

versetzte das iPhone den Handymarkt in helle Aufregung. Quasi über Nacht wurde Apple mit dem ersten Smartphone, das sich vollständig über einen großen Touchscreen per Fingergesten bedienen ließ, zum weltweiten Trendsetter. Am ersten Verkaufstag wechselten bereits 270.000 Geräte den Besitzer, und das, obwohl der Einführungspreis des Geräts bei 599 US$ lag.

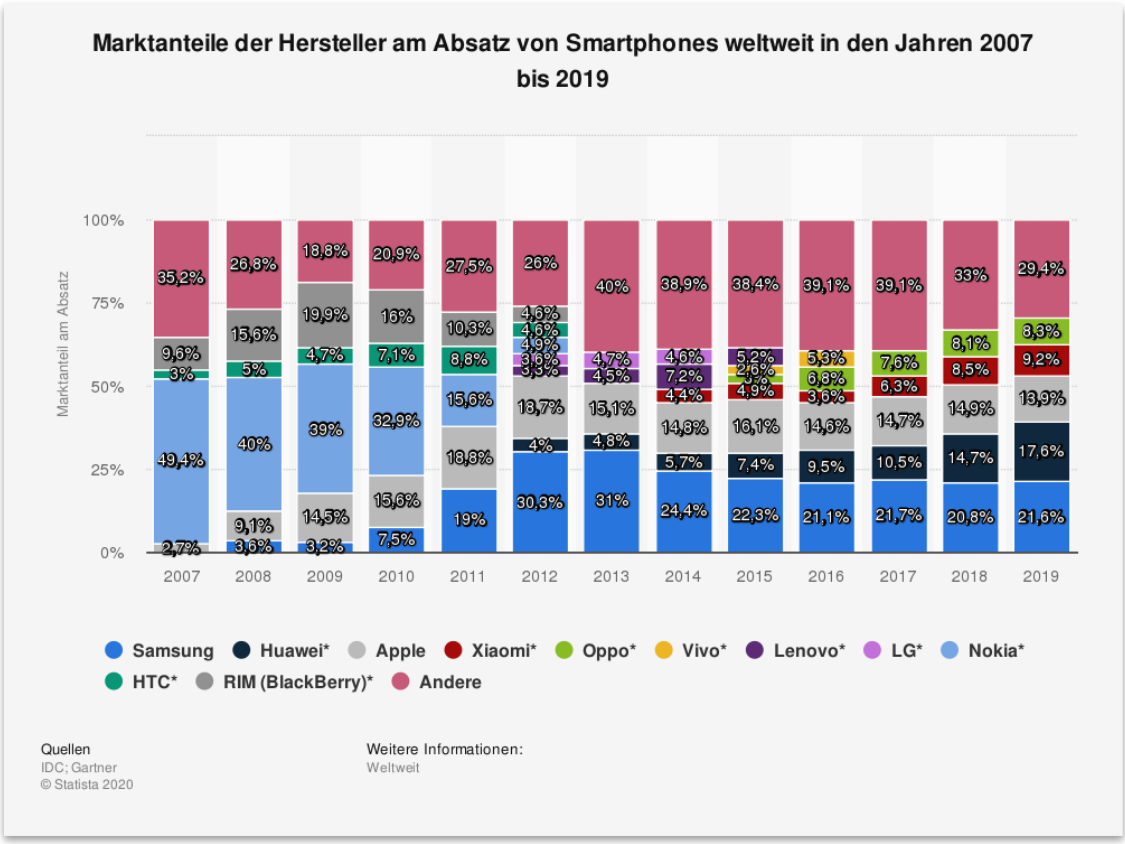
(Semler and Tschierschke, 2019, p. 21)

Aber auch die Konkurrenz, namentlich Google mit seinem Betriebssystem *„Android"*, schlief nicht und konnte alsbald zum großen Apfel aufschließen. Die Geschichte rund um iPhone und Android Devices, deren Wechselwirkung, Konkurrenzsituation, Anhängerschaft (welche durch den Lock-In Effekt ihr eigenes System fast als Religion versteht und das jeweils andere dämonisiert) sowie die Versionsgeschichten mit ihren vielen voneinander kopierten Features würden bei genauerer Betrachtung viele Seiten füllen und eine eigenständige (meiner Meinung nach interessante) Arbeit abgeben. An dieser Stelle sei allerdings nur erwähnt, dass das Verhältnis dieser beiden Firmen kompliziert ist und jede den Communities gewisse Vor- und Nachteile bietet. Sieht man sich den Marktanteil der Betriebssysteme für Smartphones in Österreich sowie global von 2016 bis 2019 an, bemerkt man eine starke Dominanz von Android.



Quelle: Österreichische Webanalyse, Satista Austria 2020

Der Grund dafür ist, dass Android im Gegensatz zu iOS eine freie Software darstellt und somit von diversen Firmen wie Samsung, HTC, Huawei oder Motorola verwendet wird. Dies steht gegenüber einer einzigen Firma welche iOS einsetzt, oder vielmehr einsetzen darf. Namentlich *Apple*.



Quelle: IDC; Gartner, Satista Austria 2020

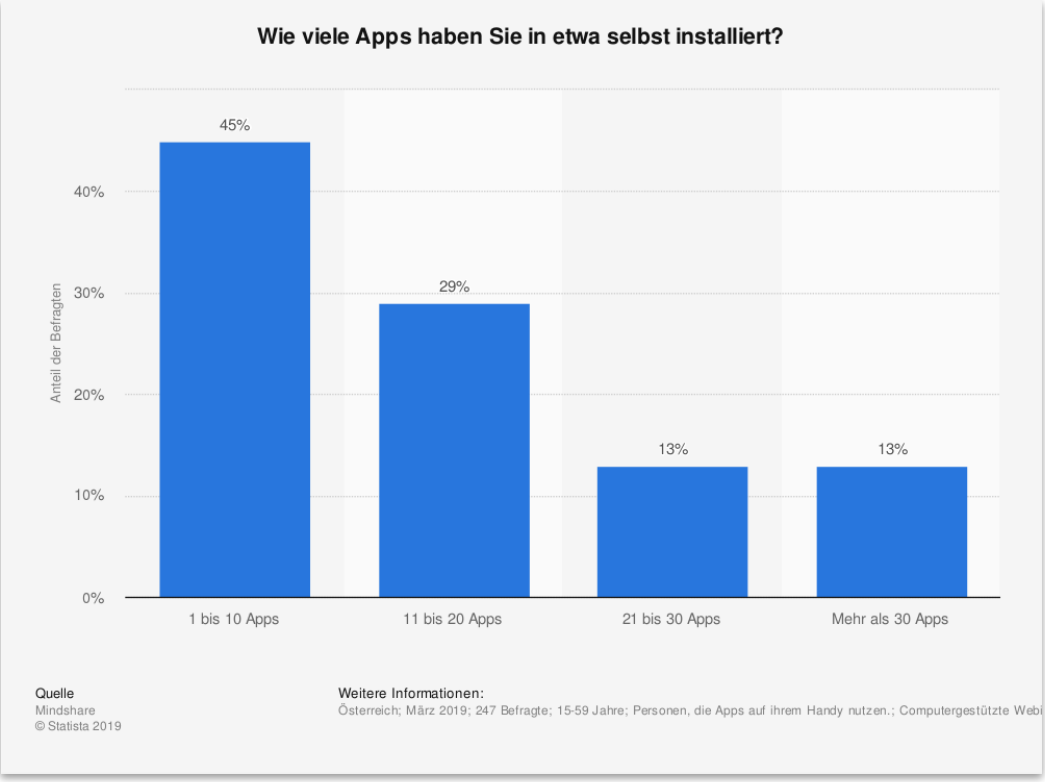
Naturgemäß eröffnet die Erfolgsgeschichte dieser neuen Devices viele Geschäftsfelder für App-Entwickler und -Designer, waren ja die App-Stores auf den verschiedenen Devices offen für Drittentwickler.

Vom Handel mit den entstehenden Daten über werbefinanzierte Apps und Inhalte bis zu teuren Abonnement Modellen für das Beziehen von Premiuminhalten sind dem unternehmerisch denkenden Zeitgenossen nur wenige Grenzen gesetzt. Nicht zu vergessen der *"normale"* Verkauf über den jeweiligen App-Store sowie App-Auftragsarbeiten für die zahlreichen Klein- und

Mittelunternehmer, welche naturgemäß auch gerne in der digitalen Welt der App-Stores vertreten und auffindbar sein wollen.

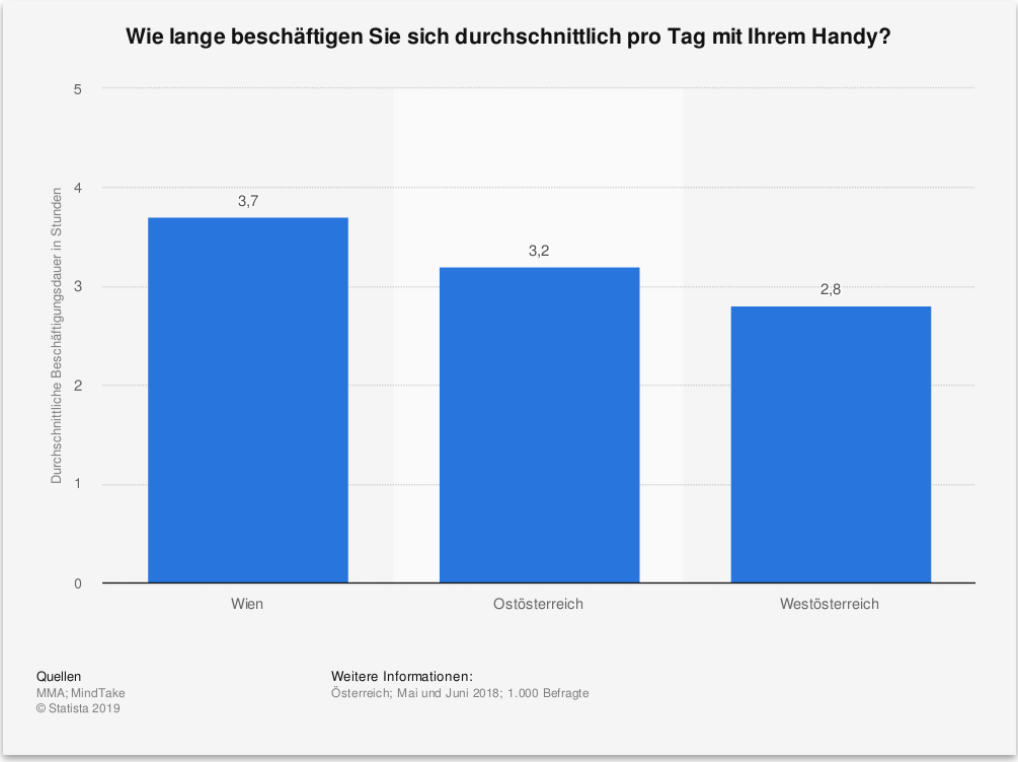
> *Menschen benutzen täglich Werkzeuge, um Probleme zu lösen, von einem Hammer bis hin zu Robotern sind diese Werkzeuge sehr vielfältig. Ein Smartphone, Tablet oder eine Smartwatch sind im Grunde nichts anderes als ein Werkzeug oder - besser gesagt - ein Allzwecktaschenmesser. Jedoch mit dem Unterschied, dass wir selbst bestimmen, mit welchen Werkzeugen wir unser Taschenmesser bestücken.*
>
> *(Semler and Tschierschke, 2019, p. 29)*

Im Jahr 2019 wurden 247 Smartphone User in Österreich gefragt wie viele Apps sie auf dem Smartphone installiert haben:



Quelle: Mindshare, Satista Austria 2019

Auch die durchschnittliche Zeit, die die User täglich mit dem Smartphone verbringen, war im Jahr 2018 in Österreich rund 3h/Tag.



Quelle: MMA; MindTake, Satista Austria 2019

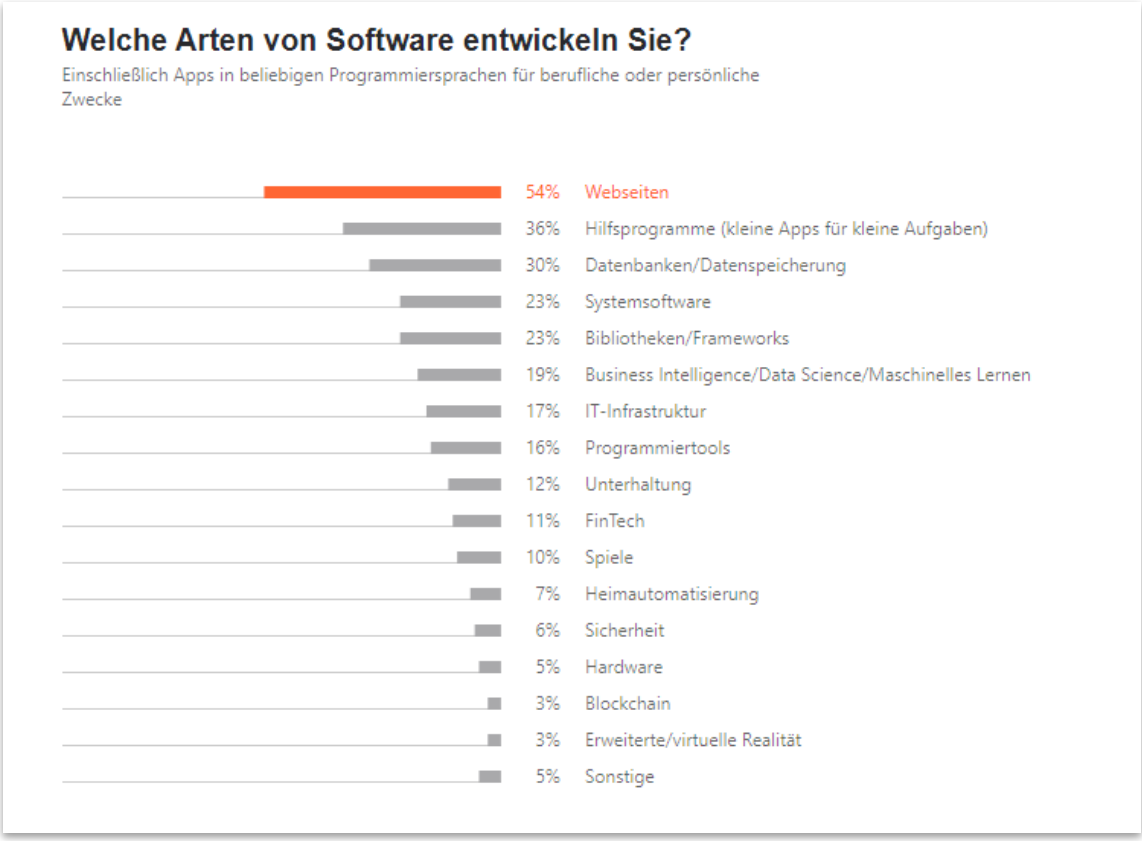
Allerdings sind die User auch eine gewisse Güte gewohnt und verwenden nicht alle Applikationen, die ihnen im Store vorgesetzt werden. Falls eine App nicht die hohen Qualitätsansprüche der User erfüllt, wird sie mit einem schlechten Review bestraft und nicht weiterverwendet oder deinstalliert. Semler und Tschierke teilen in Ihrem Buch „App-Design" folgende Statistik mit ihren Lesern:

> *Rund 30% der neu installierten Applikationen werden nur einmal genutzt, und weniger als 5% sind rund einen Monat nach der Installation noch in Gebrauch*
> *(*https://pharmaphorum.com/articles/are-apps-medical-devices-part-three-how-do-you-create-an-app-which-also-creates-downloads-and-engagement/*)*
>
> *(Semler and Tschierschke, 2019, p. 34)*

Angesichts dieser Überlegungen verwundert es nicht besonders, dass unzählige Apps jeden Tag in den Stores veröffentlicht werden. Für die meisten Anwendungsfälle gibt es nicht nur eine, sondern eine Vielzahl an Apps zum

Download. Die Nachfrage und das Angebot scheinen unbegrenzt und der Konkurrenzdruck ist relativ hoch. Es haben sich verschiedene technische Wege herauskristallisiert, eine App zu entwickeln, jede mit Vor- und Nachteilen. In den folgenden Kapiteln werde ich einen Überblick schaffen, welche Hauptmöglichkeiten zum App Development im Jahr 2020 existieren und einen Weg (Cross Platform Development) mit seinen Vorteilen besonders hervorheben.

Sieht man sich die JetBrains Developer Survey Ergebnisse des Jahres 2020 an, stellt man fest, dass bereits 36% (Platz 2 nach Webseiten) der befragten Entwickler aus allen Bereichen hauptsächlich an „Apps“ arbeiten. Spezialformen wie Spiele, und Unterhaltung sind da noch nicht mit eingerechnet.



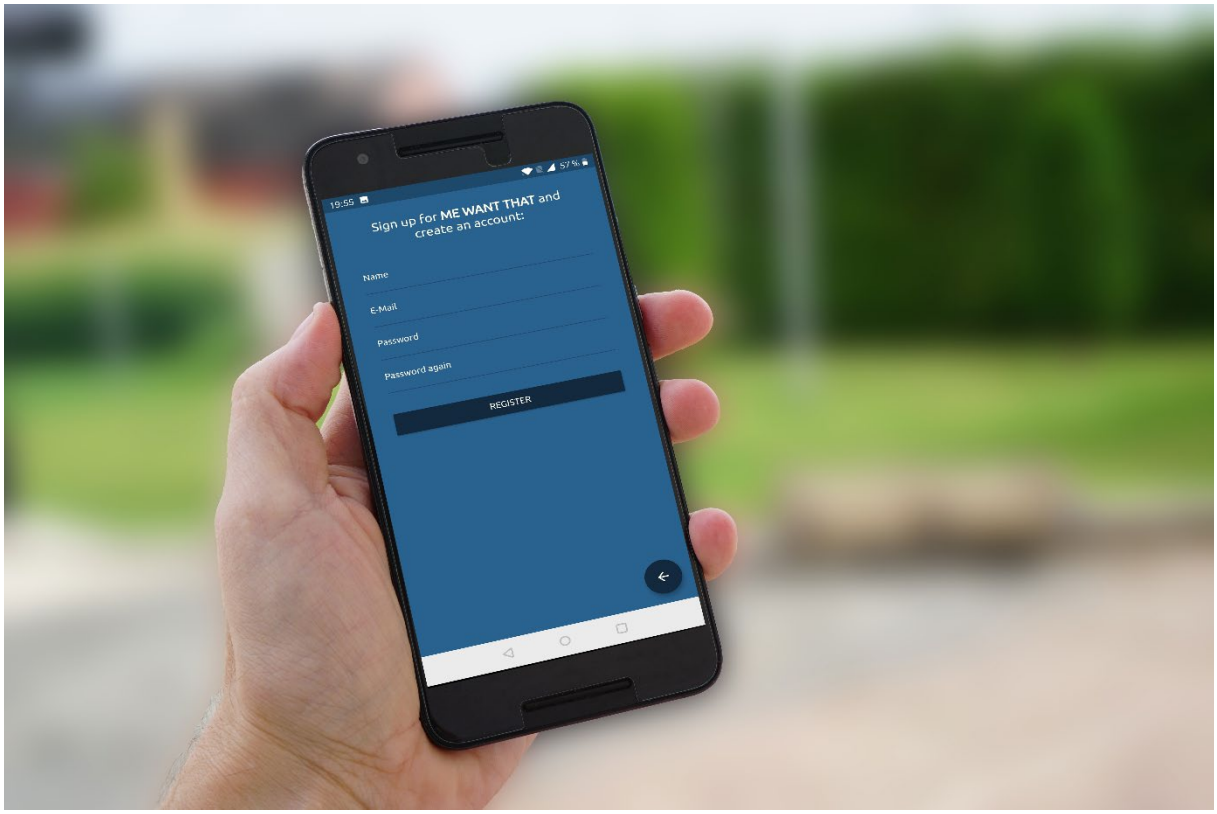
Quelle: JetBrains Developer Survey 2020

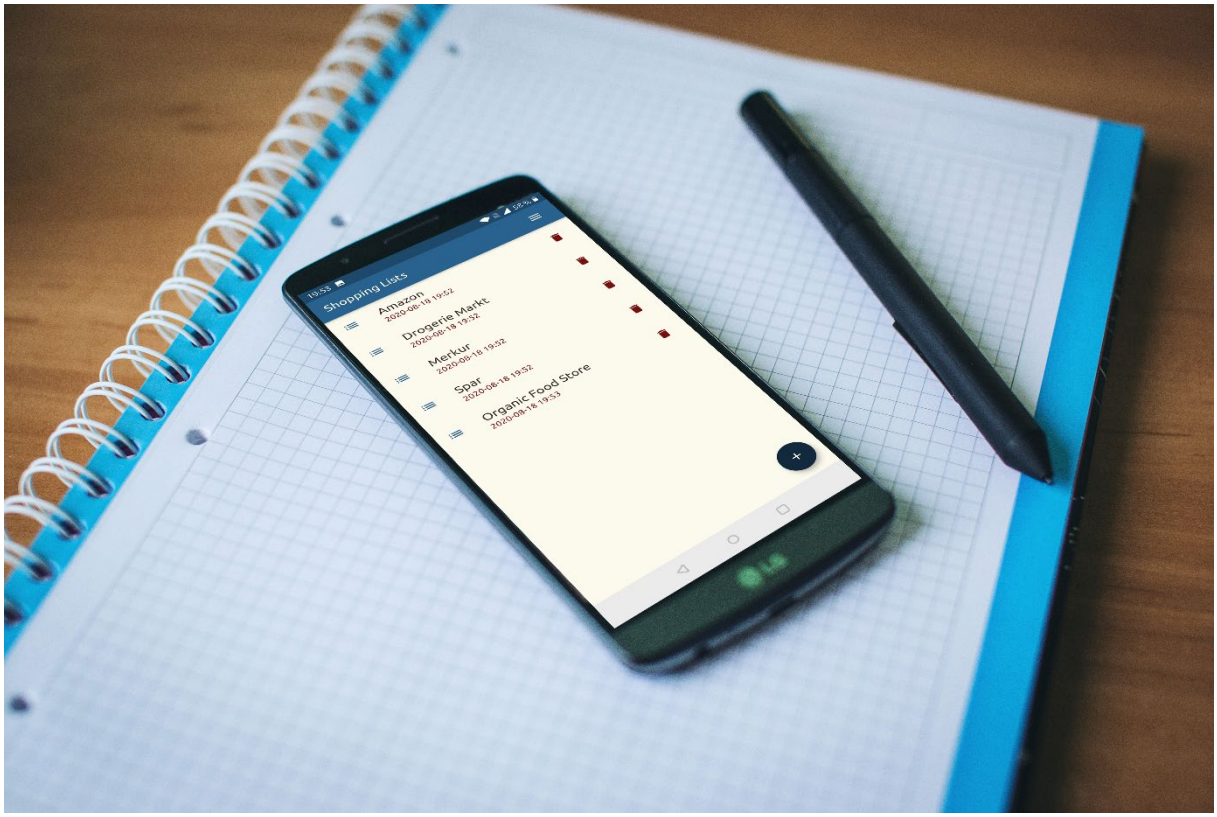
2 Forschungsfrage

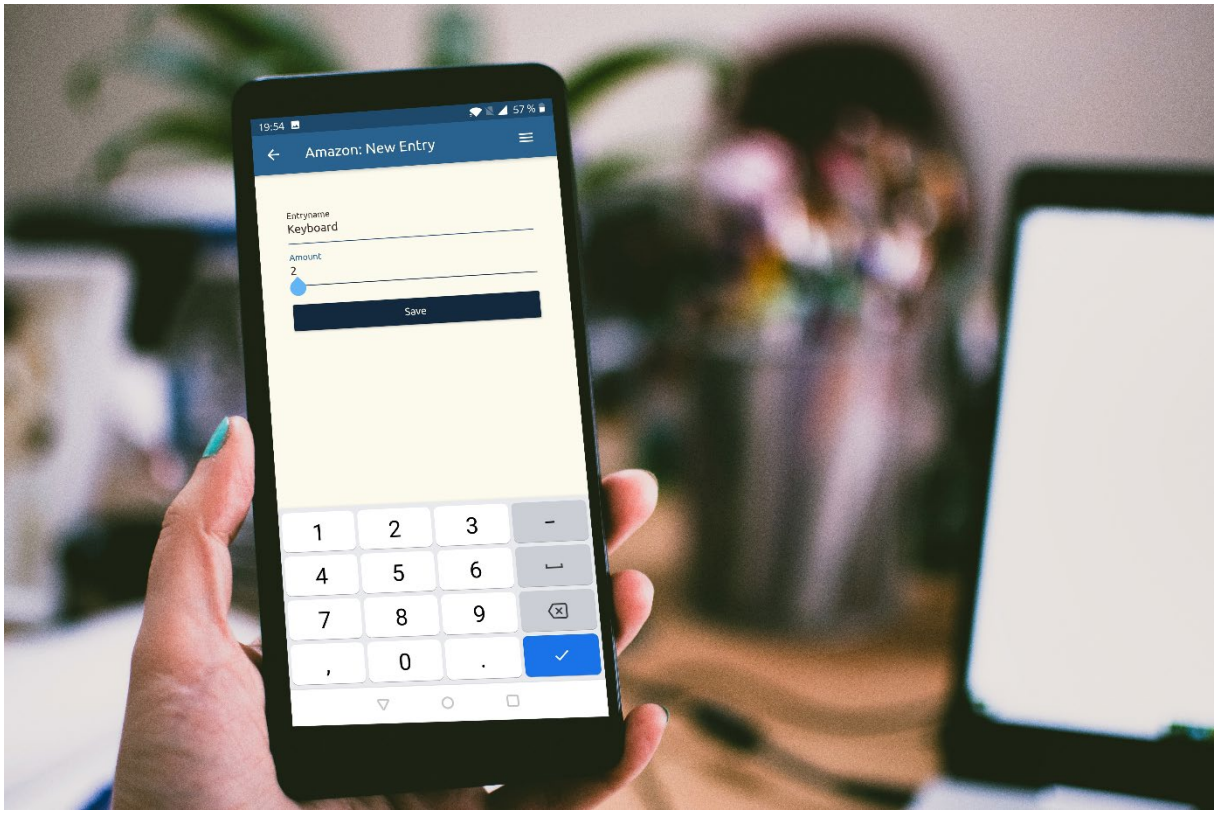
Meine Forschungsfrage ist also: Welche Fertigkeiten im App Back- und Frontend Development sind im Jahr 2020 von Relevanz, und **welche Hauptmöglichkeiten des Mobile App Developments gibt es aktuell** und kennzeichnen besonders den spezifischen Bereich des Cross Platform Developments.

Anhand der Beispielapplikation *„Shopping List"* will ich eine *Full-Stack Entwicklung* beschreiben und diese Hauptmöglichkeiten im Blick auf **Geschwindigkeit der Ausführung**, **Startzeit**, **User Interface Komponenten**, **Komplexität der Anwendung**, **Größe der Builds**, **Entwicklungsprozess**, **Debugging** und **Dokumentation** mithilfe der beiden Frameworks *„Flutter"* und *„Ionic"* untersuchen und die diesbezüglichen Ergebnisse am Ende formulieren.

3 Vorstellung der Beispiel-App „Me Want That“




Shopping Lists
Amazon
Drogerie Markt
Merkur
Spar
Organic Food Store


19:54
Amazon: New Entry
Entryname
Keyboard
Amount
2
Save

3.1 Beschreibung der App

Für eine beispielhafte Implementierung in den Cross Plattform Frameworks **Flutter** und **Ionic** werden wir eine einfache Online Einkaufslisten App mit dem Namen *„Me Want That“* erstellen. Diese App bietet zu Gunsten von kurzem, verständlichem Sourcecode nur sehr wenige Kernfeatures an und stellt eine userbasierte Sammlung an Einkaufszetteln oder Einkaufslisten dar.

Das App Frontend wird aus 6 verschiedenen Screens bestehen:

- *Login*
- *Registrierung*
- *Auflistung der Einkaufslisten*
- *Anlegen einer neuen Einkaufsliste*
- *Auflistung der Posten einer spezifischen Einkaufsliste*
- *Anlegen eines neuen Postens auf einer Einkaufsliste*

Für jeden dieser Screens wird zuerst ein grobes Wireframe ausgearbeitet und danach ein einfaches Screendesign erstellt.

Des Weiteren wird für die **CRUD** (**c**reate, **r**ead, **u**pdate, **d**elete) Operationen der App eine *REST API Ressource* online angelegt (siehe nächstes Kapitel) und im Frontend je eine asynchrone REST-API Library implementiert, welche die Operationen an die Serverstelle kommunizieren und das Userinterface über das Ergebnis der Operation informieren.

3.2 App Design

Würden wir hier nicht exemplarisch, sondern konkret eine neue App entwerfen und designen, würden uns zu diesem Zeitpunkt außerdem weitere wichtige Schritte im App Design begegnen:

- Wir würden uns intensive Gedanken zu unserer **Zielgruppe** machen
- Wir würden alle möglichen Szenarien als **User Stories** dokumentieren
- Wir würden Personas anlegen und die User Stories mit diesen **Personas** durchspielen
- Wir würden ein passendes **Farbthema**, geeignete **Typografie**, **Bildsprache**, **Ikonografie** und **Layout** überlegen
- Iteratives, zyklisches Ausarbeiten von **Wireframes** der Komponenten und Screens mit Auftraggeber und Zielgruppe
- Erstellen von **Screendesigns**, inklusive **AB-Tests**
- **Wahl der Technologie** anhand von Erwartung gegenüber Endprodukt in Bezug auf Umsetzbarkeit, Zielplattformen, Ansprüche an Performance, …
- **Implementierung** und **Testing**

Jeder dieser Punkte ist relevant und muss für eine erfolgreiche Umsetzung berücksichtigt werden. Für unsere exemplarische Implementierung geht es uns aber nicht um App Design und Marketing, sondern um ein Sichtbarmachen der technischen Erfordernisse und Unterschiede in den einzelnen Implementierungsvarianten, weswegen wir die folgenden Seiten kürzer halten und nicht zu tief ins Detail gehen werden.

3.3 Wireframes

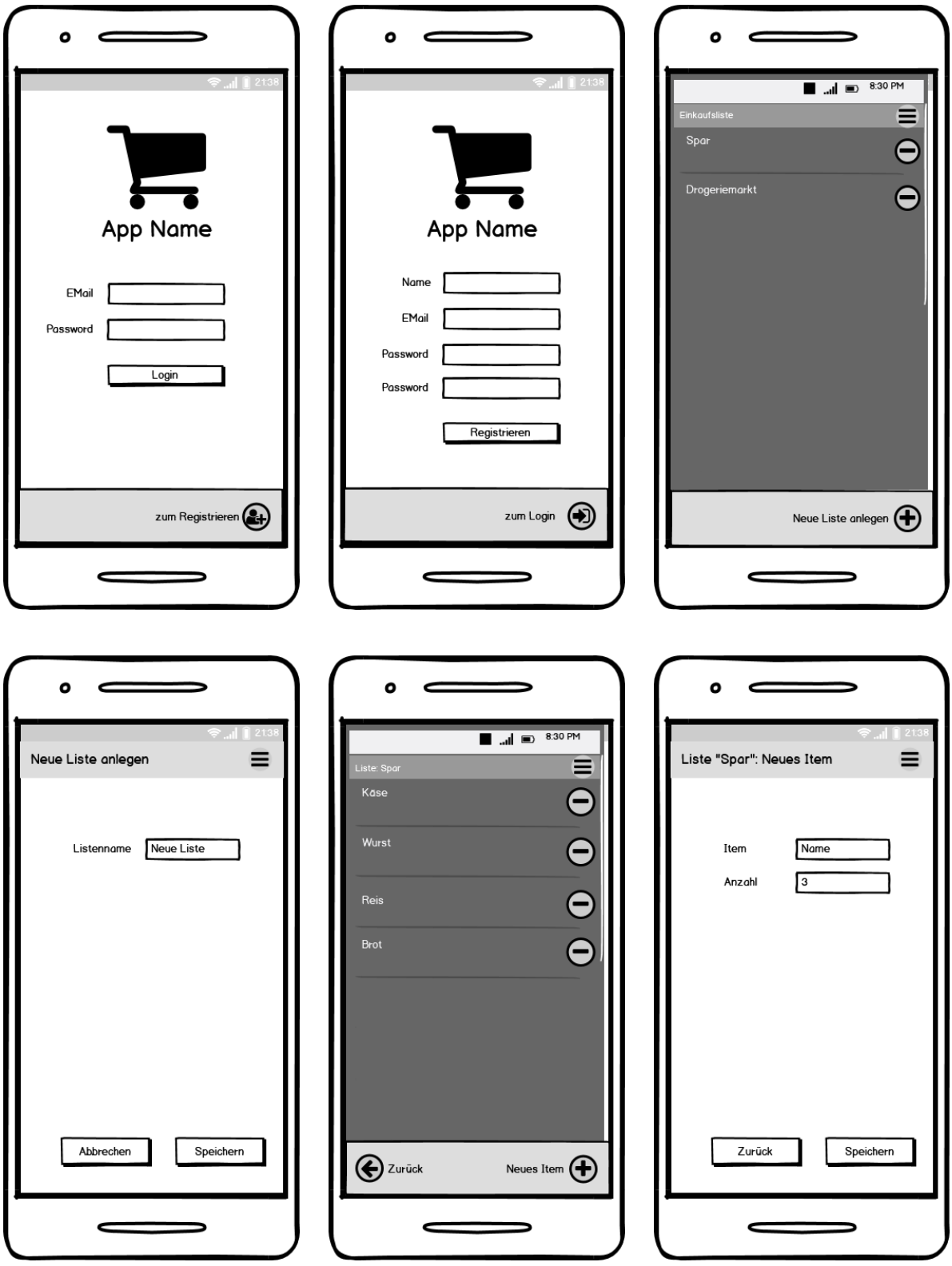
Wireframing, ein wichtiger Schritt im Development Prozess der gerne übergangen wird. Ohne diese konzeptionellen Überlegungen läuft man schnell Gefahr, dass in der Implementierung konzeptionelle Fehler begangen werden. Diese Erfahrung kann ein laufendes Projekt aus der Bahn werfen oder bestehende Deadlines nach hinten verschieben.

> *Ein Wireframe ist ein sehr früher konzeptioneller Entwurf der Website. Dabei geht es vor allem um die Anordnung und Positionierung der oben aufgeführten Elemente und noch nicht um die konkrete visuelle Gestaltung und die Funktionalität. Wireframes werden häufig schon in der Konzeptionsphase eingesetzt.*
>
> *(Hahn, 2017, p. 303)*

Wireframes sind bewusst frei von Farben und schönem Design gehalten, da es in diesem Schritt primär um die Ausrichtung, Anordnung und das Layouting der Screens geht.

> *Da grafische Elemente wie Farben, Formen und Bilder oder Typographie hier noch keinerlei Rolle spielen, liegt der Fokus auf den Inhalten und deren Anordnung.*
>
> *(Hahn, 2017, p. 303)*

Hier sehen wir die Wireframes unserer vier Screens, erstellt mit dem Wireframing Tool ***„Balsamiq Mockups“***.

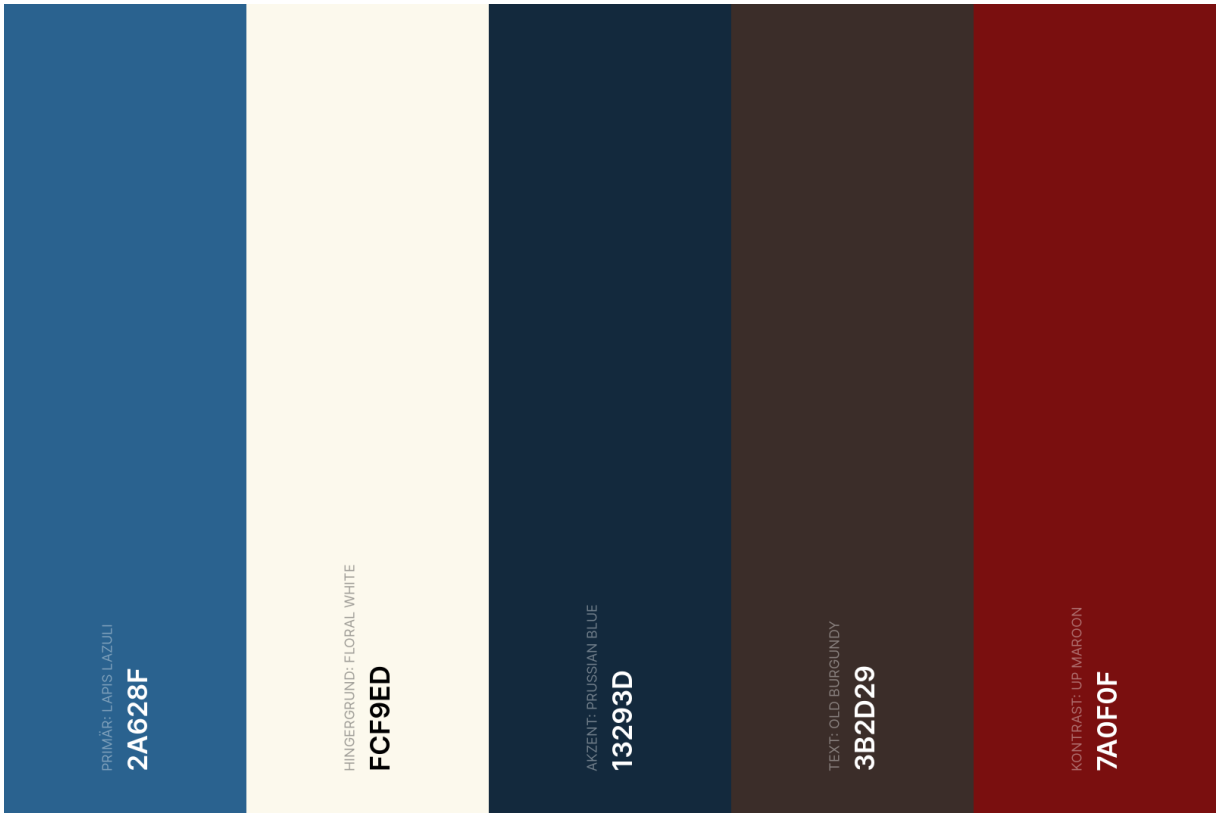


Ist man mit dem Ergebnis seiner Wireframes zufrieden, kann man in die nächsten Phasen übergehen: *Farben*, *Logo*, *Typographie* und *Screendesigns*.

3.4 Farbthema / Farbpalette

Bei der **Farbauswahl** und dem **Farbthema** waren mir zwei dunkle Blautöne wichtig, wobei die dunklere als Akzentfarbe (*Prussian Blue*) und die hellere als Primärfarbe (*Lapis Lazuli*) zu verstehen ist. Das Thema wird als Hintergrund ein sehr helles Gelb mit dem Namen *„Floral White"* erhalten. Texte, welche auf der Hintergrundfarbe geschrieben sind, erhalten die Textfarbe Dunkelbraun (*Old Burgundy*), welche ausgezeichnet zu der spärlich für zB Icons eingesetzten Kontrastfarben dunkelrot (*UP Maroon*) passt.

Effektiv sieht die Palette wie folgt aus:



Für die Auswahl der Farben hat sich der gute Online Farb-Generator http://coolers.co bewährt, welcher auch für die Feinauswahl de Farben in dieser Farbpalette hilfreich war.

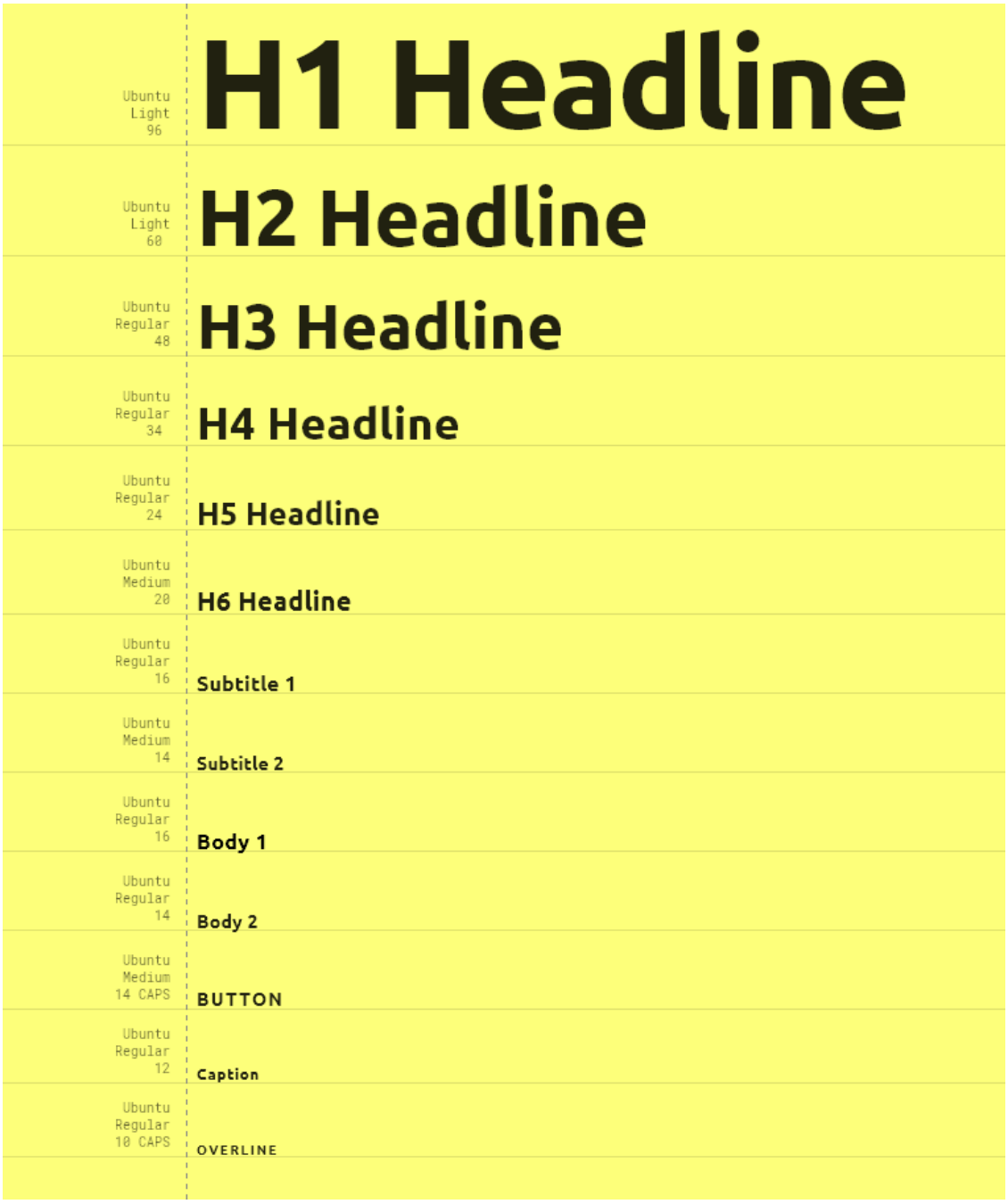
3.5 Logo

Die Gestaltung des Logos übernahm im Groben, nach eingeben der mir wichtigen Parameter (genaue Farbcodes, kombiniertes Symbol- und Schriftlogo und Stil) der Online Logo Maker der Firma Wix: https://de.wix.com/logo/erstellen

Im Folgenden werden das App-Logo, der Android Splashscreen und die Ionic- und Flutter Icons abgebildet:

3.6 Typographie und Ikonografie

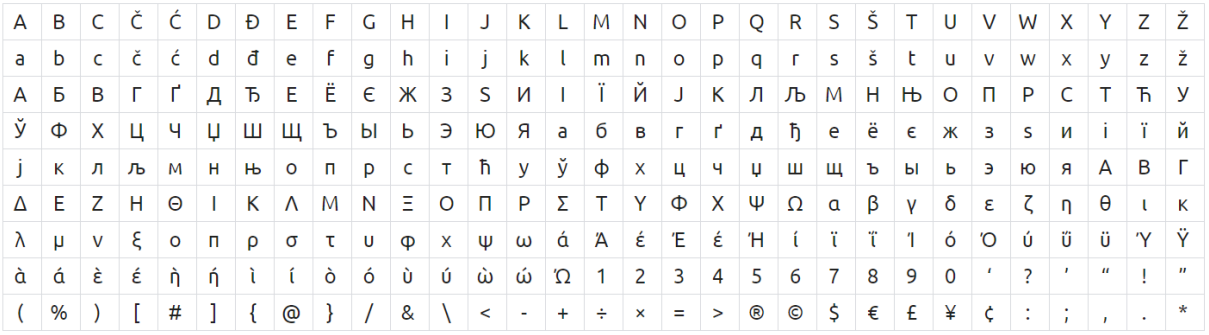
Das Schriftbild wurde in den Größen, Verhältnissen und Letter Spacings aus der Google Material Design Beschreibung übernommen.



Auch die in der App verwendeten Icons sind dem unter freier Lizenz befindlichen Google Material Design entlehnt. Die verwendeten Google-Schriftarten Ubuntu (Sans Serif) und Caveat (Handwriting) stellen sich wie folgt dar:

*3.6.1 Hauptschrift: **Ubuntu***

Glyphen:



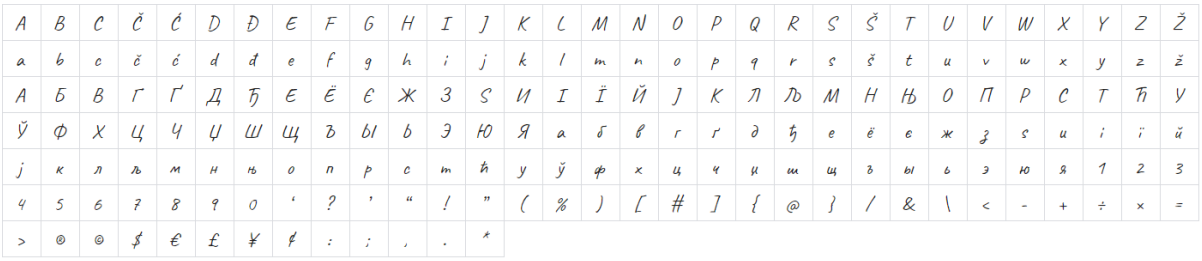
Schriftbild:

Regular: Almost before we knew it, we had left the ground.

Bold: **Almost before we knew it, we had left the ground.**

*3.6.2 Akzentschrift: **Caveat***

Glyphen:



Schriftbild:

Regular: Almost before we knew it, we had left the ground.

Bold: **Almost before we knew it, we had left the ground.**

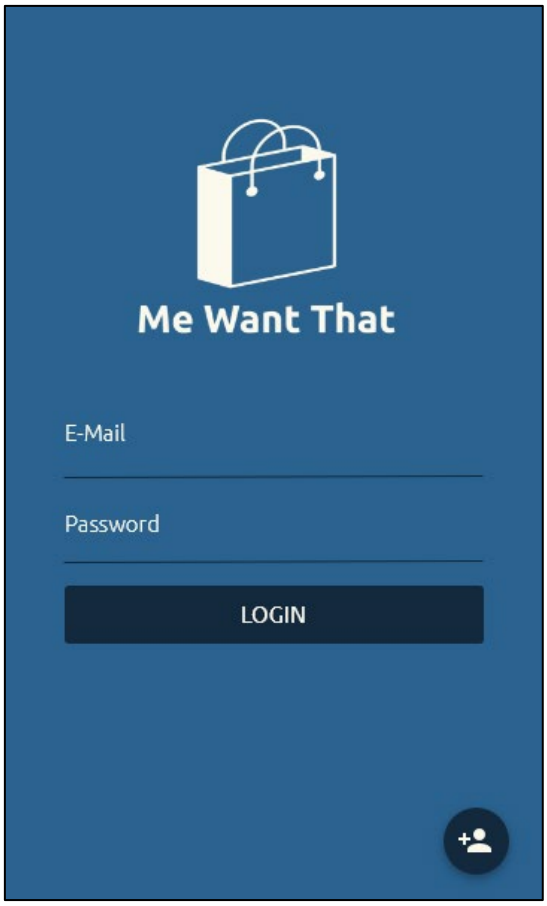
3.7 Screendesign

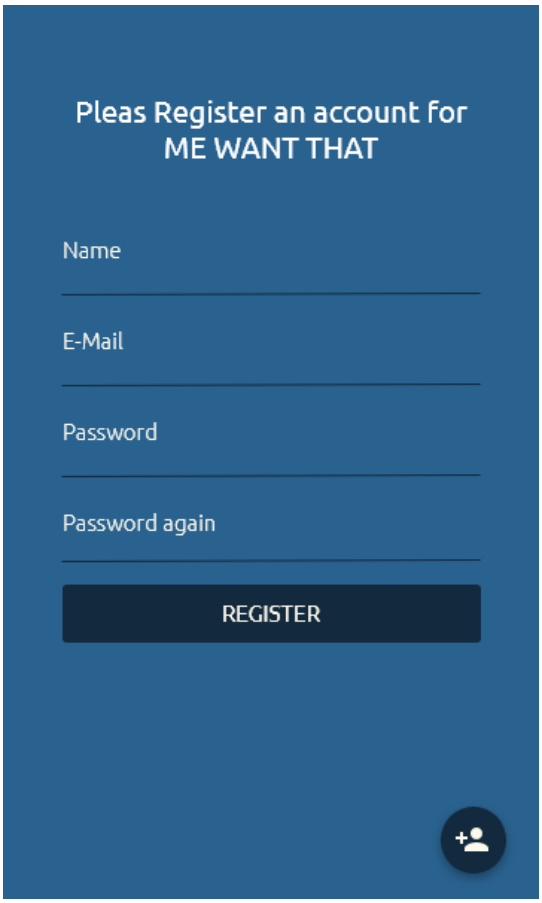
> *Screendesign ist mehr als reine Dekoration oder das Bestreben, eine Website optisch ansprechend zu machen. Ein gutes Screendesign sorgt entscheidend mit dafür, dass eine Website ihre Ziele erreicht.*
>
> *(Hahn, 2017, p. 259)*

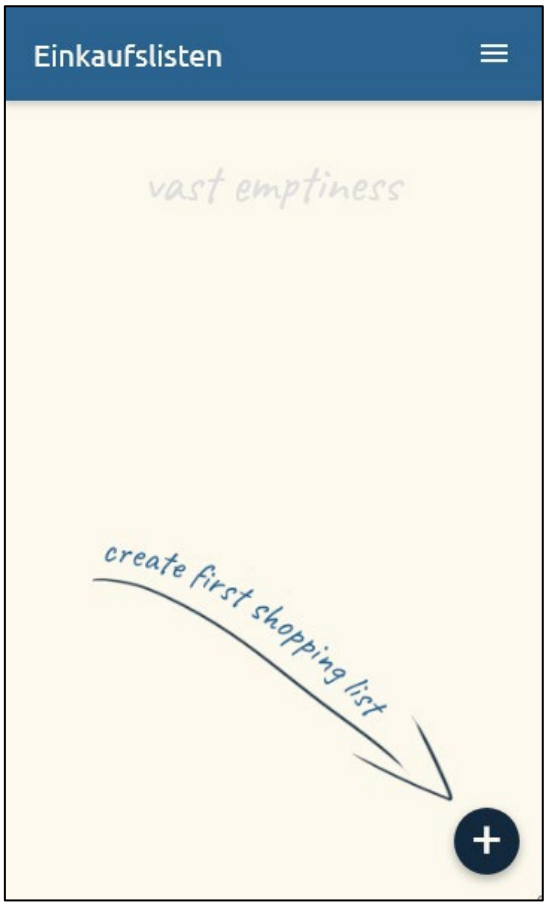
Dem Schritt des Screendesigns kommt in der Entwicklung einer App eine besondere Rolle zu. Hier werden zum ersten Mal die Überlegungen aus den vorhergehenden Schritten miteinander kombiniert. Farbpsychologie, Typografie und Ikonografie greifen in die fertig gelayouteten Wireframes und ergeben ein stimmiges Gesamtbild, an welchem sich die Implementierung orientieren wird.

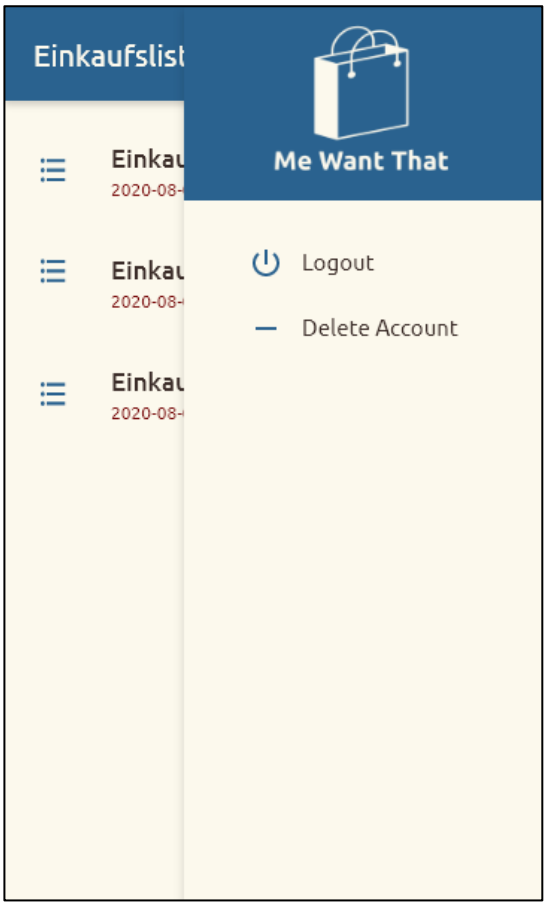
> *Design ist mehr, als alles ein „bisschen hübsch" zu machen. Design bedeutet, Informationen zu strukturieren und zu gestalten. Design heißt, den Anwender durch die Seite zu führen. Design gibt Orientierung und macht Bedeutungen sichtbar. Wenn das nicht gelingt, handelt es sich eben doch nur um Dekoration.*
>
> *(Hahn, 2017, p. 28)*

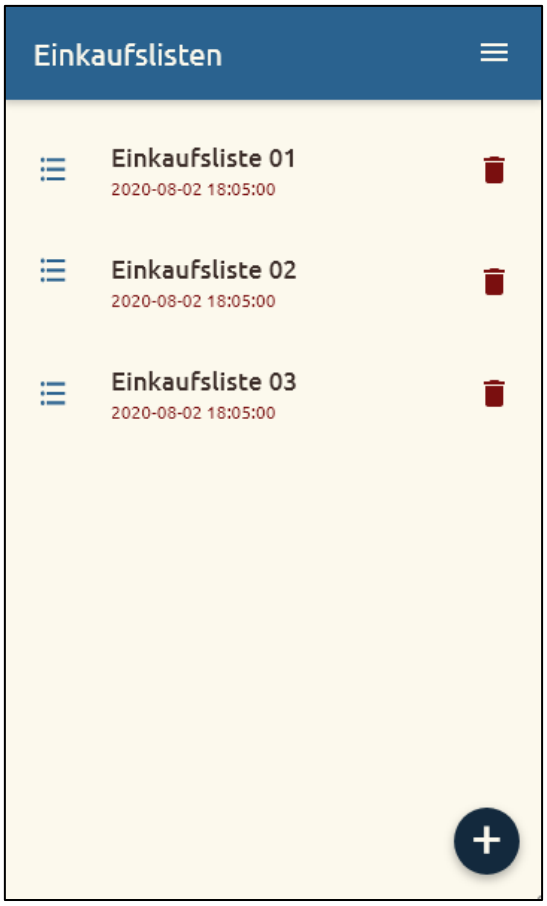
Im Folgenden sehen wir die fertigen Screendesigns der vier Screens unserer Beispiel Applikation „Shopping List". Das Layouting, die Anordnung und das Navigationskonzept aus dem vorherigen Schritt des Wireframings wurde eingehalten und optisch mit dem Material Design von Google in eine ansprechende Form gebracht. Verwendet wurde dafür die vektorbasierte User Interface Grafiksoftware ***„Adobe XD"***.

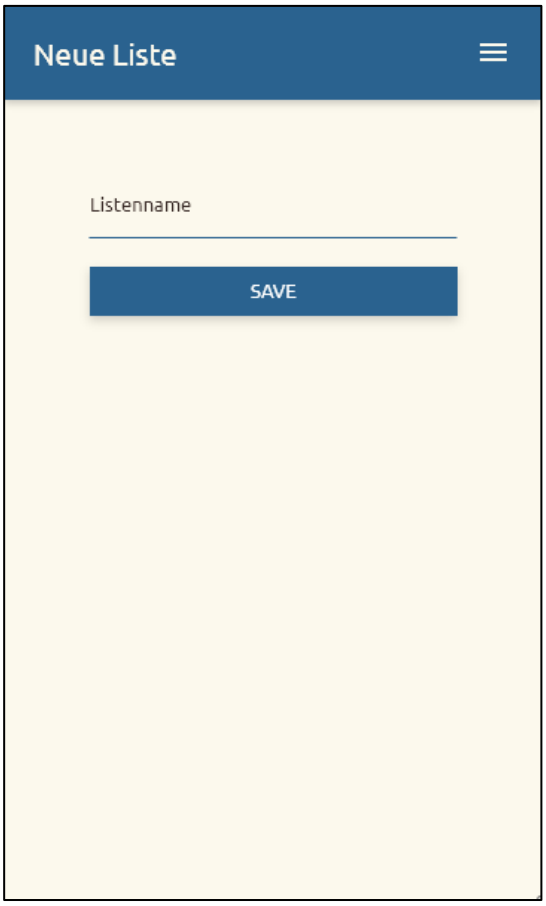

Me Want That
E-Mail
Password
LOGIN

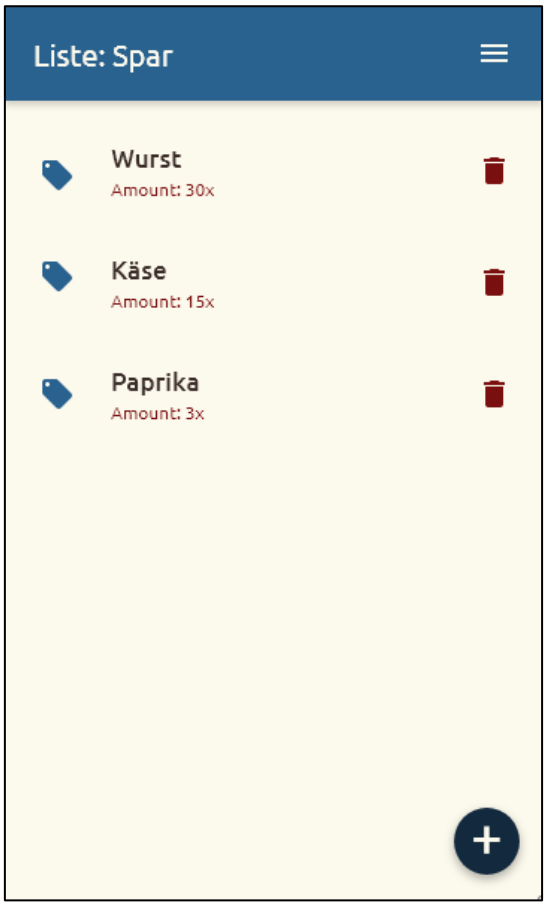

Pleas Register an account for
ME WANT THAT
Name
E-Mail
Password
Password again
REGISTER

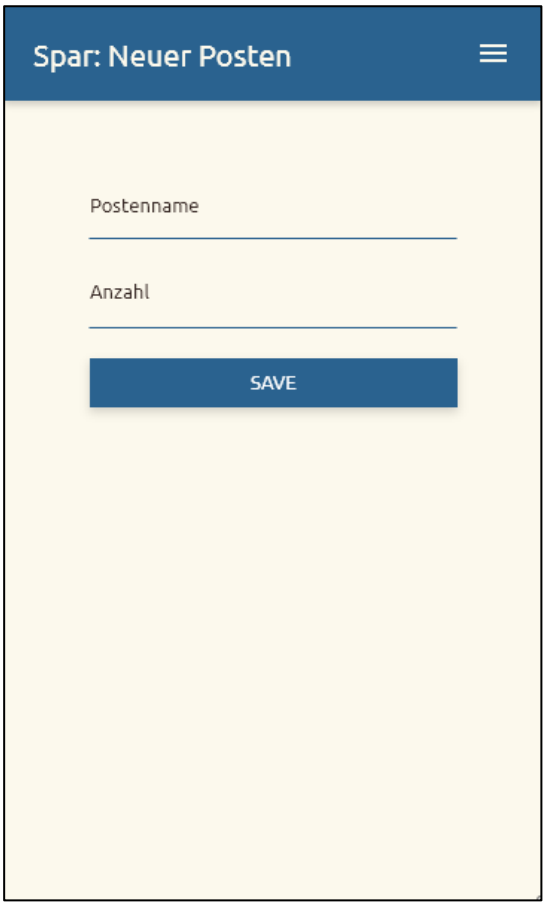

Einkaufslisten
vast emptiness
create first shopping list


Einkaufslist
Einkau
2020-08-
Einkau
2020-08-
Einkau
2020-08-
Me Want That
Logout
Delete Account


Einkaufslisten
Einkaufsliste 01
2020-08-02 18:05:00
Einkaufsliste 02
2020-08-02 18:05:00
Einkaufsliste 03
2020-08-02 18:05:00


Neue Liste
Listenname
SAVE


Liste: Spar
Wurst
Amount: 30x
Käse
Amount: 15x
Paprika
Amount: 3x


Spar: Neuer Posten
Postenname
Anzahl
SAVE

4 Backend Development

Die Geschäftslogik der Beispiel Applikation „Me Want That“ wird, wie bei den meisten Client/Server Applikationen, auf der Serverseite implementiert. Dafür verwenden wir zur Datenspeicherung die Datenbank **MySQL** und als Server-Sprachumgebung die Programmiersprache **PHP** auf einem Apache Webserver. Die Entscheidung fiel auf diese Tools, da sie nicht nur sehr weit verbreitet, sondern außerdem auch außerordentlich einfach zu deployen beziehungsweise auszurollen sind: Die Kombination PHP/MySQL ist günstig sowie schnell und unkompliziert verfügbar.

Wieso beschreibe ich in dieser Arbeit, in der es um App Development gehen soll, eine REST API? Es soll um den Ablauf eines Beispielprojektes gehen, welches einer ähnlichen Form auch in der *„echten“* Entwicklerwelt vorkommen kann.

Ein **Full-Stack Zugang**, also ein ganzheitlicher Blick auf das Thema App-Development, beginnt und endet nicht am mobilen Endgerät, sondern knüpft an weitere wichtige Themengebiete wie Speichern der Daten in eine Datenbank, Deployment der App in einen App-Store, API Design, Verbindungsaufbau und Kommunikation an. Das Thema API-Design und Datenhaushalt erscheint mir hier besonders relevant und wichtig.

Ich stelle in diesem Kapitel das von mir aufgrund der hohen Verbreitung, Einfachheit und Relevanz präferierte Tool ***„Laravel“*** als Model View Controller Framework für die Backend PHP-Umgebung vor.

4.1 Beschreiben der REST API für die Beispiel-App

Um ein Verständnis zu erlangen, was genau eine RESTful API ist und warum wir diese benötigen um den verschiedenen Clients wie Apps und Websites eine Kommunikation mit der Businesslogic unserer Anwendung zu ermöglichen, erkläre ich im Folgenden, wie sich diese Schnittstelle aufbaut, wie sie verwendet wird und aus welchen Komponenten sie besteht.

> *„REST steht für **RE**presentational **S**tate **T**ransfer, API für **A**pplication **P**rogramming **I**nterface. Gemeint ist damit eine Programmierschnittstelle, die sich an den Paradigmen und Verhalten des World Wide Web (WWW) orientiert und einen Ansatz für die Kommunikation zwischen Client und Server in Netzwerken beschreibt." (Was ist eine REST API 2017, p. 1)*

In diesem Client/Server Anwendungsbeispiel ist die REST API die Serverkomponente für unsere Beispiel Applikation und im online verfügbar. Unsere mit verschiedenen Frameworks erstellten Apps (siehe nachfolgende Kapitel) kommunizieren mit dieser API. Das bedeutet, die Apps holen und sichern Daten über diese Schnittstelle aus und in die Datenbank oder triggern Events zur Datenverarbeitung.

Die Verständigung beruht auf einem zustandslosen Protokoll, im Normalfall http, für die Interaktion.

Die Aktionen, welche über die Schnittstelle durchgeführt werden können, beschreiben sich über die http Verben und halten sich an deren semantische Bedeutung:

- GET: Daten oder Ressourcen abrufen
- POST: Daten oder Ressourcen erzeugen
- PUT: Daten oder Ressourcen updaten beziehungsweise verändern
- DELETE: Daten oder Ressourcen löschen

4.2 Datenbank

Natürlich muss diese Datenbank erst entworfen werden. Da es sich um eine Beispielanwendung dreht, wird das Datenmodell und die Datenbankstruktur bewusst sehr einfach gehalten. Das hat auch eine reduzierte Anzahl an CRUD Endpunkten in der API zur Folge und soll die Anwendung einfach und übersichtlich halten.

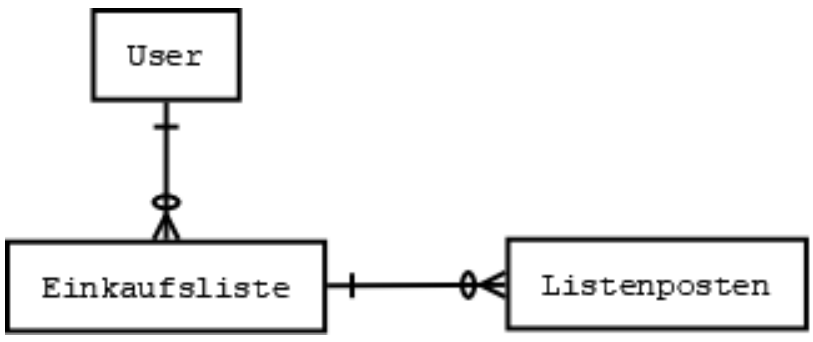
4.2.1 ER (Entity Relations) Modell

In der Applikation werden nur drei verschiedene Entitäten vorkommen. Die Abhängigkeiten sehen wir folgt aus:

Ein User oder Benutzer repräsentiert eine natürliche Person, welche eigene Einkaufslisten für sich anlegen kann. Somit gehört jede Liste genau einem User, und jeder User kann mehrere Listen anlegen.

Einkaufslisten stellen den Ordnungsbegriff des Containers für Listenposten dar. Ein Listenposten kann sich nur auf genau einer Einkaufsliste befinden. Jede Einkaufsliste kann null oder mehrere Listenposten aufweisen.

Eine grafische Darstellung dieser Abhängigkeiten und Entitäten lässt sich im sogenannten ER (Entity Relation) Modell visualisieren:

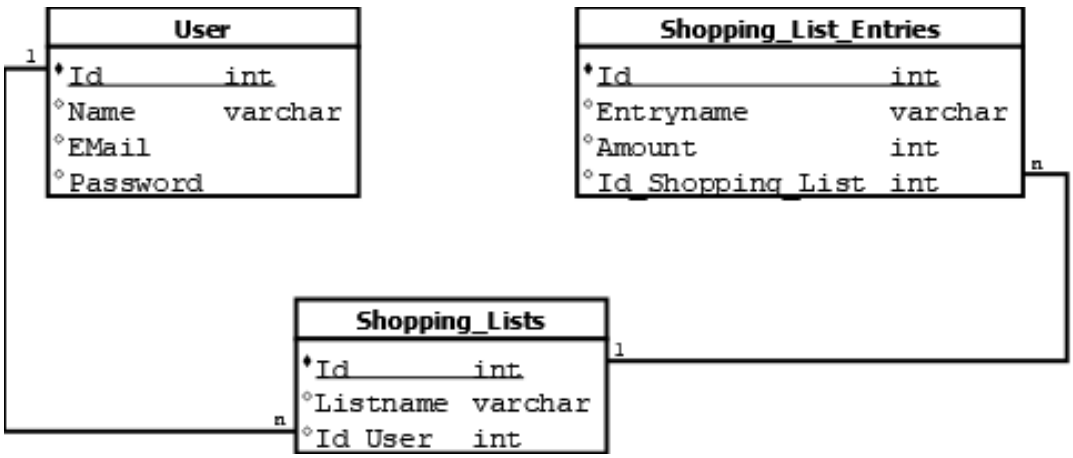


Obwohl diese Anwendung sich einfach und übersichtlich darstellt, können ER-Modelle von komplexeren Applikationen mit vielen Ordnungsbegriffen und Entitäten unübersichtlich werden. Sie sind allerdings für ein Verständnis der

Applikation unbedingt notwendig. Wer die natürlichen Zusammehänge der zugrundeliegenden Daten einer Applikation nicht versteht, oder diese Daten nicht in einfachen 1:n Abhängigkeiten ausdrücken kann, wird bei der Implementierung von Front- und Backend erhebliche Schwierigkeiten haben.

4.2.2 Relationales Datenbankmodell

Das relationale Datenbankmodell folgt sehr streng dem ER-Modell. Für jede der drei Entitäten wird es eine Tabelle geben. Der Primärschlüssel wird in allen drei Fällen ein aufsteigender Integer Identifier sein. Vom User wird zusätzlich der Name, die E-Mail-Adresse und das Passwort gespeichert. Die Einkaufsliste erhält, neben dem Fremdschlüsselfeld für die ID des Users, als einziges Feld den Namen der Liste. Die einzelnen Listenposten erhalten einen Namen, eine Anzahl und ein Fremdschlüsselfeld, welche den Posten auf eine Einkaufsliste verortet.



4.3 REST Endpunkte

Für jede Entität müssen entsprechend alle notwendigen **CRUD** (**C**reate, **R**ead, **U**pdate, **D**elete) Operationen zur Verfügung gestellt werden. Sturgeon schreibt in seinem Werk *„Build API's you won't hate"* dazu folgende goldrichtigen Sätze:

> *„Try thinking of everything your API will need to handle. This will intitially be a list of CRUD (create, read, update, delete) endpoints for your resources. Talk to your mobile app developer, your JS front-end people, or just talk to yourself if you are the only developer on the project" (Sturgeon and Bohill, 2015, p. 13)*

Jede CRUD Operation erhält eine eigene, fixe **URL,** über die der Befehl ausgeführt werden kann. Für unsere Beispielanwendung *„Me Want That"* benötigen wir folgende Endpunkte:

- GET /lists
- POST /list
- DELETE /list/{id}
- GET /list/{list-id}/entries
- POST /list/{list-id}/entry
- DELETE /entry/{id}
- POST /login
- POST /logout
- POST /register
- POST /unregister

Mit diesem Set an Anweisungen ist es uns möglich, über die API-Schnittstelle die notwendigen Schritte wie Anlegen und Löschen von Einkaufslisten sowie Anlegen und Löschen von einzelnen Posten auf diesen Einkaufslisten

durchzuführen. Außerdem können wir neue User registrieren, einloggen, ausloggen und wieder deregistrieren.

Ich weise ausdrücklich darauf hin, dass diese Anwendung eine Veranschaulichung der Vorgehensweise von Datenkommunikation zwischen Backend (PHP/Laravel) und Frontend (Cross Plattform Apps) darstellt und bewusst Themen wie Userverwaltung, Security, Verschlüsselung von Daten oder Nachverfolgbarkeit und Logging vernachlässigt werden.

4.4 Implementierung mit dem Laravel Framework (PHP)

Zur Implementierung dieser Schnittstelle verwende ich das PHP-Framework Laravel.

Der Grund dafür ist einfach:

Laravel ist zum Zeitpunkt dieser Arbeit das Framework, welches sich durch Einfachheit in der Entwicklung, Übersichtlichkeit und sehr einfaches Deployment auszeichnet. Es basiert auf der Skriptsprache PHP und ist laut der Website *http://dev.to* im Jahr 2020 nach *Phoenix* (ein Backend-Tool basierend auf der Erlang Virtual Machine) das zweittrendigste Backend-Framework auf dem Markt.

Antony Gore beschreibt Laravel wie folgt:

> *„Laravel is an open source MVC framework for PHP that is used to build robust web applications. Laravel is currently at version 5.5 [Anmerkung: Die aktuelle Version zum Zeitpunkt dieser Arbeit ist 6.17] and is among the most popular PHP frameworks, beloved for its elegant syntax and powerful features.“*
>
> *(Gore, 2017, p. 69)*

Es gibt noch unzählige andere Backend-Solutions und Sprachen die großartige Arbeit leisten und viele Seiten und Apps im Hintergrund befeuern und betreiben. Um einige zu nennen:

- Microsoft ASP.Net
- Java Spring
- Node.JS Express.JS
- PHP Symfony

4.4.1 Github Repository

Der gesamte Sourcecode ist öffentlich unter folgendem GitHub Repository verfügbar:

https://github.com/MikesDevCorner/MeWantThat-backend-laravel

4.4.2 Laravel Model-View-Controller

Laravel ist ein PHP-Framework, welches dem Model-View-Controller Programmiermuster folgt und dem Developer durch sein genau definiertes Gerüst und seine vielen Features und integrierten Komponenten Arbeit abnimmt und die Aufmerksamkeit des Entwicklers auf der Business Logic bündelt, ohne dass zu viel Energie in die Programmierung der Infrastruktur fließen muss.

> „Beim Model-View-Controller (auch MVC genannt) handelt es sich um ein Set von Design Patterns, die eingesetzt werden, um die einzelnen Schichten der Applikation voneinander zu trennen"
> *(Schmidt, 2007, p. 354)*

Wie so oft in der Software Entwicklung wird eine Trennung von Kompetenzen angestrebt, welche Stefan Schmidt als Schichtentrennung innerhalb der Applikation erklärt.

MVC-Frameworks bedienen sich dabei verschiedener Entwurfsmuster, wie dem *Strategy Pattern*, dem *Composite Pattern* und dem *Observer Pattern* und kombinieren diese intelligent miteinander. Das Resultat ist ein zusammengesetztes Entwurfsmuster, welches den Entwickler stark an der Hand nimmt und bereits viele Lösungen zu gängigen Problemstellungen anbietet. Eric und Elisabeth Freeman beschreiben das MVC-Pattern mit den Worten „Der König der zusammengesetzten Muster“ (Freeman et al., 2005, p. 526)

Sie erklären die einzelnen Bestandteile wie folgt:

- ***View:*** *Liefert Ihnen eine Präsentation des Models. Der View erhält normalerweise den Zustand und die Daten, die er benötigt, direkt vom Model*

- ***Controller:*** *Nimmt die Eingabe des Benutzers an und stellt fest, was sie für das Model bedeutet*

- ***Model:*** *Das Model enthält die gesamte Daten-, Zustands- und Anwendungslogik. Es weiß nichts über View und Controller, allerdings bietet es eine Schnittstelle, über die sein Zustand beeinflusst und abgerufen werden kann, und es kann Benachrichtigungen über Zustandsänderungen an Beobachter senden.*

  (Freeman et al., 2005, p. 530)

Im Falle unserer einfachen REST API werden wir aus diesem Entwurfsmuster lediglich die **Controller** und **Model** Komponenten benötigen. Unsere verschiedene **Hybrid-Apps** werden als **View-Komponente** fungieren, welche die entsprechenden Daten direkt über die API Schnittstelle abrufen.

Wir werden also drei Model Klassen erstellen, je eines für unsere Entitäten aus dem *Entity Relation Model* (siehe 4.2.1). Des Weiteren benötigen wir einen Controller, welcher die Eingaben der Benutzer entgegennimmt und die Zustandsveränderungen oder Zustandsabfragen der Models dirigiert.

Laravel bietet für die Erstellung von Webservices, immerhin eines der Kerneinsatzgebiete von Laravel, eine weitere Komponente an. Den **API-Router**. Dieser Backend-Router entscheidet anhand der übergebenen URL- sowie http-Parameter und anhand der verwendeten http Methode, welcher Controller und welche Controllermethode angesprochen werden muss, um den gewünschten Auftrag des Benutzers auszuführen.

4.4.3 Eloquent OR Mapper

Neben vielen weiteren Hilfsmitteln und Developer Comfort Tools bietet Laravel auch ein eigenes Werkzeug für **Object-Relational Mapping (ORM)** an. Diese Methode, die Datenschicht anzusprechen und abzufragen hat sich in den letzten Jahren weit verbreitet und findet in den gängigsten großen Frameworks mit unterschiedlichen Nomenklaturen Verwendung. In **JAVA** zum Beispiel ist der Name dieses Tools *„Hibernate“*, in ASP.Net *„Entity Framework“* und in PHP Laravel ***„Eloquent“***.

> *Object-Relational Mapping (ORM) is a technique for converting data between incompatible systems in object-oriented programming languages. Relational databases such as MySQL can only store scalar values such as integers and strings, organized within tables.*
>
> *(Gore, 2017, p. 87)*

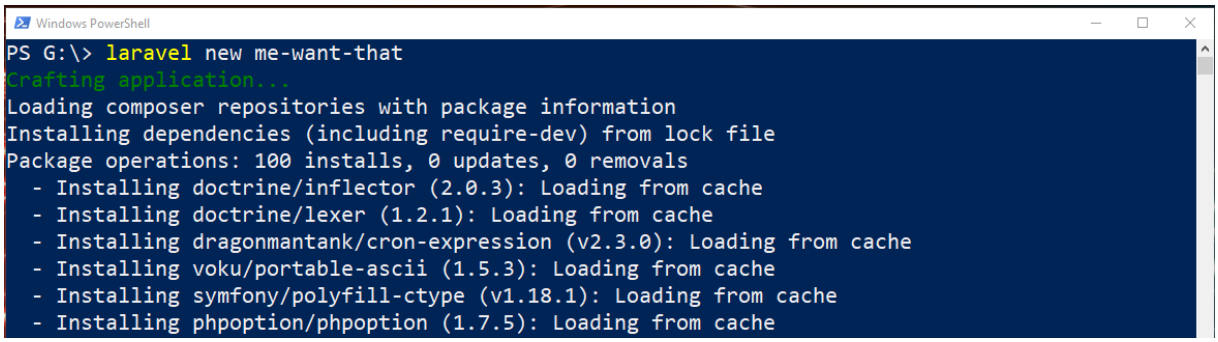
Eloquent dient somit als pseudo Zwischenschicht und erlaubt über Methoden auf den PHP Models (Facets) den direkten Zugriff auf die Datenschicht, ohne dass der Entwickler sich mit SQL beschäftigen muss.

4.4.4 Erstellen des Projektes

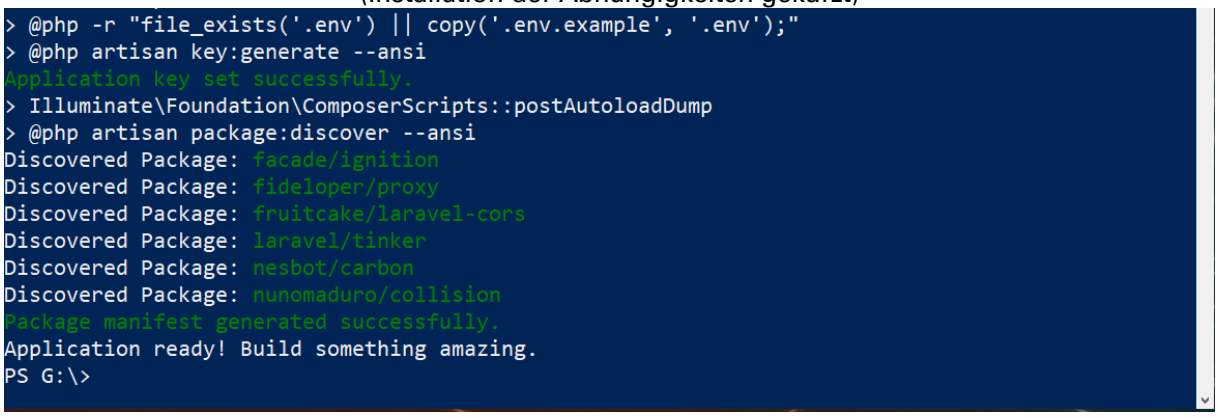
Nachdem die Prerequisites für Laravel erfüllt und auf der Development Maschine installiert und das Laravel Command Line Interface verfügbar ist, erstellt man mit dem Befehl

```
$ laravel new me-want-that
```

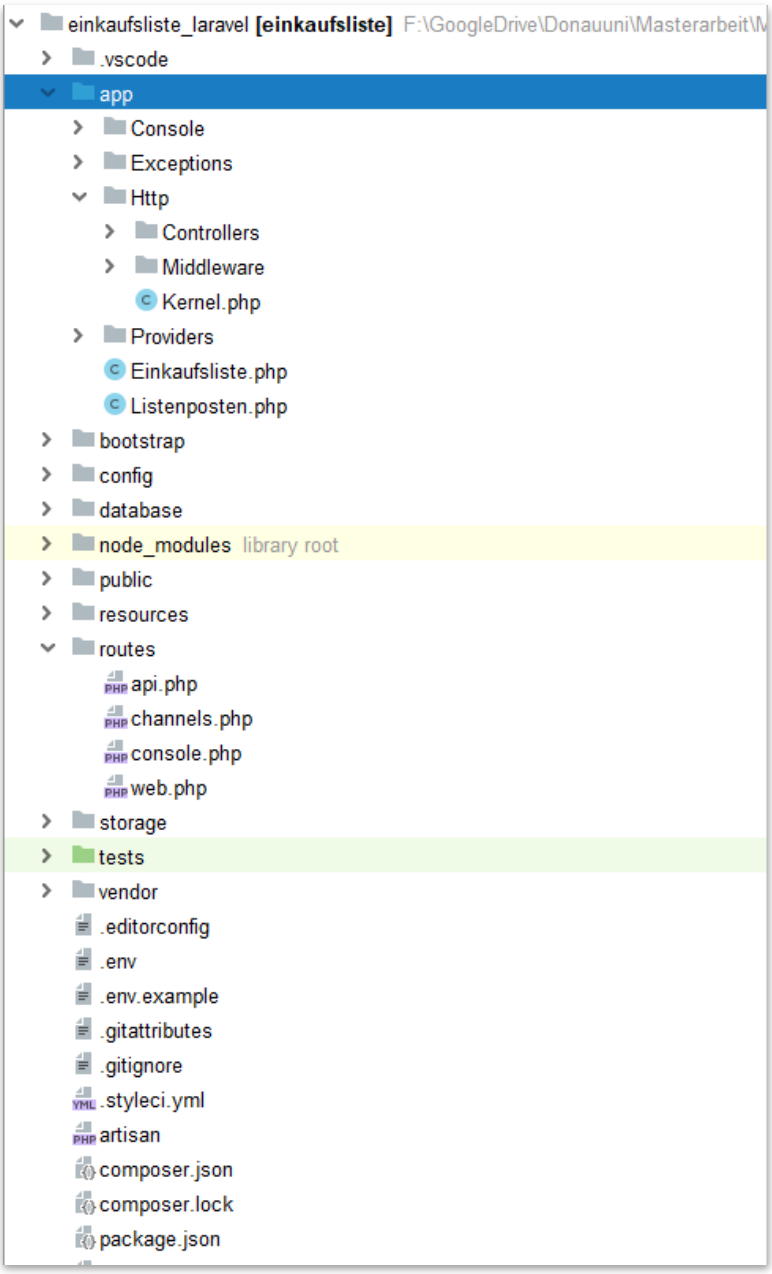
ein neues Laravel Projekt mit dem Namen der Beispielapplikation „**me-want-that“**. Dieses Projekt repräsentiert unsere **REST API** und ich werde in diesem Kapitel die benötigten **Models** und **Controllers** für die notwendigen Endpunkte anlegen und erklären.



(Installation der Abhängigkeiten gekürzt)



Nach dem Anlegen des Projektes über das Laravel **C**ommand **L**ine **I**nterface, erhalten wir eine Standard Laravel Projekt-Ordnerstruktur, welche sich wie folgt präsentiert:



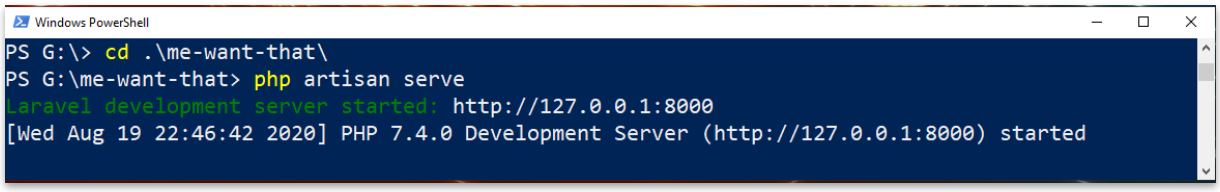
Für uns besonders wichtig sind die Ordner **app** wo die Model-Klassen und die Controller liegen, der Ordner **routes**, wo in der Datei „api.php“ die benötigten Routen definiert werden, sowie der Ordner **database**, worin die Migrationsfiles liegen, welche die Eloquent Models an die mySQL Datenschicht binden. In der Datei „**.env**“ werden die notwendigen Konfiguraitonen eingetragen. Eine Verbindung zu einer neuen, leeren mySQL Datenbank muss hier konfiguriert werden.

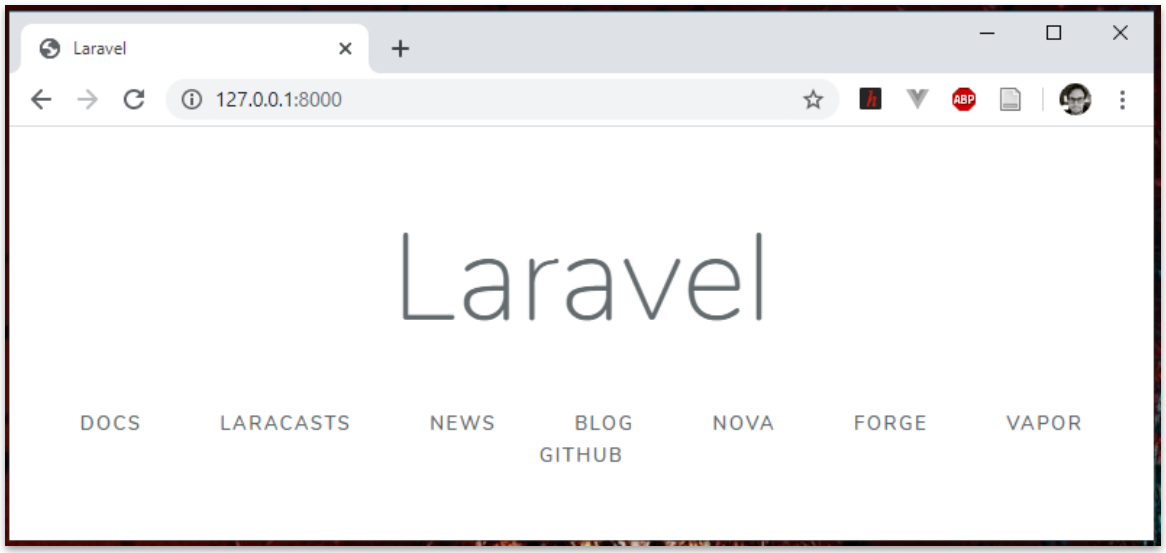
4.4.5 Starten des Development Servers

Nach dem erfolgreichen Erstellen des Lavavel-Projektes (siehe Screenshot) kann ein lokaler Development Server mit dem Befehl

```
$ php artisan serve
```

gestartet werden. Danach ist das Laravel Beispielprojekt im Browser über die im Terminal angegebene Adresse erreichbar. In weiterer Folge wird während der Entwicklung unter dieser Adresse unsere API erreichbar sein. Nach erfolgreicher Implementierung des Backends wird die API-Schnittstelle auf eine PHP Hostinginstanz deployed und die API wird unter der Adresse https://me-want-that.com/api abrufbar sein.





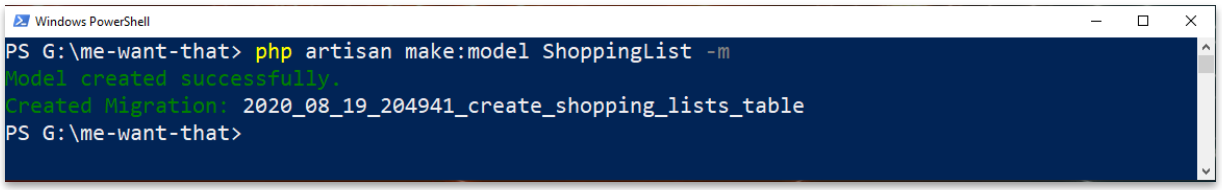
4.4.6 Eloquent

Für die Verständigung mit der Datenbank verwenden wir ***„Eloquent“***, ein ORM (**O**bject **R**elation **M**apper) nach dem Vorbild von Java's *Hibernate* oder ASP.Net's *Entity Framework* und integraler Bestandteil von Laravel. Diese Zwischenschicht erspart dem Developer die direkte Kommunikation mit der Datenbank über SQL (Structured Query Language) und vereinfacht so das Abrufen, Aggregieren und Speichern von Daten.

Mit Hilfe des Befehls

```
$ php artisian make:model ShoppingList -m
```

erstellt das Laravel CLI ein Modell für unsere Einkaufsliste und eine Datenbank-Migration dafür.



4.4.7 Datenbank Migration Files

Das entstandene Migrationsfile im Ordner /database/migrations muss um die fehlenden Felder, die wir im Datenbankmodell definiert haben, in der up() Methode ergänz, Indizes und Foreign-Keys müssen definiert werden. Danach wird entsprechend der gleichen Methodik (Erstellung und Anpassung) das zweite Modell „ShoppingListEntries“ erstellt. Eloquent wird daraus, sobald wir den entsprechenden Command absetzen, die Datenbanktabellen und -felder generieren. Für die User-Tabelle hat Laravel bereits eine Vorlage an Bord, welche unserem Datenmodell entspricht. Der automatische *$table->timestamps()* Befehl stattet die Tabellen mit Create- und Update- Zeitpuntken aus und sollte behalten werden. Danach sehen die 3 Migrationsdateien wie folgt aus:

```php
class CreateShoppingListsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('shopping_lists', function (Blueprint $table) {
            $table->id();
            $table->string('listname');
            $table->unsignedInteger('user_id');
            $table->foreign('user_id')->references('id')
                  ->on('users')->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('shopping_lists');
    }
}


class CreateShoppingListEntriesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('shopping_list_entries',
                       function (Blueprint $table) {
            $table->id();
            $table->string('entryname');
            $table->integer('amount');
            $table->unsignedInteger('shopping_list_id');
            $table->foreign('shopping_list_id')
               ->references('id')->on('shopping_lists')
               ->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('shopping_list_entries');
    }
}
```

```php
class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

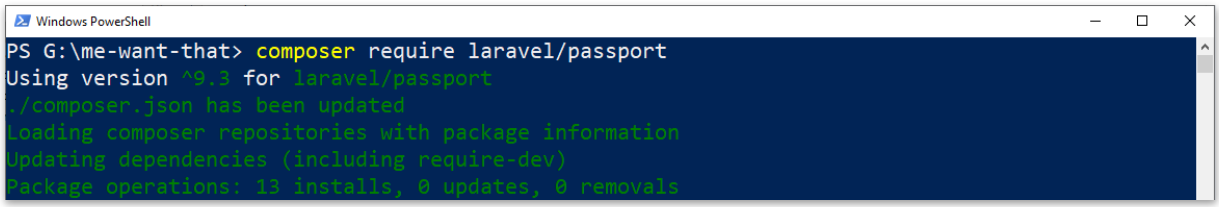
Die restlichen Migrationsfiles sind für diese Beispiel-App nicht von Relevanz und werden zu diesem Zeitpunkt ignoriert. Bevor die Datenbank erstellt wird, ein paar Worte zum Thema Security:

4.4.8 oAuth2 Passport Security

Wir werden im Laravel-Backend ein Paket aus der **Composer Paketverwaltung** nachinstallieren, welches automatisch einige **Authentication-Routes** erzeugt, eine **Authentifizierungs-Middleware** zur Verfügung stellt mit welcher wir unsere API-Routes vor unberechtigtem Zugriff schützen können und das Backend mit dem robusten und sehr sicheren **oAuth2** (Open Authentication 2) Verfahren ausstatten. Das Paket trägt den Namen ***„Passport“*** Da sich diese Arbeit bewusst nicht weiter mit dem breiten Fachgebiet der Security beschäftigt, werden an dieser Stelle keine weiteren Worte über dieses Verfahren verloren. Folgender

Befehl ist auf der Commandline auszuführen, um die Authentifizierung einzurichten:

```
$ composer require laravel/passport
```



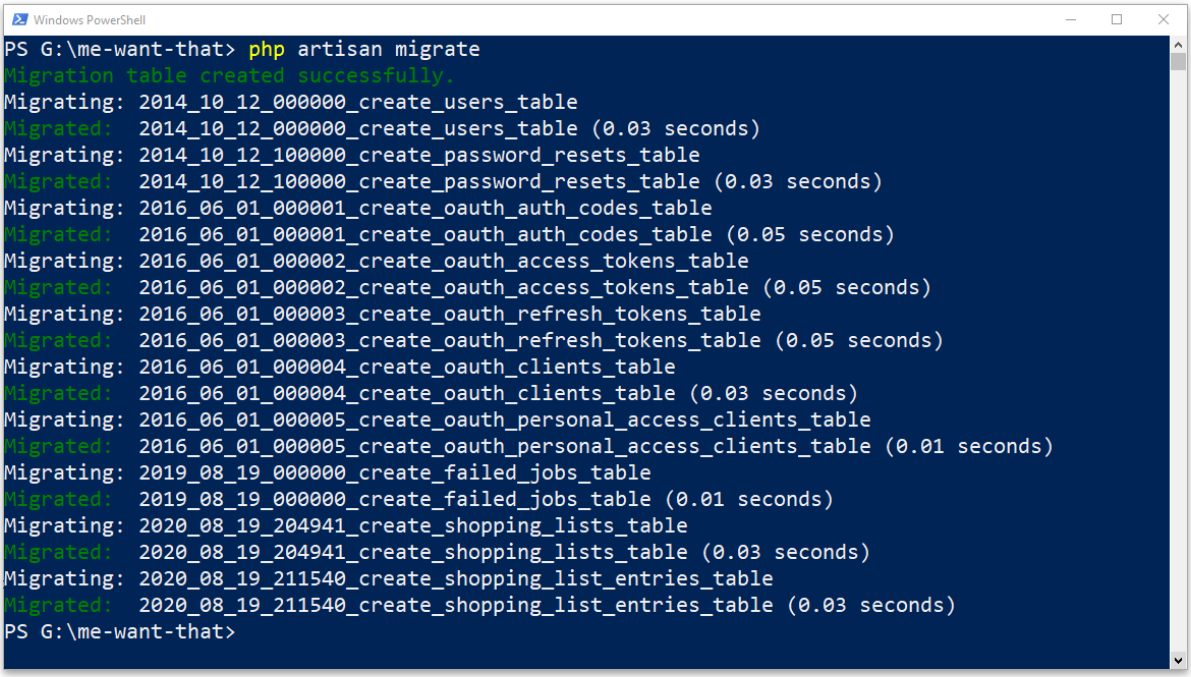
Der neu installierte Provider muss in der Datei config/app.php im Array ‚providers' eingetragen werden:

```
'providers' => [
…

/*
 * Package Service Providers...
 */
Laravel\Passport\PassportServiceProvider::class,
…
```

Zum Abschließen der Migration und Erstellen der Datenbank wird jetzt der Befehl

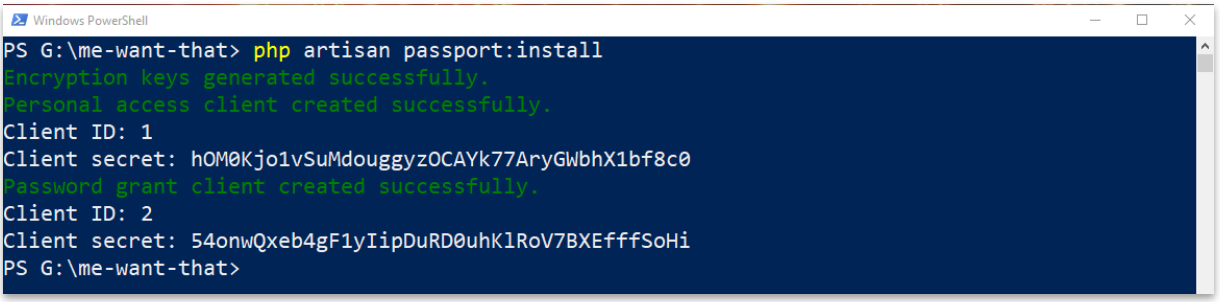
```
$ php artisan migrate
```

erwartet, welcher die Migrationsfiles liest und verarbeitet.



Damit das neu installierte Security-Plugin funktioniert müssen noch Encryption-Keys erstellt werden, was mit dem folgenden Befehl erledigt wird:

```
$ php artisan passport:install
```



Die Installation des Passport-Plugins ist somit fast erledigt. An 2 Stellen müssen noch Anpassungen vorgenommen werden, welche in den folgenden Abschnitten mit gelb markiert sind.

- app/Providers/AuthServiceProvider.php

```php
namespace App\Providers;

use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
use Illuminate\Support\Facades\Gate;
use Laravel\Passport\Passport;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        // 'App\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();
        Passport::routes();
    }
}
```

- config/auth.php

```php
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],

    'api' => [
        'driver' => 'passport',
        'provider' => 'users'
    ],
],
```

Als nächstes sehen wir uns die beiden Modell-Klassen an, welche wir zuvor erstellt haben, sowie die bereits bestehende User-Klasse an:

4.4.9 Model Klassen

Die zuvor über die Commandline generierten Migrations und Models haben uns im Ordner **/app** drei PHP-Klassen erstellt, welche unsere **Datenmodelle** widerspiegeln.

Innerhalb der Model-Klassen müssen die im Datenmodell fixierten **Datenfelder** angelegt werden. Außerdem stellen wir hilfreiche Referenzen auf die Foreign Key

Collections, sowie das Owner-Objekt her. Dafür erstellen wir Methoden „list“ und „entries“ auf den Modelklassen und verwenden die geerbten „hasMany“ sowie „belongsTo“ Methoden, um Laravel und Eloquent den Bezug klar zu machen. In der boot() Methode legen wir einen Handler an, welcher die Daten im Falle des Löschens kaskadierend löscht.

```
class ShoppingList extends Model
{
    protected $fillable = ['listname', 'user_id'];

    public function entries()
    {
        return $this->hasMany('App\ShoppingListEntry');
    }

    public function user()
    {
        return $this->belongsTo('App\User');
    }
```

```
    public static function boot() {
      parent::boot();
      static::deleting(function($list) { // cleanup
        foreach($list->entries as $entry) { $entry->delete(); }
      });
    }
}


class ShoppingListEntry extends Model
{
    protected $fillable = ['entryname', 'amount', 'shopping_list_id'];

    public function ShoppingList()
    {
        return $this->belongsTo('App\ShoppingList');
    }
}



use Laravel\Passport\HasApiTokens; // added

class User extends Authenticatable
{
    use Notifiable, HasApiTokens;

    protected $fillable = [
        'name', 'email', 'password',
    ];

    public function lists()
    {
        return $this->hasMany('App\ShoppingList');
    }

    protected $hidden = [
        'password', 'remember_token',
    ];

    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public static function boot() {
        parent::boot();

        static::deleting(function($user) { // cleanup
            foreach($user->lists as $list) { $list->delete(); }
        });
    }
}
```

4.4.10 API-Routen definieren

Nachdem diese wichtigen Grund- und Vorbereitungsarbeiten erledigt sind, kommen wir zum eigentlichen Anlegen der **API-Zugangspunkte**, den **Routen** und **Controllern**. Die gewünschten Routen haben wir bereits im Punkt 4.3 definiert, der nächste Schritt ist eine Umsetzung in Laravel.

Im ersten Schritt definieren wir die benötigten Routen, die auf unserer API vorhanden sein werden. Dafür muss das File **routes/api.php** bearbeitet werden:

```
use Illuminate\Http\Request;
use App\Http\Controllers\ShoppinglistController;
use App\Http\Controllers\Auth\ApiAuthController;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/


Route::middleware(['auth:api'])->group(function () {
    Route::get('lists/', 'ShoppinglistController@showLists');
    Route::post('list/', 'ShoppinglistController@createList');
    Route::delete('list/{idList}/', 'ShoppinglistController@deleteList');
    Route::get('list/{idList}/entries/', 'ShoppinglistController@showListEntries');
    Route::post('list/{idList}/entry/', 'ShoppinglistController@createEntry');
    Route::delete('entry/{id}/', 'ShoppinglistController@deleteEntry');
    Route::post('logout/', 'Auth\ApiAuthController@logout');
    Route::post('unregister/', 'Auth\ApiAuthController@unregister');
});


Route::group([], function () {
    Route::post('register/', 'Auth\ApiAuthController@register');
    Route::post('login/', 'Auth\ApiAuthController@login');
});
```

In diesem Schritt definieren wir die verfügbaren API-Endpunkte. und greifen über die (noch nicht existenten) Controller auf die Datenebene zu. Durch Gruppieren der Routen, welche eine Authentifizierung benötigen unter der Middleware auth:api wird eine von Passport gesteuerte Prüfung auf einen oAuth Access Token eingeleitet. Anschließend kümmert sich jede Controller-Methode um ein Ausliefern der Daten im JSON-Format.

Phil Sturgeon schreibt dazu:

> *„Any modern API you interact with will support JSON unless it is a financial service API, or the developer is a moron – probably both to be fair. Sometimes they will support XML too.“ (Sturgeon and Bohill, 2015, p. 31)*

Für unsere Einkaufslisten Test-Applikation ist eine Auslieferung im JSON Format ausreichend.

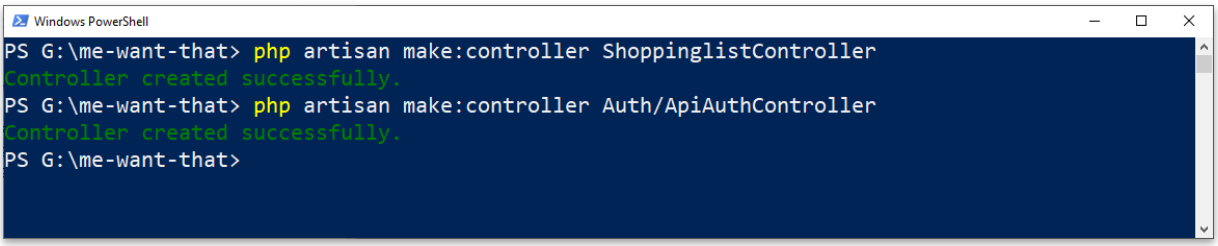
4.4.11 Controller

Wir haben bereits über das *Model-View-Controller* Programmiermuster gesprochen, und seine Aufgabe, Kompetenzen zu trennen und den Code unter Verwendung verschiedener Software Pattern in Aufgabencluster zu strukturieren.

Die richtige und vollständige Ausprägungsform verlangt von uns das Erstellen eines oder mehrerer *Controller*, welche die benötigten Methoden beherbergen, die von den einzelnen registrierten Routen angesprochen werden können. Auch hier sollte man einen Blick auf die Programm-Architektur werfen. Controller und Controllermethoden sollten sinnvoll nach Themenkomplexen gruppiert werden. Für unser begrenztes Beispiel mit den überschaubaren Funktionen, werden wir nur zwei Controller anlegen. Einen für die Domäne der Einkaufslistenverwaltung, eine für Themen der Authentifizierung. Dafür setzen wir folgende Befehle auf der Commandline ab:

```
$ php artisan make:controller ShoppinglistController
```

```
$ php artisan make:controller Auth/ApiAuthController
```



Diese Befehle erzeugen neuen Controller im Bereich **app/http/Controllers.**

In den Controllern werden nun die in den Routen angesprochenen Methoden definiert und ausprogrammiert.

Dazu ist anzumerken, dass auch diese Lösung nicht die beste Praxis ist, da ORM oder Query Builder Logic besser in austauschbare Services ausgelagert werden sollten. Wir verzichten darauf aber bewusst, um das Beispiel übersichtlich zu halten. Sturgeon schreibt dazu:

> *"Your controller should definitely not have this sort of ORM/-Query Builder logic scattered around the methods." (Sturgeon and Bohill, 2015, p. 59)*

Die beiden Controller sehen wie folgt aus:

app/Http/Controllers/ShoppinglilstController.php

```
use Illuminate\Http\Request;
use Illuminate\Http\Response;
use Illuminate\Support\Facades\Auth;
use App\ShoppingListEntry;
use App\ShoppingList;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Validator;

class ShoppinglistController extends Controller
{
/**
 * @OA\Get(
 *   path="/lists",
 *   tags={"List"},
 *   summary="Shows all available shopping lists for this user",
 *   operationId="showLists",
 *   security={
 *   {
 *       "passport": {}},
 *   },
```

```
*  @OA\Response(
 *      response=200,
 *      description="Success",
 *      @OA\MediaType(
 *           mediaType="application/json",
 *      )
 *   ),
 *  @OA\Response(
 *      response=401,
 *      description="Unauthenticated"
 *   ),
 *)
 **/
public function showLists() {
    return response()->json(ShoppingList::where('user_id', Auth::user()
                    ->id)->get(), Response::HTTP_OK);
}

/**
 * @OA\Post(
 *   path="/list",
 *   tags={"List"},
 *   summary="Creates a new shopping list for this user",
 *   operationId="createList",
 *   security={{"passport": {}},},
 *  @OA\Parameter(
 *      name="listname",
 *      in="query",
 *      required=true,
 *      @OA\Schema(
 *           type="string"
 *      )
 *   ),
 *  @OA\Response(
 *      response=201,
 *      description="Created",
 *      @OA\MediaType(
 *           mediaType="application/json",
 *      )
 *   ),
 *  @OA\Response(
 *      response=401,
 *      description="Unauthenticated"
 *   ),
 *   @OA\Response(
 *      response=400,
 *      description="Bad Request"
 *   )
 *)
 **/
public function createList(Request $request) {
    $validator = Validator::make($request->all(), [
      'listname' => 'required|max:60'
    ]);
    if ($validator->fails()) {
      return response()->json(['error' => $validator->errors()],
                                          Response::HTTP_BAD_REQUEST);
    }

    $vals = $request->all();
    $vals['user_id'] = Auth::user()->id;
    $newList = ShoppingList::create($vals);
    return response()->json($newList, Response::HTTP_CREATED);
}
```

```
/**
 * @OA\Delete(
 *   path="/list/{idList}",
 *   tags={"List"},
 *   summary="Deletes users shopping list with the given ID",
 *   operationId="deleteList",
 *   security={
 *   {
 *      "passport": {}},
 *   },
 *  @OA\Parameter(
 *      name="idList",
 *      in="path",
 *      required=true,
 *      @OA\Schema(
 *           type="integer"
 *      )
 *   ),
 *  @OA\Response(
 *      response=200,
 *      description="Success",
 *      @OA\MediaType(
 *           mediaType="application/json",
 *      )
 *   ),
 *  @OA\Response(
 *      response=401,
 *      description="Unauthenticated"
 *   ),
 *   @OA\Response(
 *      response=400,
 *      description="Bad Request"
 *   )
 *)
 **/
public function deleteList(int $idList) {
  try {
    $list = ShoppingList::where('user_id', Auth::user()->id)
                          ->where('id', '=', $idList)->firstOrFail();
  } catch(\Exception $ex) {
    return response()->json(['error' => 'list not found for user '.Auth::user()
                             ->id], Response::HTTP_BAD_REQUEST);
  }
  $list->delete();
  return response()->json(null, Response::HTTP_OK);
}

/**
 * @OA\Get(
 ** path="/list/{idList}/entries",
 *   tags={"List"},
 *   summary="Shows all available shopping list entries from the
 *            given shopping list for this user",
 *   operationId="showListEntries",
 *   security={
 *   { "passport": {}}, },
 *  @OA\Parameter(
 *      name="idList",
 *      in="path",
 *      required=true,
 *      @OA\Schema(
 *           type="integer"
 *      )
 *   ),
```

```
*  @OA\Response(
 *      response=200,
 *      description="Success",
 *      @OA\MediaType(
 *           mediaType="application/json",
 *      )
 *   ),
 *   @OA\Response(
 *      response=401,
 *      description="Unauthenticated"
 *   ),
 *   @OA\Response(
 *      response=400,
 *      description="Bad Request"
 *   )
 *)
 **/
public function showListEntries(int $idList) {
    try {
      $list = ShoppingList::where('user_id', Auth::user()->id)
                          ->where('id', $idList)->firstOrFail();
    } catch(\Exception $ex) {
      return response()->json(['error' => 'list not found for user '
                               .Auth::user()->id], Response::HTTP_BAD_REQUEST);
    }
    return response()->json($list->entries, Response::HTTP_OK);
}


/**
 * @OA\Post(
 *   path="/list/{idList}/entry",
 *   tags={"List"},
 *   summary="Creates a new shopping list entry on the given
              shopping list for this user",
 *   operationId="createListEntry",
 *   security={
 *   {
 *      "passport": {}},
 *   },
 *  @OA\Parameter(
 *      name="idList",
 *      in="path",
 *      required=true,
 *      @OA\Schema(
 *           type="integer"
 *      )
 *   ),
 *  @OA\Parameter(
 *      name="entryname",
 *      in="query",
 *      required=true,
 *      @OA\Schema(
 *           type="string"
 *      )
 *   ),
 *   @OA\Parameter(
 *      name="amount",
 *      in="query",
 *      required=true,
 *      @OA\Schema(
 *           type="integer"
 *      )
 *   ),
```

```
     *  @OA\Response(
     *      response=200,
     *      description="Success",
     *      @OA\MediaType(
     *           mediaType="application/json",
     *      )
     *   ),
     *  @OA\Response(
     *      response=401,
     *      description="Unauthenticated"
     *   ),
     *   @OA\Response(
     *      response=400,
     *      description="Bad Request"
     *   )
     *)
     **/
    public function createEntry(Request $request, int $shopping_list_id) {
        $validator = Validator::make($request->all(), [
          'entryname' => 'required|max:60',
          'amount' => 'required|numeric'
        ]);
        if ($validator->fails()) {
          return response()->json(['error' => $validator->errors()],
                                  Response::HTTP_BAD_REQUEST);
        }
        $vals = $request->all();
        try {
          $list = ShoppingList::where('user_id', Auth::user()->id)
                              ->where('id', $shopping_list_id)->firstOrFail();
        } catch(\Exception $ex) {
          return response()->json(['error' => 'list not found for
                    user '.Auth::user()->id], Response::HTTP_BAD_REQUEST);
        }
        $vals['shopping_list_id'] = $list->id;
        $newPosten = ShoppingListEntry::create($vals);
        return response()->json($newPosten, Response::HTTP_CREATED);
    }
    /**
     * @OA\Delete (
     ** path="/entry/{id}",
     *   tags={"List"},
     *   summary="Deletes users shopping list entry with the given ID,
         regardless on wich list it is. It must be owned by the user, tho.",
     *   operationId="deleteListEntry",
     *   security={
     *   {
     *      "passport": {}},
     *   },
     *  @OA\Parameter(
     *      name="$id",
     *      in="path",
     *      required=true,
     *      @OA\Schema(
     *           type="integer"
     *      )
     *   ),
     *  @OA\Response(
     *      response=200,
     *      description="Success",
     *      @OA\MediaType(
     *           mediaType="application/json",
     *      )
     *   ),
```

```
     *  @OA\Response(
     *      response=401,
     *      description="Unauthenticated"
     *   ),
     *   @OA\Response(
     *      response=400,
     *      description="Bad Request"
     *   )
     *)
     **/
    public function deleteEntry(int $id) {
        $entry = ShoppingListEntry::find($id);
        if($entry->ShoppingList->user_id != Auth::user()->id) {
          return response()->json(['error' => 'list not found'],
                                  Response::HTTP_BAD_REQUEST);
        } else {
          $entry->delete();
          return response()->json(null, Response::HTTP_OK);
        }
    }
}
```

app/Http/Controllers/Auth/ApiAuthController.php

```
use App\Http\Controllers\Controller;
use App\User;
use Illuminate\Http\Request;
use Illuminate\Http\Response;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Facades\Auth;


class ApiAuthController extends Controller
{
    /**
     * @OA\Post(
     ** path="/login",
     *   tags={"Simple Auth"},
     *   summary="Logs user in and creates new auth token",
     *   operationId="login",
     *
     *   @OA\Parameter(
     *      name="email",
     *      in="query",
     *      required=true,
     *      @OA\Schema(
     *           type="string"
     *      )
     *   ),
     *   @OA\Parameter(
     *      name="password",
     *      in="query",
     *      required=true,
     *      @OA\Schema(
     *          type="string"
     *      )
     *   ),
```

```
*    @OA\Response(
 *        response=202,
 *         description="Accepted",
 *        @OA\MediaType(
 *             mediaType="application/json",
 *        )
 *    ),
 *    @OA\Response(
 *        response=401,
 *         description="Unauthenticated: Authorization information
                       is missing or invalid."
 *    )
 *)
 **/
public function login(Request $request)
{
    $validator = Validator::make($request->all(), [
        'email' => 'required|email',
        'password' => 'required'
    ]);

    if ($validator->fails()) {
        return response()->json(['error' => $validator->errors()],
Response::HTTP_UNAUTHORIZED);
    }

    if (!auth()->attempt($validator->validate())) {
        return response()->json(['error' => 'Unauthorised'],
Response::HTTP_UNAUTHORIZED);
    } else {
        $success['token'] = auth()->user()->createToken('authToken')
                                    ->accessToken;
        $success['user'] = auth()->user();
        return response()->json(['success' => $success])
                         ->setStatusCode(Response::HTTP_ACCEPTED);
    }
}

/**
 * @OA\Post(
 ** path="/register",
 *   tags={"Simple Auth"},
 *   summary="Register new user",
 *   operationId="register",
 *
 *  @OA\Parameter(
 *      name="name",
 *      in="query",
 *      required=true,
 *      @OA\Schema(
 *           type="string"
 *      )
 *    ),
 *  @OA\Parameter(
 *      name="email",
 *      in="query",
 *      required=true,
 *      @OA\Schema(
 *           type="string"
 *      )
 *    ),
```

```
*    @OA\Parameter(
 *        name="password",
 *        in="query",
 *        required=true,
 *        @OA\Schema(
 *             type="string"
 *        )
 *    ),
 *        @OA\Parameter(
 *        name="password_confirmation",
 *        in="query",
 *        required=true,
 *        @OA\Schema(
 *             type="string"
 *        )
 *    ),
 *    @OA\Response(
 *        response=201,
 *         description="Success",
 *        @OA\MediaType(
 *             mediaType="application/json",
 *        )
 *    ),
 *    @OA\Response(
 *        response=400,
 *         description="Bad Request"
 *    )
 *)
 **/
public function register(Request $request)
{
    $validator = Validator::make($request->all(), [
        'name' => 'required',
        'email' => 'required|email|unique:users',
        'password' => 'required|confirmed'
    ]);

    if ($validator->fails()) {
        return response()->json(['error' => $validator
                        ->errors()], Response::HTTP_BAD_REQUEST);
    }

    $input = $request->all();
    $input['password'] = Hash::make($input['password']);
    $user = User::create($input);
    $success['token'] =  $user->createToken('authToken')->accessToken;
    $success['user'] = $user;
    return response()->json(['success' => $success])
                    ->setStatusCode(Response::HTTP_CREATED);
}
/**
 * @OA\Post(
 ** path="/logout",
 *   tags={"Simple Auth"},
 *   summary="Revokes current access token",
 *   operationId="logout",
 *   security={
 *   {
 *       "passport": {}},
 *   },
```

```
     *   @OA\Response(
     *      response=200,
     *       description="Success",
     *      @OA\MediaType(
     *           mediaType="application/json",
     *      )
     *   ),
     *   @OA\Response(
     *      response=401,
     *       description="Unauthenticated"
     *   )
     *)
     **/
    /**
     * details api
     *
     * @return \Illuminate\Http\Response
     */
    public function logout(Request $request)
    {
      $request->user()->token()->revoke();
      return response()->json(null, Response::HTTP_OK);
    }


  /**
   * @OA\Post(
   ** path="/unregister",
   *   tags={"Simple Auth"},
   *   summary="Deletes all user data from system",
   *   operationId="unregister",
   *   security={
   *   {
   *      "passport": {}},
   *   },
   *   @OA\Response(
   *      response=200,
   *       description="Success",
   *      @OA\MediaType(
   *           mediaType="application/json",
   *      )
   *   ),
   *   @OA\Response(
   *      response=401,
   *       description="Unauthenticated"
   *   )
   *)
   **/
  /**
   * details api
   *
   * @return \Illuminate\Http\Response
   */
  public function unregister(Request $request)
  {
    $user = User::where('id', Auth::user()->id)->firstOrFail();
    $user->delete();
    return response()->json(null, Response::HTTP_OK);
  }
}
```

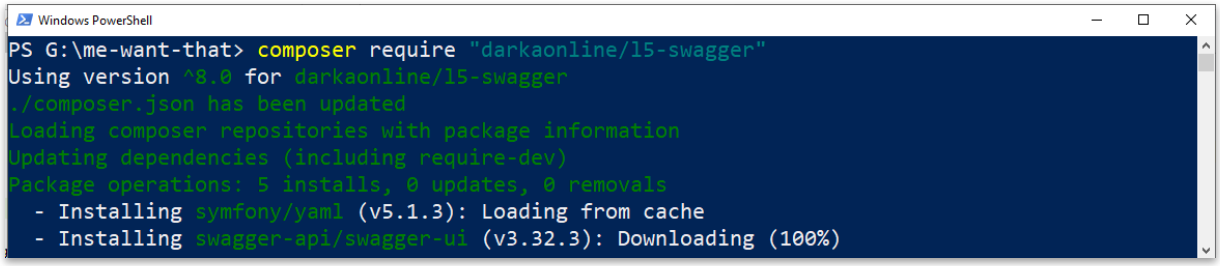
Zu beachten sind die Annotationen und Beschreibungen über jeder Methode, welche im späteren Verlauf eine automatisierte API-Dokumentation im Open Documentation Standard und Auslieferung mit dem Tool Swagger erlaubt.

Somit ist die neue API so gut wie fertig und für unser Beispiel bestens gerüstet. Als nächstes muss getestet werden, ob die Endpunkte wie erwartet ansprechbar sind.

4.4.12 Api Dokumentation mit Swagger / OpenAPI

OpenAPI ist ein Standard zur Beschreibung von REST-konformen Programmierschnittstellen. Dabei wird ein Definitionsfile erzeugt oder generiert, welches diesem Standard folgt. Swagger ist eine Sammlung von Werkzeugen, welche dieses Definitionsfile erzeugen und anzeigen kann. Mit der Swagger UI können die einzelnen API-Endpunkte auch getestet und ausprobiert werden. Um die Swagger Dokumentation für unser REST Backend zu erzeugen, benötigen wir ein weiteres Paket aus der Composer Paketverwaltung, welches wir mit folgendem Befehl nachinstallieren:

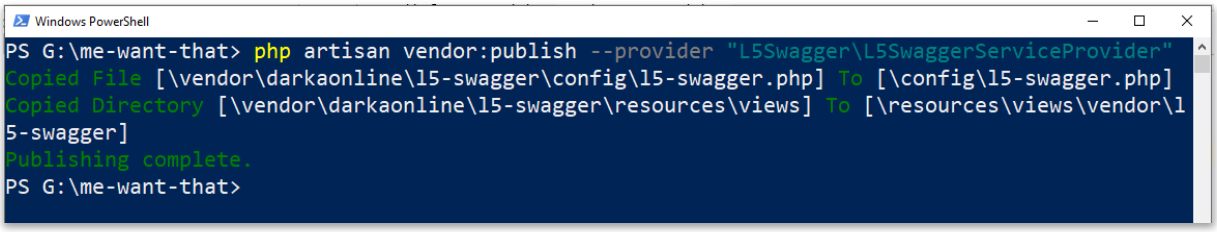
```
$ composer require "darkaonline/l5-swagger:7.*"
```



Nach erfolgreicher Installation des Paketes müssen 3 Konfigurationsschritte ausgeführt werden:

- Veröffentlichen der View- und Konfigurationsdateien

```
$ php artisan vendor:publish --provider
"L5Swagger\L5SwaggerServiceProvider"
```



- Anpassen der Swagger Konfuguration unter /config/l5-swagger.php

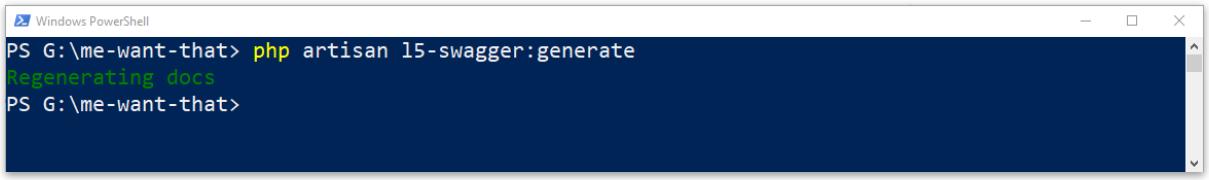
```php
<?php

return [
    'api' => [
        /*
        |---------------------------------------------------------------------
        | Edit to set the api's title
        |---------------------------------------------------------------------
        */
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
        \Laravel\Passport\Http\Middleware\CreateFreshApiToken::class,
        'auth',
        'title' => 'Me Want That API Docs',
    ],
…
/*
|---------------------------------------------------------------------------
| API security definitions. Will be generated into documentation file.
|---------------------------------------------------------------------------
*/
'security' => [
    // Open API 3.0 support
    'passport' => [ // Unique name of security
        'type' => 'oauth2', // The type of the security scheme.
                            // Valid values are "basic", "apiKey" or "oauth2".
        'description' => 'Laravel passport oauth2 security.',
        'in' => 'header',
        'scheme' => 'https',
        'flows' => [
            "password" => [
                "authorizationUrl" => config('app.url') .
                                      '/oauth/authorize',
                "tokenUrl" => config('app.url') . '/oauth/token',
                "refreshUrl" => config('app.url') . '/token/refresh',
                "scopes" => []
            ],
        ],
    ],
],
```

- Anpassen der Controller Basis-Klasse mit ein paar Annotationen unter /app/Http/Controllers/Controller.php:

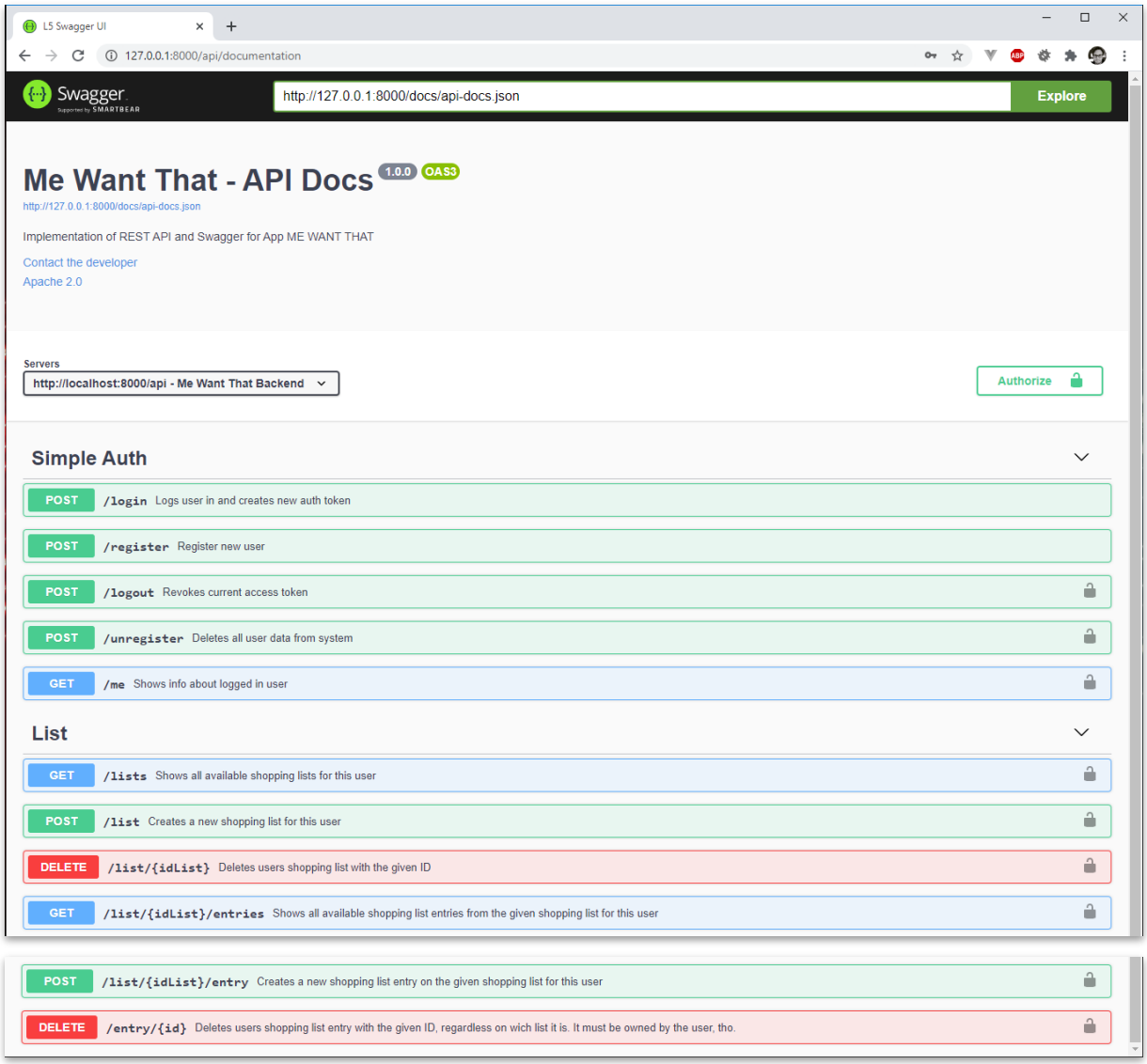
```php
/**
 * @OA\Info(
 *      version="1.0.0",
 *      title="Me Want That - API Docs",
 *      description="Implementation of REST API and Swagger for App ME WANT THAT",
 *      @OA\Contact(
 *          email="wagnmich@gmail.com"
 *      ),
 *      @OA\License(
 *          name="Apache 2.0",
 *          url="http://www.apache.org/licenses/LICENSE-2.0.html"
 *      )
 * )
 *
 * @OA\Server(
 *      url=L5_SWAGGER_CONST_HOST,
 *      description="Me Want That Backend"
 * )

 *
 *
 */
use AuthorizesRequests, DispatchesJobs, ValidatesRequests;
```

- Generieren der API-Docs mit dem Befehl

```
$ php artisan l5-swagger: generate
```



Nachdem die notwendigen Schritte erledigt und Konfigurationen vorgenommen wurden, erreichen wir ab sofort unter der Adresse **/api/documentation** unsere API-Beschreibung. Dokumentationen von Rest Schnittstellen sind für die Softwareentwicklung unerlässlich, da sie die Anbindung von externen Systemen ungemein erleichtern.



4.4.13 Testen der API-Endpunkte

Nachdem die API-Schnittstelle für unsere Testapplikation *„Me Want That“* jetzt steht und funktionstüchtig ist, sollten wir die Endpunkte **testen**. Softwaretesting ist in jeder Entwicklung und in jedem Entwicklungsschritt wichtig. Dieses Thema an sich stellt einen so umfangreichen Themenkomplex dar, dass eigene Masterthesen sich ausschließlich damit beschäftigen. An dieser Stelle möchte ich erwähnen, dass **keine** automatisierten Softwaretests und keine normierten Softwaretools in dieser Masterthesis verwendet werden.

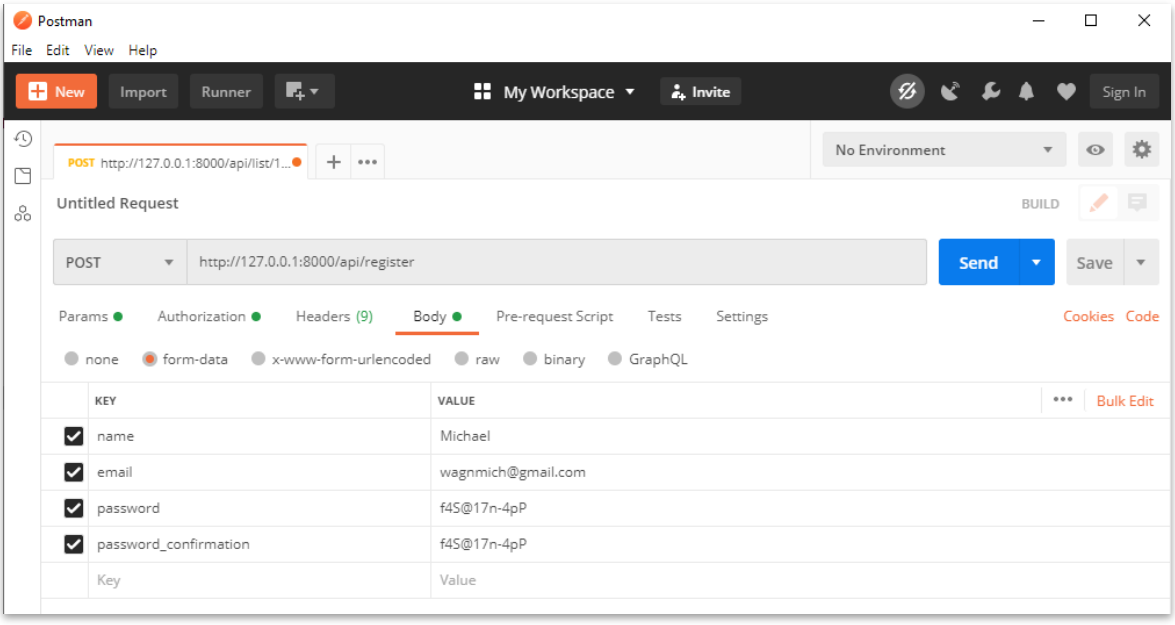
Allerdings wollen und sollen wir eine Möglichkeit haben, die entsprechenden von uns geschaffenen Endpunkte einzeln anzusprechen und auszuprobieren.

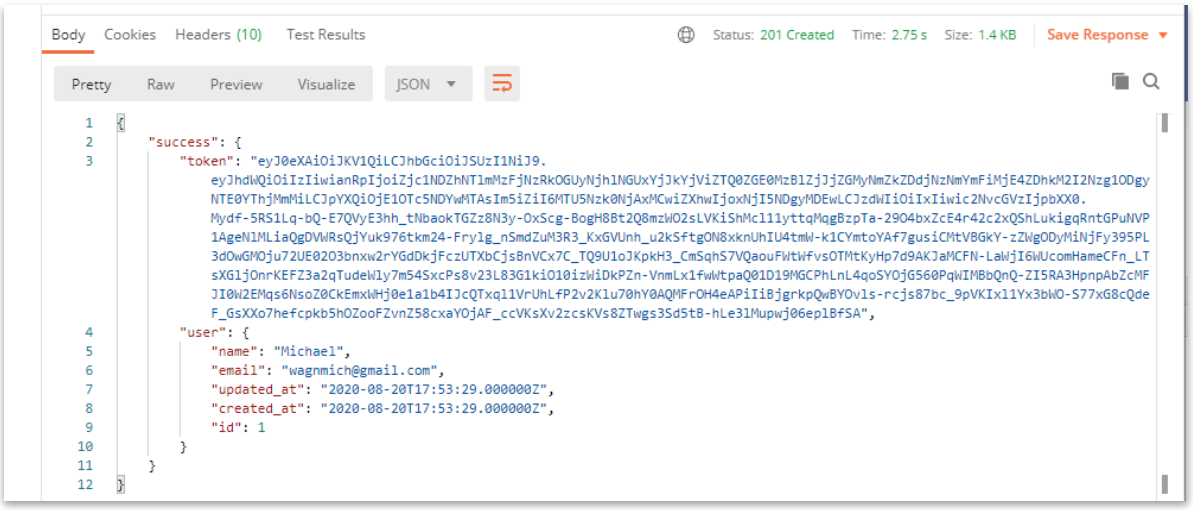
> *„With an API, there are a few things to test, but the most basic idea is, ‚when I request this URL, I want to see a foo resource', and ‚when I throw this JSON at the API, it should a) accept it or b) freak out.'" (Sturgeon and Bohill, 2015, p. 49)*

Um die Routen zu testen verwenden wir das Tool ***„Postman"*** mit dem man unkompliziert verschiedene Aufrufe mit unterschiedlichen http Verben an einer beliebigen API-Schnittstelle durchführen kann. Die Oberfläche des Tools beherbergt im oberen Bereich die **HTTP Methode** sowie die **Adresse**, auf welche der API Call abgesetzt wird, mittig einen Bereich für zu übergebende **Parameter** und an der Unterseite eine Ausgabe für den Response, den die API zurücksendet.

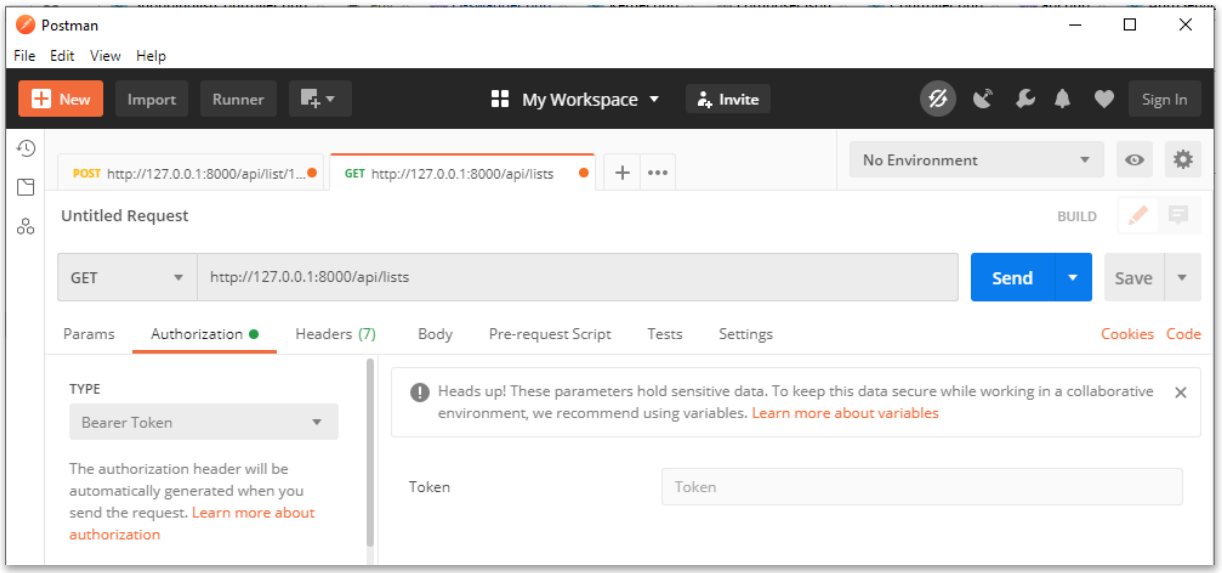
Folgende Funktionstests auf unsere API-Endpunkte geben uns Aufschluss darüber, dass die Schnittstelle funktioniert und für den weiteren Gebrauch in unseren Einkaufslisten-Apps verwendet werden können.

Als ersten Test der Routes müssen wir einen User registrieren. Dafür setzt man einen POST-Call an die entsprechende Adresse **„/register"** ab:

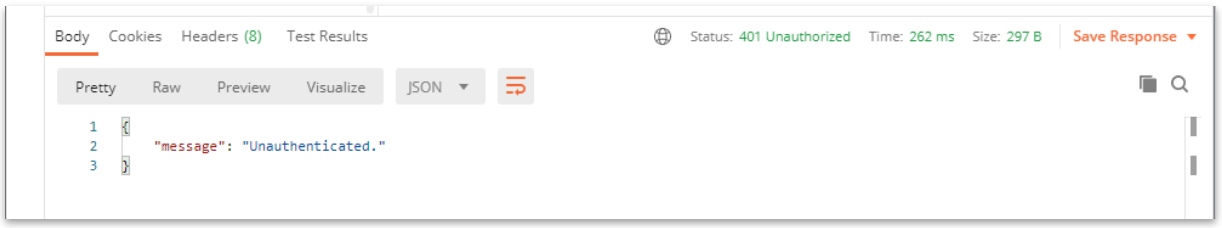




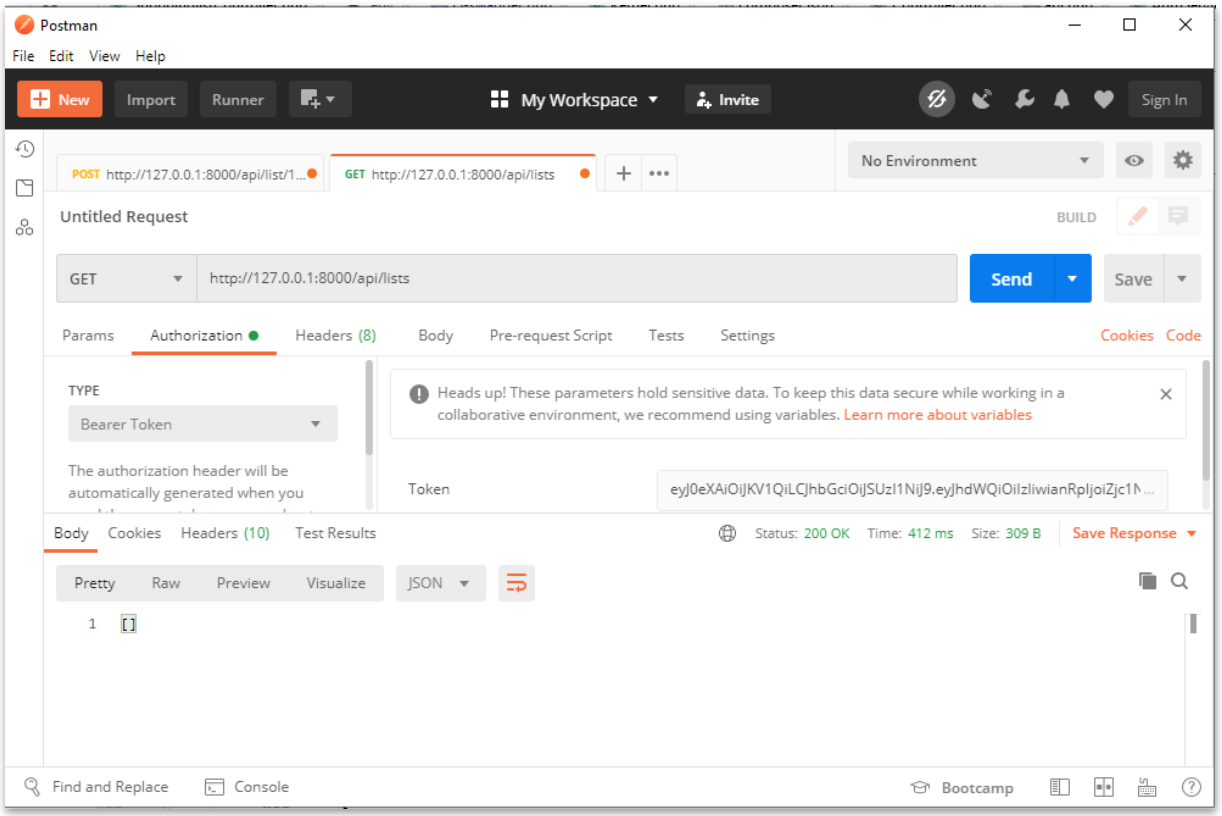
Wir erhalten von der API eine JSON formatierte Antwort, welche uns mittels des **Status „201 Created“** darüber informiert, dass der neue User erfolgreich in der Datenbank angelegt wurde. Außerdem befinden sich im Antwort-JSON nicht nur die Informationen über den neu angelegten User, sondern auch ein **oAuth Access-Token**. Dieser sogenannte **„Bearer Token“** ermöglicht, fortan im Request als Authentication-Header, den Zugriff auf die weiteren Endpunkt-Kontakte der Schnittstelle, die eine Authentifizierung erfordern. Solange, bis der Token ausläuft oder durch einen Logout-Vorgang ungültig gemacht wird.



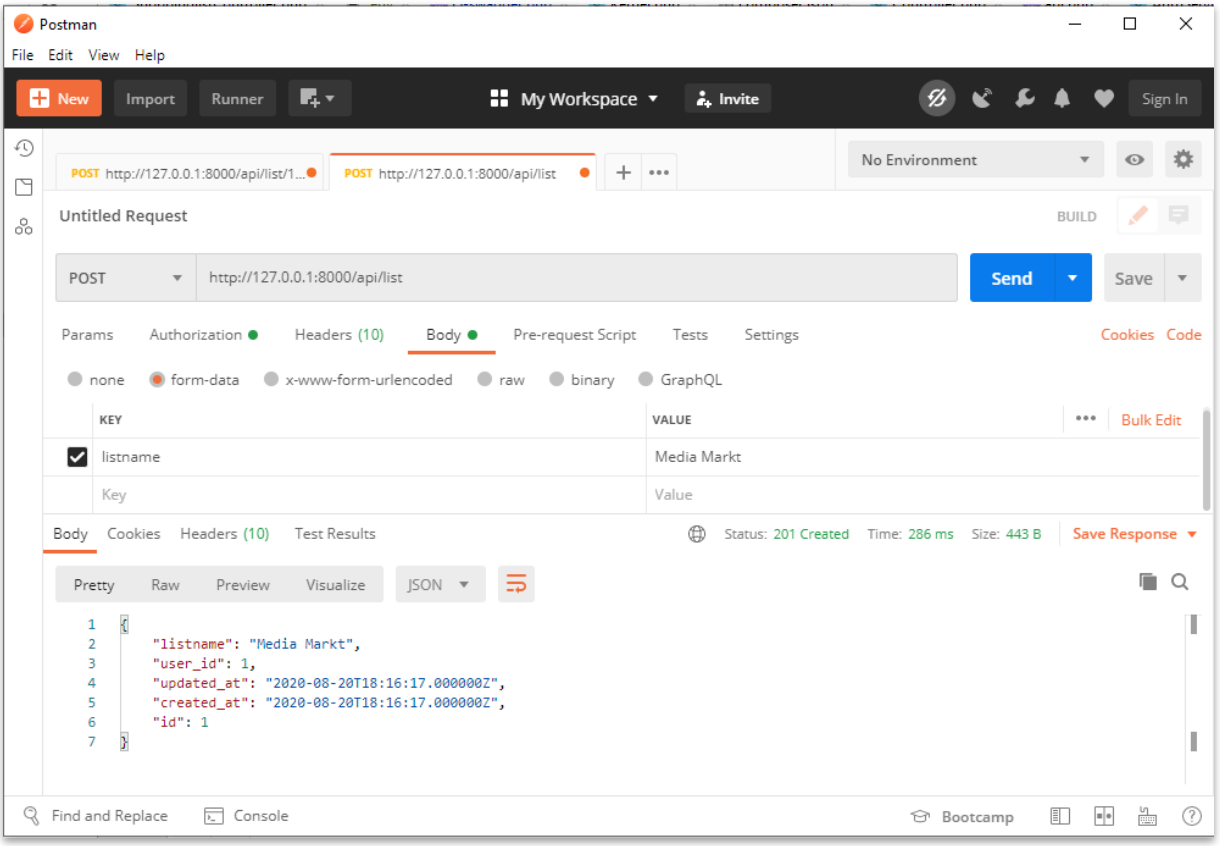
Hier sieht man den Zugriff auf den API-Endpunkt **„/lists“**, welcher hinter der Authentifzierung geschützt ist. Den Bearer Auth Token übergeben wir **nicht**, was zum Ergebnis **„401 Unauthorized“** in der Antwort führt.



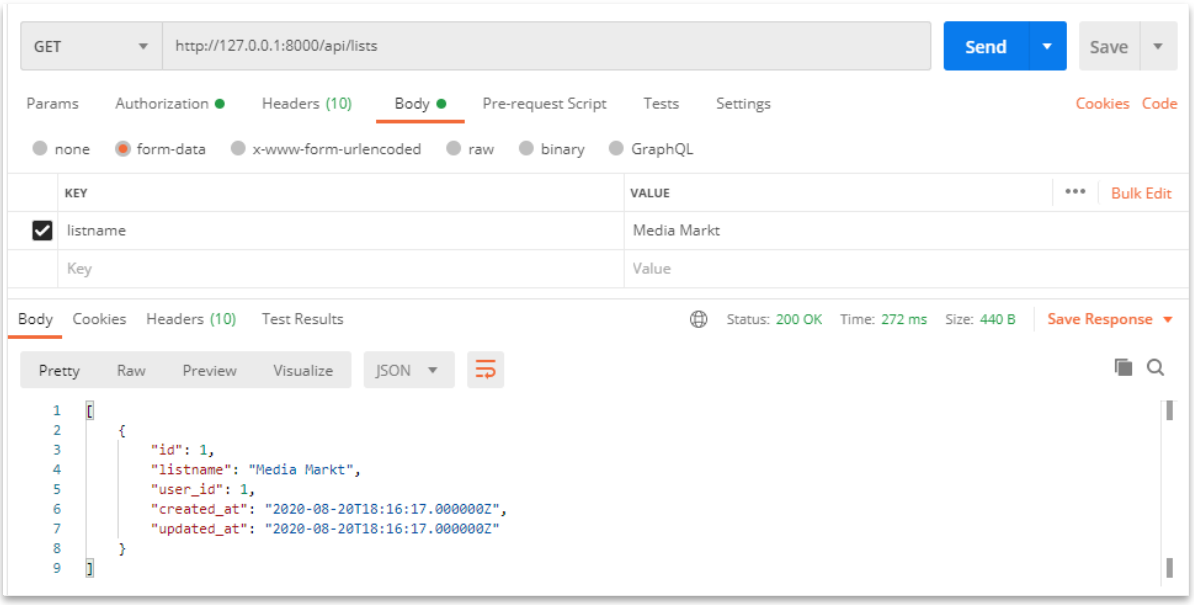
Kopiert man allerdings den Token in das dafür vorgesehene Token-Feld im Bereich **Authorization** von Postman, wird im Request der entsprechende Header mitgesendet und authentifiziert den Aufruf am Server. Die Antwort ist jetzt **„200 OK“** mit einem leeren JSON-Array [] im Body, da noch keine Listen angelegt wurden.



Mittels eines weiteren POST-Calls an den Endpunkt **„/list“** erzeugen wir eine neue Liste mit dem Namen *„Media Markt“* auf dem Server:



Die Antwort ist **„201 Created“** mit dem JSON der neu angelegten Liste im Response Body. Ein neuerliches GET Abrufen des **„/lists“** Endpunktes zeigt uns, dass der neue Eintrag erfolgreich in der Datenbank gespeichert wurde und ordnungsgemäß ausgeliefert wird:



Entsprechend dieser Vorgehensweise testen wir alle angelegten Endpunkte und vergewissern uns, dass die durch Swagger dokumentierten Response-Codes und die Inhalte korrekt sind.

4.4.14 CORS Headers (Cross Origin Resource Sharing)

Oft ist es notwendig, dass auf online verfügbare Ressourcen nur innerhalb derselben Quelle oder Domain zugegriffen werden darf. Zugriffe von außerhalb werden deswegen standardmäßig von Browsern mit der ***„same origin policy"*** unterbunden, was bedeutet Anfragen an fremde Quellen werden abgelehnt.

Eine Lösung für dieses Problem stellen **CORS Headers** dar, die beim Anfragen an die API mitgesendet werden müssen, um allen Beteiligten an dieser Kommunikation klar zu machen, dass es sich um eine geplante und sichere Aktion handelt.

Da unsere geplanten Apps aus verschiedenen Quellen (engl. *Origins*) auf die Ressource zugreifen werden, müssen wir sicherstellen, dass die Schnittstelle diese Art des Zugriffs möglich macht und dass diese Header gelesen und verstanden werden.

Bei den HTTP-Methoden **Post** und **Delete** wird zusätzlich ein Request an den Server gesendet, um zu überprüfen ob der Request sicher ist. Dieser Call wird ***CORS Preflight*** genannt und findet über die HTTP Methode **Options** statt.

Laravel integriert in der aktuellen Version das Composer Paket **„fruitcake/cors-headers"** und integriert diese Middleware bereits automatisch bei jedem Aufruf. In früheren Versionen (< 7.0) musste die CORS Middleware manuell in der Datei „app/Http/Kernel.php" registriert werden.

4.4.15 Deployment der REST-API

Nachdem die lokale Entwicklung und ein kurzer Funktionstest nun abgeschlossen sind, wird die Applikation auf einen entsprechenden PHP (Version 7.4) Server hochgeladen und wird für unsere Frontends fortan unter der Url

https://me-want-that.com/api/

erreichbar sein.

5 Übersicht – Welche Möglichkeiten eine App zu entwickeln gibt es?

> *If you ask ten different developers how they develop their mobile apps for Android and iOS devices, you'll probably get ten different answers. (Zammetti, 2019, p. 1)*

Dieses Zitat steht auf der ersten Seite des Buches *„Practical Flutter"* von Frank Zametti und könnte nicht wahrer sein. Die technologische Vielfalt, eine Anwendung für verschiedene mobile Endgeräte herzustellen, ist überwältigend. Das ist auch ein Grund, warum viele Entwickler mit der Wahl des entsprechenden Frameworks, oder mit der Wahl der passenden Methodik überfordert sind.

Vielleicht ist die Frage nach dem *„ease of use"* und den Vorlieben des Entwicklers aber auch die falsche Frage. Vielmehr sollten wir uns von Projekt zu Projekt die Frage stellen, welches Tool, welcher Weg des App Developments für dieses Projekt der Richtige ist. Die Wahl des richtigen Werkzeuges hängt auch sehr stark von der **Problemstellung**, der **Erwartungshaltung** an das Endergebnis und nicht zuletzt dem **Budget** ab.

> *App-Designer müssen die technologischen Aspekte, Mechanismen und Möglichkeiten kennen, unter denen Apps entwickelt werden können. Designer möchten nach Möglichkeit keine Grenzen auferlegt bekommen, aber in der Softwareentwicklung lassen sich gewisse technische Rahmenbedingungen nicht umgehen.*
>
> *(Semler and Tschierschke, 2019, p. 45)*

Deshalb ist es von Vorteil, wenn man die technischen Möglichkeiten im Überblick samt ihren Vorzügen und Limitierungen kennt. Erst dann ist eine gute Entscheidungsgrundlage für das richtige Tool und den richtigen Weg gegeben.

Frank Zametti bringt es in der Einleitung in seinem Buch *„Practical React Native“* auf den Punkt:

> *Creating mobile apps that look, feel, and function like native apps and are also cross-platform is a difficult proposition, even after all these years of developers working to achieve this. You can write native code for each platform and do your best to make them as similar as possible, and thats certainly a good way to get native performance and capabilities into your app, but essentially, that means writing your app multiple times. Instead you can take the HTML route and have a single code base that works everywhere, but you will often be left out in the cold, in terms of native device capabilities, not to mention performance frequently being subpar.*
>
> *(Zammetti, 2018, chap. Introduction)*

Folgende Varianten stehen zur Verfügung, jede dieser Varianten bietet in sich wieder viele verschiedene Tools, Frameworks und Libraries an:

5.1 100% native Entwicklung

- Android SDK / Android Studio
- Apple xCode

5.2 Progressive Web App / HTML5 App

Werden im mobilen Browser ausgeführt.

- VueJS + Vuetify
- jQuery Mobile

5.3 Progressive Web App / Hybrid App

Unsichtbar in einen Browser Frame verpackt, Zugriff auf die Betriebssystem APIs über eine Zwischenschicht.

> *PWAs [Anm. Progressive Web Apps] oder Hybrid Apps sind im Grunde Webseiten mit angereicherten nativen Funktionen.*
>
> *(Semler and Tschierschke, 2019, p. 24)*

> *Hybrid Apps sind eine Mischung aus Nativen- oder Webkomponenten. Diese Hybrid-Apps können in den jeweiligen App Stores bereitgestellt und angeboten werden.*
>
> *(Semler and Tschierschke, 2019, p. 25)*

> *Der native Teil der App läuft damit auf dem Betriebssystem und stellt eine HTML-Rendering-Engine bereit, die Browserfunktionalität bietet und darüber hinaus eine Brücke zum Betriebssystem darstellt. So läuft der Webquellcode in seiner Umgebung, gleichzeitig kann aber auch auf die APIs und Features des Betriebssystems zugegriffen werden.*
>
> *(Semler and Tschierschke, 2019, p. 51)*

> *Progressive Web Apps lassen Webanwendungen in einem nativen Gewand erscheinen. Sie basieren auf dem Web-Stack mit den Sprachen HTML, CSS und JavaScript. Darauf aufbauende Sprachen wie Typescript vereinfachen die Entwicklung.*
>
> *(Liebel, 2018, p. 65)*

- IONIC
- Sencha Touch

5.4 Native Hybrid App

Werden kompiliert und verwenden keine Browserschicht für die Darstellung, allerdings meistens auch keine nativen UI Elemente

- Flutter
- React Native
- Xamarin
- NativeScript

5.5 Game Engines / App Engines

Verpacken eine eigene Runtime für die Rendering-Loop und die Darstellung des UI ins App Bundle mit

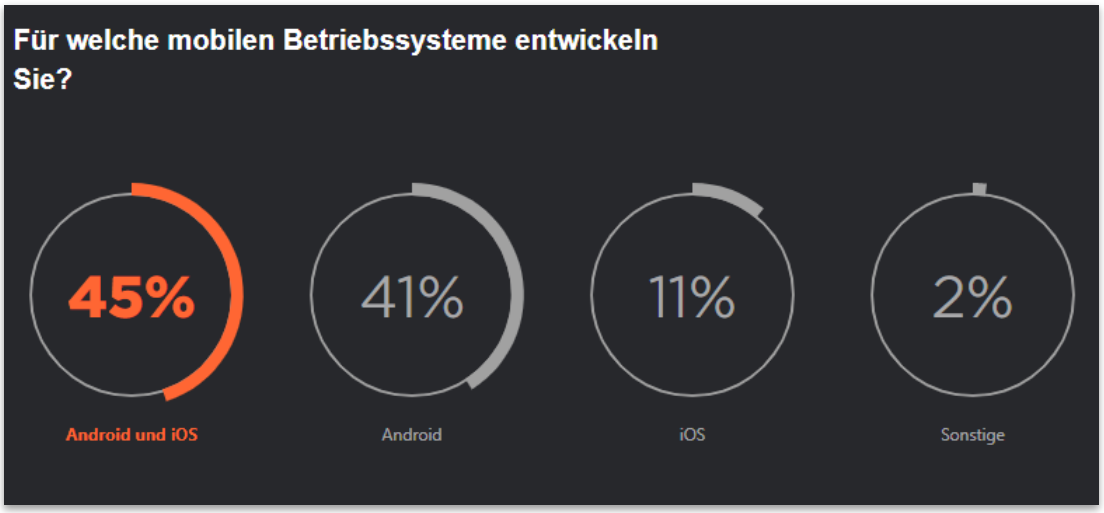
- Unity
- libGDX
- Unreal Engine
- Corona Labs

6 Traditionelles Native App Development

In der *JetBrains Development Survey*, einem seit 2016 jährlich erscheinenden Bericht über das Entwickler Ökosystem, findet man interessante Zahlen und Aussagen über die Verteilung von Developern auf die nativen Tools und, ihnen gegenübergestellt, die Cross Plattform Tools:

> *Native Tools werden bei der Mobile-Entwicklung weiterhin bevorzugt: Zwei Drittel der Mobile-Entwickler nutzen sie. Daneben verwendet die Hälfte der Mobile-Entwickler plattformübergreifende Technologien oder Frameworks. Unter diesen Frameworks ist React Native immer noch am populärsten – es wird von 42% der Mobile-Entwickler verwendet.*
>
> *("Jetbrains Developer Survey 2020," 2020)*

Das bedeutet, dass dem Entwickeln von *„echten"* nativen Apps nach wie vor eine sehr bedeutsame Rolle zukommt und viele Entwickler sich aufgrund von Performance und nativem Feeling der App dazu entscheiden, die traditionellen Wege zu beschreiten. So wie Apple und Google ursprünglich vorgesehen hatten.



Quelle: JetBrains Developer Survey 2020

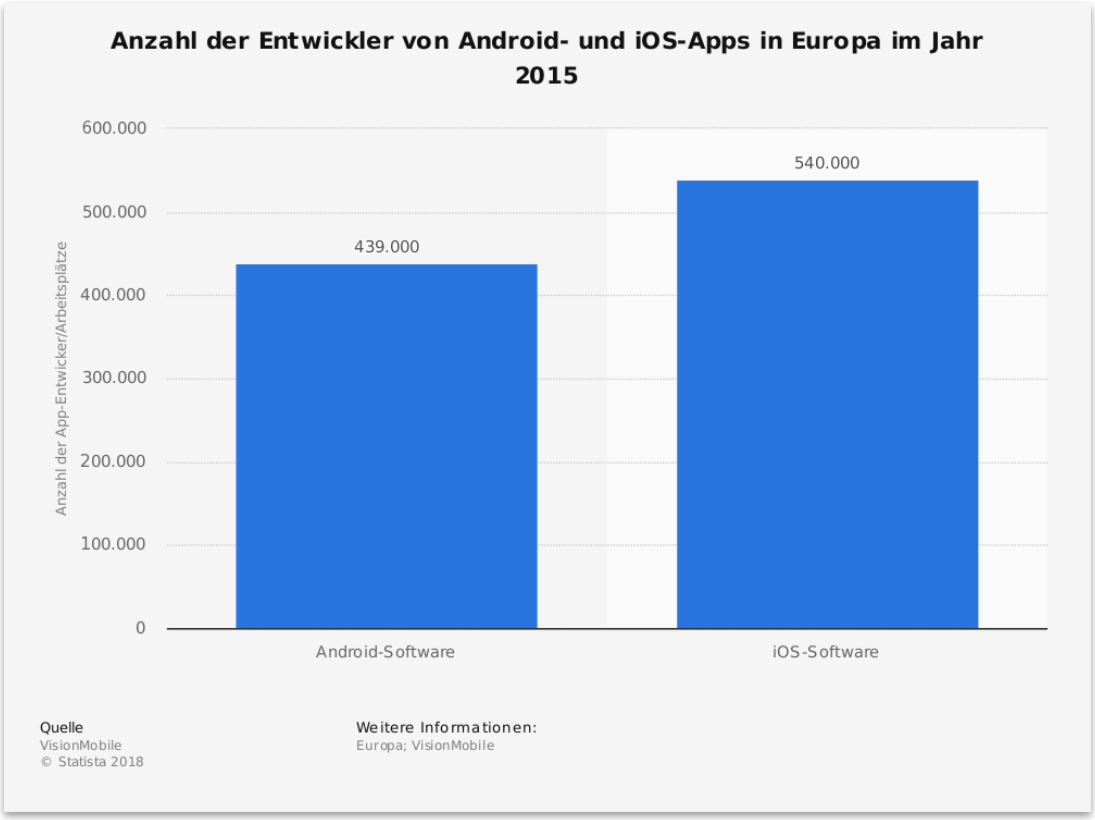
Semler und Tschierschke beschreiben die Native App-Entwicklung mit folgenden Worten:

> *Native Apps sind „echte" Programme, die direkt im Betriebssystem laufen. Hinter einer nativen App steckt ein binärer Code, der mit dem Betriebssystem interagiert. Es besteht hierbei Zugriff auf alle APIs, die das jeweilige System zur Verfügung stellt.*
>
> *(Semler and Tschierschke, 2019, p. 46)*

> *Eine native App wird in Entwickler-Quellcode geschrieben. Zusätzlich werden Assets mitgeliefert und beides auf dem Betriebssystem abgelegt. Entwickelt wird bei iOS in Xcode, bei Android in Android SDK oder NDK. Jeder Anbieter von Betriebssystemen stellt hierfür in der Regel Entwicklungsumgebungen für Entwickler bereit.*
>
> *(Semler and Tschierschke, 2019, p. 46)*

> *Ein großer Nachteil der nativen Entwicklung ist, dass der einmal geschriebene Quellcode eines spezifischen Betriebssystems nicht ohne Weiteres in eine andere Entwicklungsumgebung adaptiert werden kann. Hier steigt in dem Fall der Entwicklungsaufwand.*
>
> *(Semler and Tschierschke, 2019, p. 46)*

Ein interessanter Vergleich ist die Anzahl der Entwickler für native iOS Apps mit der Anzahl der Entwickler für native Android Apps. Hier waren im Jahr 2015 mehr iOS Entwickler in Europa verzeichnet:



Quelle: VisionMobile, Statista Austria 2018

6.1 Android

> *Es handelt sich um ein Open-Source-Projekt von Open-Handset Alliance (gegründet von Google) und wird quelloffen entwickelt. Von Januar 2018 bis Januar 2019 hatte Android einen weltweiten Marktanteil von 74,45% (zum Nachlesen: https://computerwelt.at/news/iphone-vs-android-wer-hat-die-groesseren-marktanteile/).*
>
> *(Semler and Tschierschke, 2019, p. 40)*

Semler und Tschierschke heben in ihrem Buch „App Design“ positiv hervor, dass ein natives App-Development nahezu unbegrenzte Gestaltungsmöglichkeiten bietet, optimale Ausnutzung von Hardwarekomponenten stattfindet und die so entwickelten Apps eine hohe Performance durch Zugriff auf hardwarebeschleunigte Grafik etc. unterstützen. Sie halten dagegen, dass der Entwicklungsaufwand und die Kosten entsprechend hoch ausfallen, hoher Pflegeaufwand der App durch Updates und Features besteht und eine Anpassung an Multiplattformen nur sehr schwer bzw unmöglich ausfällt. (Semler and Tschierschke, 2019, p. 48)

> *Der Android-Gerätemarkt weist eine hohe Diversität an Android-Versionen auf, die es Entwicklern sehr schwer macht, Applikationen zu entwickeln, die auf allen Versionen lauffähig sind. Zwar ist der grundsätzliche Aufbau von Android meist überall gleich, doch die visuelle Erscheinung der Oberfläche variiert oft sehr stark.*
>
> *(Semler and Tschierschke, 2019, p. 41)*

Für die Entwicklung selbst benötigt man das Android SDK, welches in der IDE Android Studio bereits integriert ist. Dort enthalten sind die jeweiligen Plattform Tools, welche eine konkrete Implementierung für eine der zahlreichen Android Versionen ermöglicht. Programmiert wird in der bewährten, den meisten Entwicklern bekannten und gut dokumentierten Programmiersprache **Java**.

Ein Nachteil der Entwicklung für Android ist die große **Fragmentierung** des Device Marktes. Die Endgeräte sind unterschiedlicher als sie schlimmer nicht sein könnten. Low Budget Geräte mit niedriger Auflösung laufen im Extremfall dieselben Apps wie die Flagschiff-Devices der Hersteller.

> *Viele Hersteller setzen auf Gerätevielfalt, um eine möglichst breite Käufergruppe abzudecken. So gibt es unterschiedliche Bildschirmgrößen, Farben und Materialien in allen möglichen Preisklassen, die jeweils auf die bestimmte Zielgruppe zugeschnitten sind. Für das herstellende Unternehmen kann dies ein Vorteil sein, für Entwickler und Designer von Applikationen stellt es jedoch ein großes Problem dar.*
>
> *(Semler and Tschierschke, 2019, p. 43)*

6.2 iOS

Die Entwicklung für iOS erfordert zuallererst einen Mac. Ohne dieses Development Gerät lässt sich kein iOS Projekt bewerkstelligen, benötigt man doch zwingend die Entwicklungsumgebung **xCode**, welche nur über den Mac App Store installiert werden kann.

Entwickelt werden iOS Apps in der Sprache **Objective-C**, mit der die Community sehr zufrieden ist und die sich mit ihrer C-ähnlichen Syntax gut beherrschen lässt.

> *Apples iOS ist – anders als das Anroid-Betriebssystem – nicht lizenzierbar, sondern nur auf Apples eigener Hardware lauffähig. Von Januar 2018 bis Januar 2019 hatte iOS einen weltweiten Marktanteil von 22,85%.*

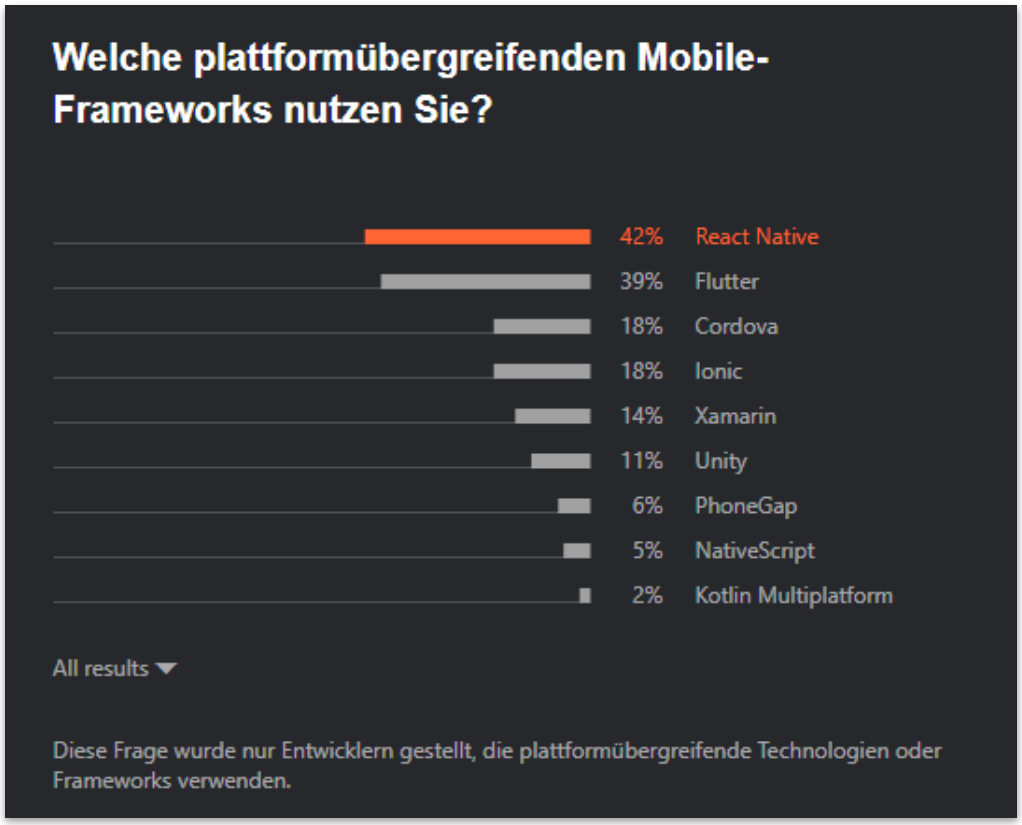
7 Non Traditionelles App Development / Cross Plattform Development

„Nicht traditionell“ bedeutet in diesem Kontext, anders als von den großen OS Herstellern ursprünglich vorgesehen, also nicht *„nativ“*. Damit sind die verschiedenen Varianten gemeint, welche eine einzige Code-Base auf beiden Plattformen exportieren können und somit *„Cross Plattform“* also auf beiden Plattformen laufen. Dabei ist es unerheblich, ob in diesem Zusammenhang von einer **PWA**, **Hybrid-App** oder einer **Nativen Hybrid-App** gesprochen wird.

Jan Semmler und Kira Tschierschke beschreiben hier in einem kurzen Absatz sehr gut das größte Argument für die Verwendung von Cross Plattform Frameworks. So wie auch *React Native*, ist dies auch mit vielzähligen anderen Frameworks und Libraries möglich. Eine davon ist ***Flutter***, welche wir uns in dieser Arbeit genauer ansehen werden.

> *Wie React Native ist Flutter auch eine Cross-Entwicklungsplattform, die das Ziel hat, beide Plattformen iOS und Android zu bedienen.*
>
> *(Semler and Tschierschke, 2019, p. 25)*

Betrachtet man die 2020er Entwickler-Survey von JetBrains zu diesem Thema, stellt man fest, dass Flutter, obwohl sehr neu, bereits dicht auf den Fersen des Platzhirsches *React Native* ist:



Quelle: JetBrains Developer Survey 2020

Flutter ist im Vergleich zum Vorjahr populärer geworden – sein Anteil hat um 9 Prozentpunkte zugenommen. Im selben Zeitraum verloren Cordova, Ionic und Xamarin jeweils rund 10 Prozentpunkte.

("Jetbrains Developer Survey 2020," 2020)

Für die nähere Betrachtung und exemplarische Implementierung habe ich mich für ***Flutter*** als Beispiel für die Native Hybrid App Entwicklung entschieden. Grund dafür ist der starke Zuwachs an Entwicklern im letzten Jahr. Weiters wird ***Ionic*** als Beispiel für eine Nicht Native Hybrid App Entwicklung sein, da es der modernere Ansatz neben HTML5 Apache Cordova ist und ein runderes Gesamtpaket anbietet.

7.1 Frontend Development für Einkaufsliste mit Cross Platform Development Tools

7.1.1 Flutter

> *The original stated goal of Flutter, or at least one of the main ones, was being able to render app UIs at a consistent 120fps no matter what. Google very much understands that a consistently smooth UI is one that users will be delighted by and want to use, so it was at the forefront of their thinking with Flutter.*
>
> *(Zammetti, 2019, p. 4)*

7.1.1.1 Entwicklungsparadigmen

Eine der vielleicht größten Hürden für die meisten Entwickler sich mit Flutter auseinander zu setzen, ist das Erlernen der dafür notwendigen, relativ neuen Programmiersprache „Dart". Dart wurde 2015 für die neue Google Plattform Fuchsia entworfen und wird seit 2017 auch für Flutter eingesetzt. Dart ist eine Hochsprache der 3. Generation und wird „ahead of time" kompiliert, um die Performance während der Laufzeit zu optimieren. Obwohl Dart eine neue Sprache ist, findet man sich in der vertrauten C-Syntax schnell zurecht. Nach kurzer Einarbeitungszeit mit der guten Dokumentation oder einem Online-Tutorial steht dem produktiven Arbeiten mit Flutter und Dart nichts mehr im Weg.

> *Dart is simple and powerful, object-oriented, and strongly typed, allowing developers to be very productive very quickly and to do so with safety. Once you get over the (not usually large) initial learning curve, most developers tend to like Dart as compared to languages like JavaScript, Objective-C, or Java.*
>
> *(Zammetti, 2019, p. 15)*

Nachdem man sich das **Flutter SDK** installiert und ein geeignetes Werkzeug (ich empfehle Android Studio) ausgewählt hat, sollte man sich erstmal mit den grundlegenden Funktionsweisen von Flutter vertraut machen:

- Jedes Element in Flutter, vom kleinsten Button bis zur gesamten Applikation ist ein sogenanntes ***„Widget“***. Diese Widgets können entweder *stateless* oder *statefull* sein. Das bedeutet, wenn innerhalb des Widgets ein Zustand gespeichert wird, wie zB der „pressed“ state eines Buttons, ist das Widget ein „statefull“ Widget. Widgets können ineinander verschachtelt werden und durch verschiedene Layouts ausgerichtet.
- Der **Widget-Tree** kann bei größeren Apps durchaus tief verschachtelt und kompliziert werden.
- Flutter besteht konzeptionell aus 4 Hauptbereichen (Zammetti, 2019, p. 4)
    - Die **Dart Plattform**

      Die Sprachumgebung in welcher die tatsächliche Applikation geschrieben wird. Die App-Logik wird „ahead of time“ kompiliert und als Lib von der Main Flutter Engine zur Laufzeit ausgeführt und gerendert
    - Die **„Main Flutter Engine“**

      Eine hauptsächlich auf C++ basierende Codebasis welche auf mobilen Endgeräten fast mit nativer Geschwindigkeit ausgeführt werden kann. Sie benützt intern die Skia-Graphics Engine, um Komponenten zu rendern
    - Die **Foundation Library**

      Ein Interface über die nativen SDKs der zugrundeliegenden OSs
    - Die **UI Widget Library**

      Eine Komponentenbibliothek für alle UI Elemente die das Herz begehrt

7.1.1.2 Android Package Kit (APK)

Die Android-Version dieser Flutter/Dart App kann downgeloadet und ausprobiert werden. Zusätzlich liegt sie dem Speichermedium dieser Arbeit bei. Zum Downloaden, einfach den QR Link scannen oder dem Link folgen.

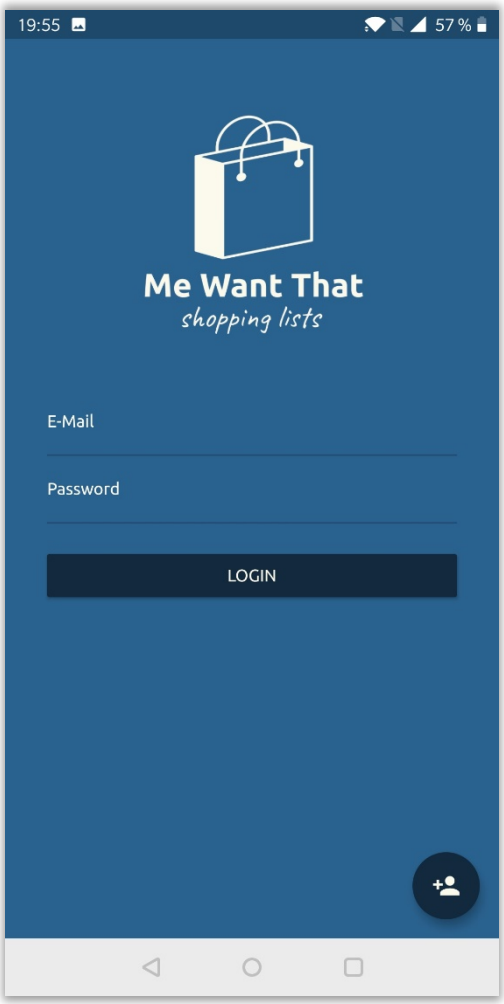
https://me-want-that.com/downloads/MeWantThatFlutter.apk

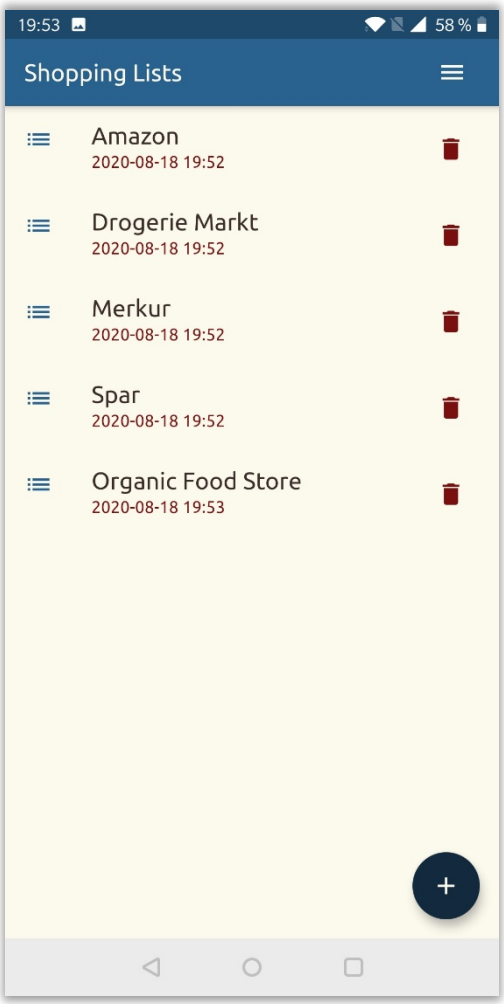
7.1.1.3 Github Repository

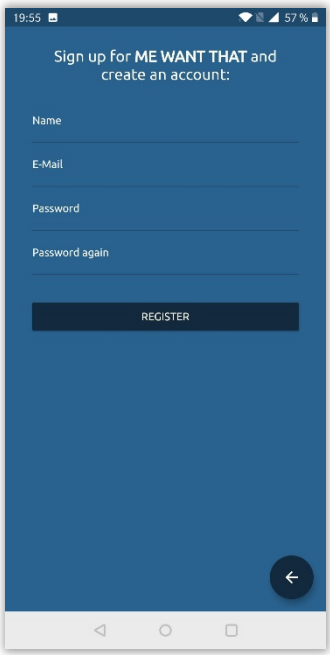
Der gesamte Sourcecode ist öffentlich unter folgendem GitHub Repository verfügbar:

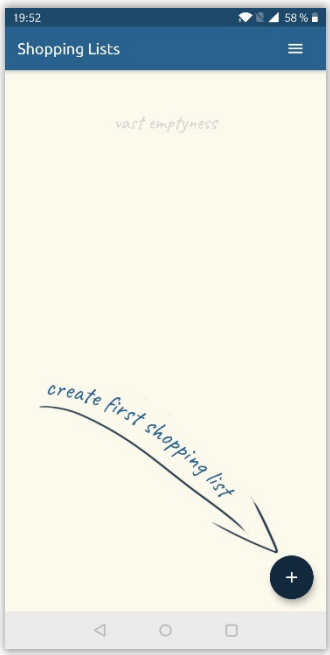
https://github.com/MikesDevCorner/MeWantThat-frontend-flutter.git

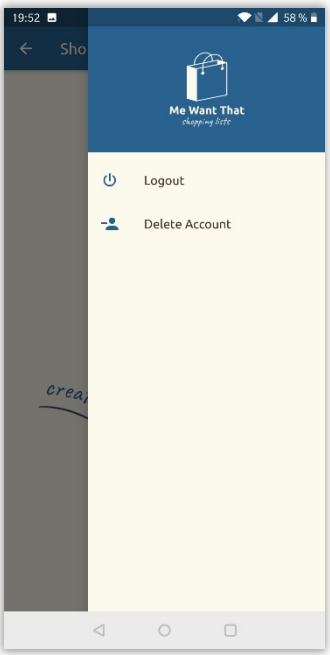
7.1.1.4 Screenshots der Flutter App

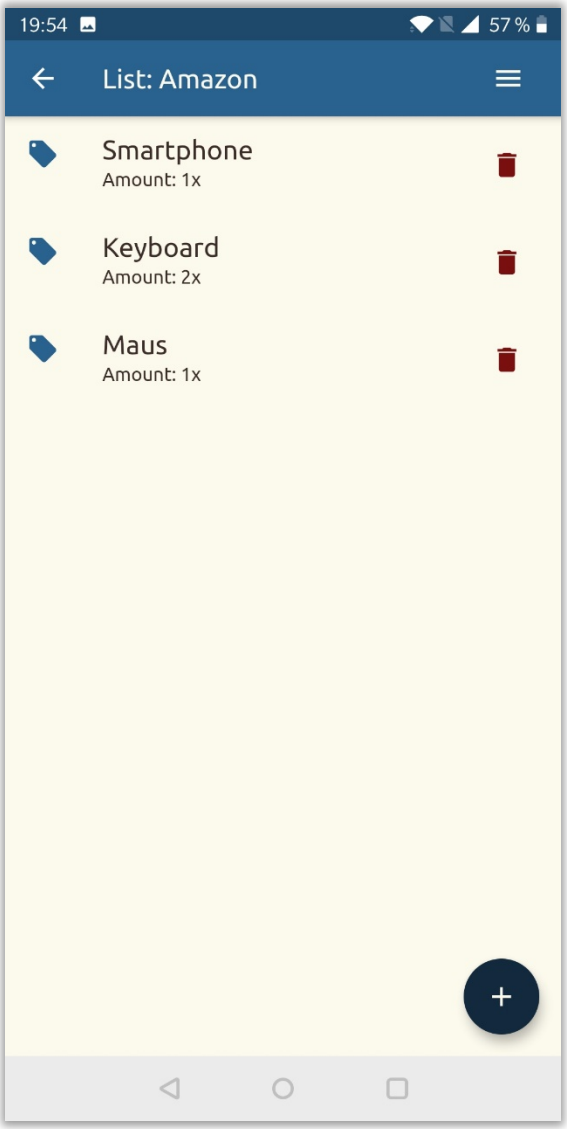


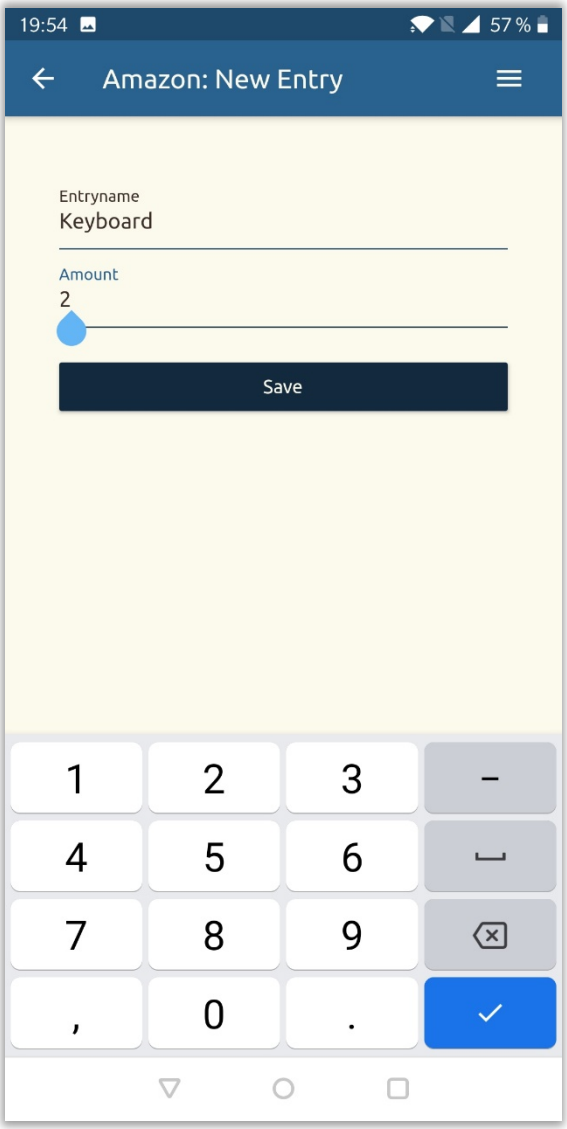









19:54
57 %
List: Amazon
Smartphone
Amount: 1x
Keyboard
Amount: 2x
Maus
Amount: 1x


19:54
57 %
Amazon: New Entry
Entryname
Keyboard
Amount
2
Save
1
2
3
4
5
6
7
8
9
,
0
.

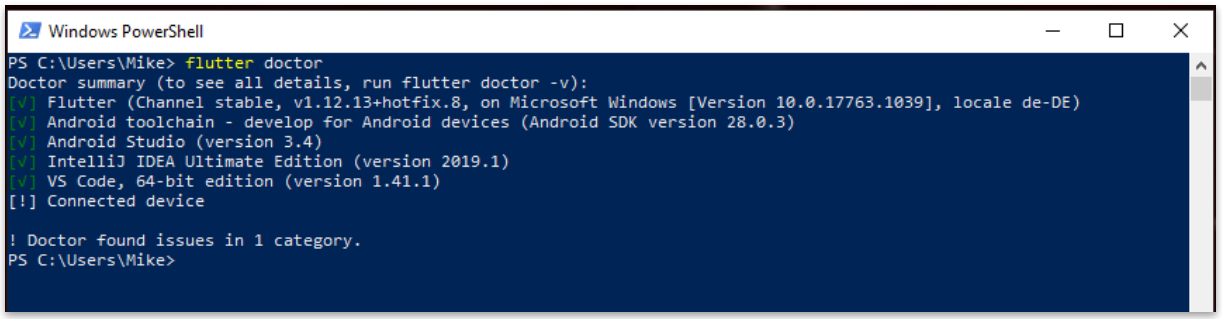
7.1.1.5 Command Line Tools

Flutter wird natürlich auch mit einem Satz an hilfreichen Commandline Tools ausgeliefert, und sobald man die aktuellste Version von Flutter für sein Betriebssystem installiert hat und die entsprechenden Binaries (Subdirectory /bin) in seine ***Path*** Systemumgebungsvariablen aufgenommen hat, können diese global am System mit dem Keyword „flutter“ benützt werden.

Ein wichtiger Command direkt nach der Installation ist:

```
$ flutter doctor
```

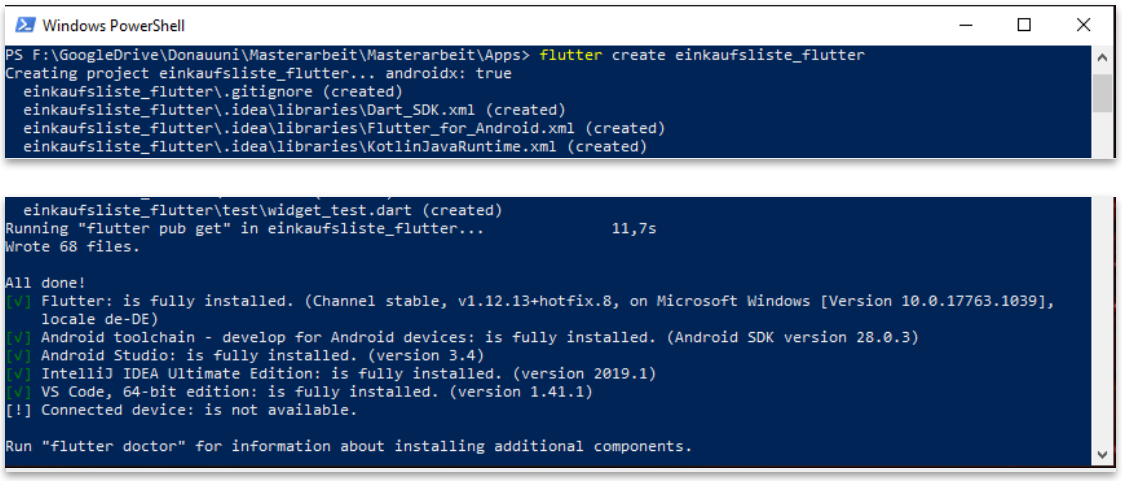
Dieser Command prüft, ob alle Prerequisites für die Entwicklung im Flutter Ecosystem auf dem Entwicklerrechner installiert sind und gibt etwaige Konfigurationsprobleme, fehlende Libraries (zB Android) oder fehlende Plugins in den installierten Entwicklungsumgebungen aus. Eine korrekte Installation aller Tools hat folgende Ausgabe des Doktors zur Folge:



Ein weiterer wichtiger Command der benötigt wird ist:

```
$ flutter create einkaufsliste_flutter
```

Wie der Name bereits vermuten lässt, erstellen wir auf diese Art ein neues Flutter Projekt. Die Ausgabe im Erfolgsfall stellt sich wie folgt dar:



Wie bereits erwähnt, greift man bei der Entwicklung mit der Programmiersprache ***Dart*** auf ein Paketmanagementsystem namens ***PUB*** zu. Dieses Werkzeug umfasst unter anderem das Laden und Veröffentlichen von mit DART geschriebenen Softwarepaketen und Command Line Tools. In unserer Flutter-Welt deklarieren wir innerhalb des PUB Konfigurationsfiles (pubspec.yaml) die Paketabhängigkeiten unserer Anwendung. Analog zum (aus der Welt der Webentwicklung) weit bekannten Paketmanagementtool **NPM**, dem **N**ode **P**ackage **M**anager, gibt es auch für PUB ein paar wichtige Command Line Interface Commandos, die zum Beispiel die Abhängigkeiten in Form von Paketen in das Projekt downloaden und für unsere Dart-Files verfügbar machen. Der wichtigste Befehl hierfür ist vermutlich

```
flutter pub get
```

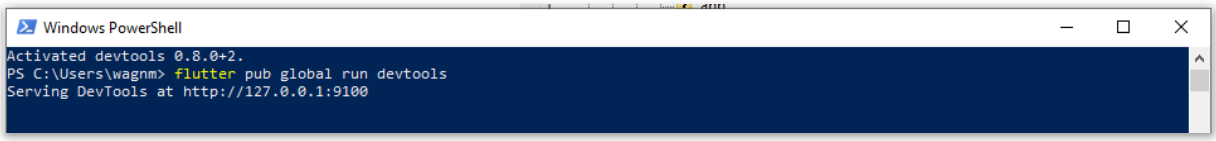
welcher die pubspec.yaml durchforstet und die Dart-Pakete ins Projekt lädt.

Flutter bringt auch ein Set an hervorragenden, browserbasierten **Dev-Tools** mit, welche die Entwicklung und das Debugging erleichtern sollen. Einmal aktiviert, müssen die DevTools nur mehr gestartet werden:

```
flutter pub global activate devtools
```

```
flutter pub global run devtools
```

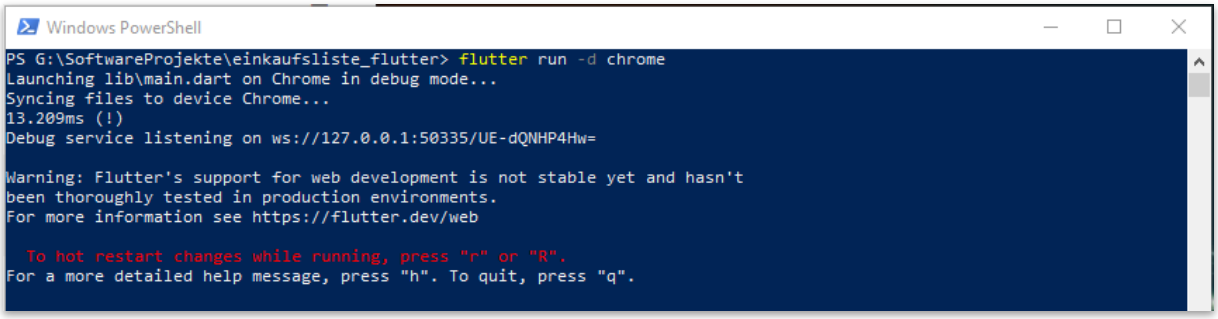
Nach dem Starten erhält man im Konsolenfenster einen Localhost-Link (127.0.0.1:9100), unter welchem die DevTools im Browser erreichbar sind:



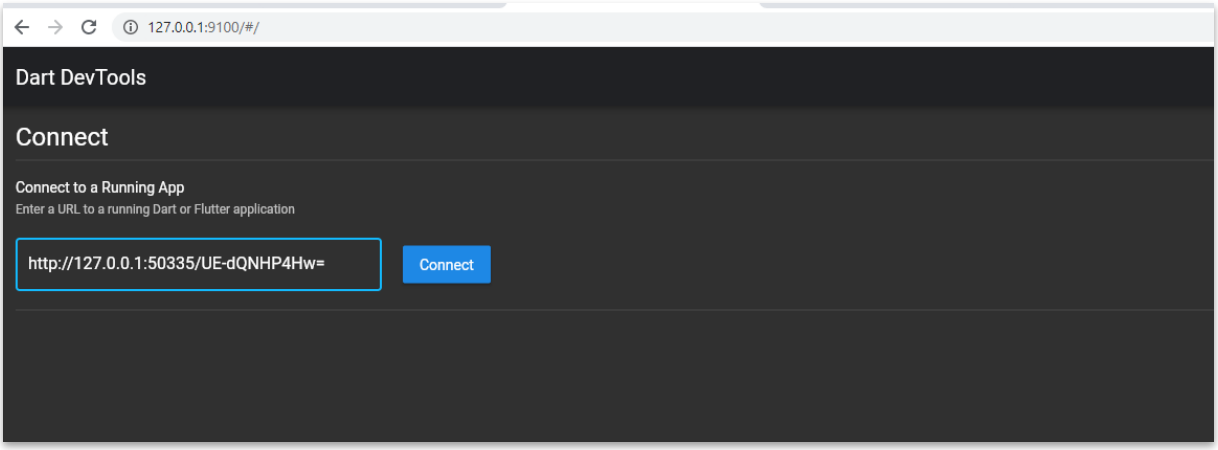
Startet man jetzt ein Flutter Projekt mit der Commandline über folgenden Befehl

```
flutter run -d chrome
```

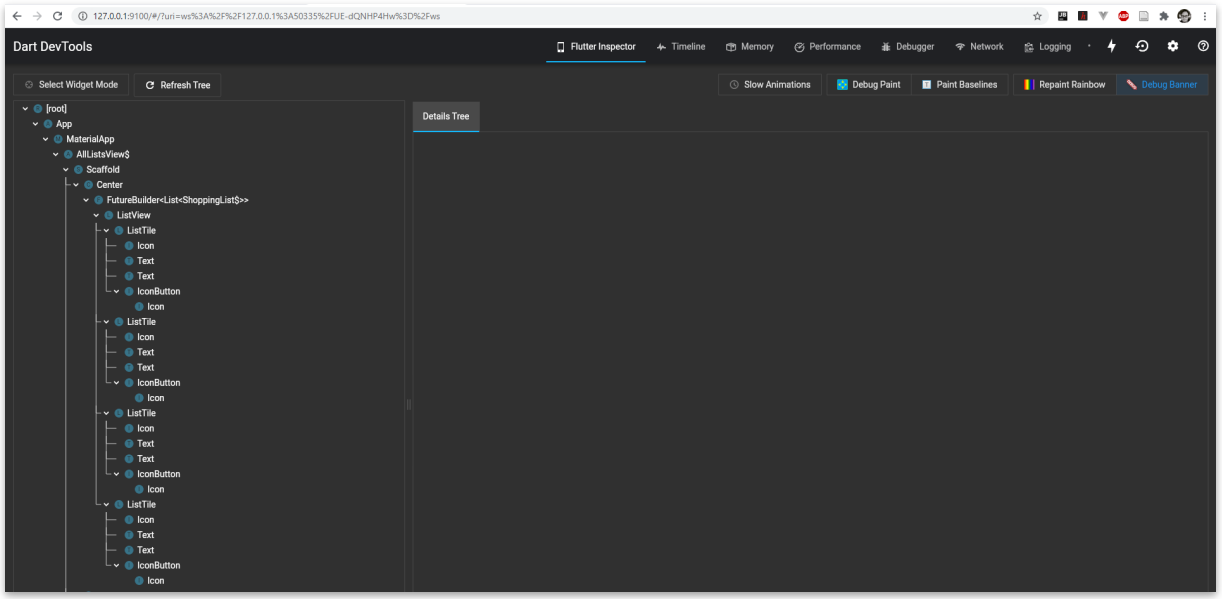
erhält man nicht nur eine weitere Localhost-Adresse mit Port (in unserem Beispiel 127.0.0.1:50335)

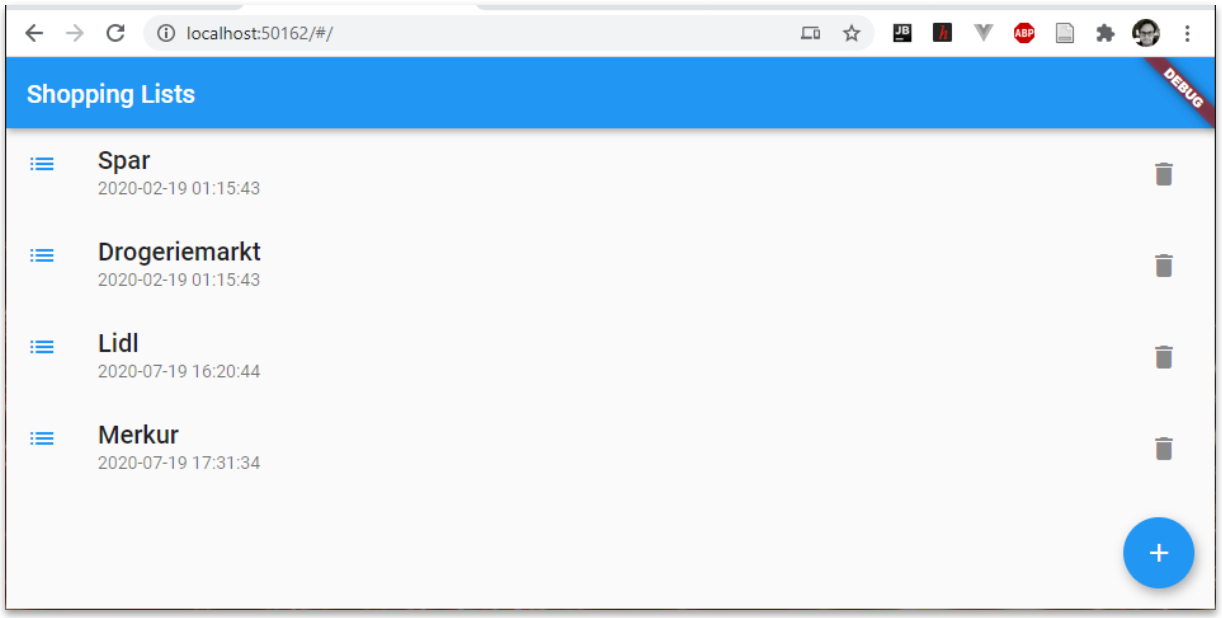


sondern auch die Möglichkeit, diese Adresse für die DevTools zu registrieren:

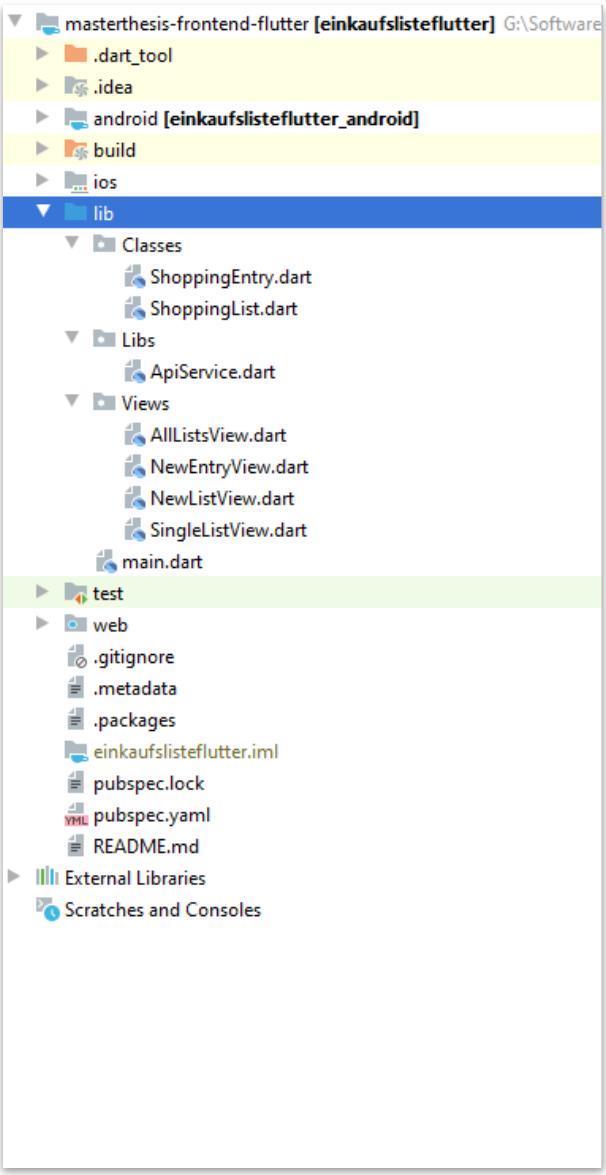


Die beiden Localhost-Adressen stellen die laufende (Web) App dar sowie die Flutter Development Tools zum Inspizieren des Component-Trees und weitere praktische Profiling Tätigkeiten im Entwicklungsprozess dar.





7.1.1.6 Ordnerstruktur



So sieht die Flutter Ordnerstruktur aus, nachdem man mit den Commandline Tools ein Projekt erzeugt hat.

Die Ordner „**android**“ und „**ios**“ beinhalten die plattformabhängigen Subbereiche des Projektes und müssen selten direkt angegriffen werden (der ios Ordner stellt das xCode Projekt für iOS Releases dar).

Der Ordner „**lib**“ beinhaltet unsere Dart-Files, die *tatsächliche* App-Entwicklung findet hier statt. Zuerst wirkt es möglicherweise etwas befremdlich, dass sich die gesamte Dart-Logik und somit eigentlich die ganze App im Ordner „lib“ befindet. Der Grund ist, dass beim Ahead-of-Time Compiling der Sourcecode in eine Library kompiliert wird und diese dann von der Flutter Engine am Endgerät ausgeführt wird.

Zu erwähnen ist noch das File **pubspec.yaml**, welches Paketabhängigkeiten und App-Configs beherbergt.

7.1.1.7 Implementierung

Die Klassen ShoppingEntry und ShoppingList stellen unsere Modellklassen dar und repräsentieren die Datensätze, die von der REST API abgerufen wird.

File: lib/Classes/ShoppingEntry.dart

```
class ShoppingEntry {

  final int id;
  final String postenname;
  final int anzahl;
  final int einkaufsliste_id;
  final String created_at;
  final String updated_at;

  ShoppingEntry({this.id, this.postenname, this.anzahl, this.einkaufsliste_id,
    this.created_at, this.updated_at});

  factory ShoppingEntry.fromJson(Map<String, dynamic> json) {
    return ShoppingEntry(
        id: json['id'],
        postenname: json['postenname'],
        anzahl: json['anzahl'],
        einkaufsliste_id: json['einkaufsliste_id'],
        created_at: json['created_at'],
        updated_at: json['updated_at']
    );
  }
}
```

File: lib/Classes/ShoppingList.dart

```
class ShoppingList {

  final int id;
  final String listenname;
  final String created_at;
  final String updated_at;

  ShoppingList({this.id, this.listenname, this.created_at, this.updated_at});

  factory ShoppingList.fromJson(Map<String, dynamic> json) {
    return ShoppingList(
        id: json['id'],
        listenname: json['listenname'],
        created_at: json['created_at'],
        updated_at: json['updated_at']
    );
  }
}
```

File: lib/Libs/ApiService.dart

ApiService stellt die Verbindung zum REST Service her und führt die asynchronen API-Calls aus.

```
import 'package:ShoppingList_Flutter/Libs/AuthService.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import '../Classes/ShoppingList.dart';
import '../Classes/ShoppingEntry.dart';
import './AuthService.dart';


class ApiService {

  static final String url = 'https://me-want-that.com/api';

  static Future<dynamic> login(String email, String password) async {
      final response = await http.post(ApiService.url + '/login',
          headers: AuthService.getHeaders(),
          body: jsonEncode(<String, String>{
            'email': email,
            'password': password
          })
      );
      if (response.statusCode == 202) {
        final Map parsed = json.decode(response.body);
        await AuthService.setToken(parsed['success']['token']);
        return true;
      } else {
        if (await AuthService.checkUnauthenticated(response))
          return false;
        else
          throw Exception("some error occured during login: ");
      }
  }


  static Future<dynamic> register(String email, String password, String passwordConfirm,
                                  String name) async {
    final response = await http.post(ApiService.url + '/register',
        headers: AuthService.getHeaders(),
        body: jsonEncode(<String, String>{
          'email': email,
          'password': password,
          'name': name,
          'password_confirmation': passwordConfirm
        }));
    final Map parsed = json.decode(response.body);
    if (response.statusCode == 201) {
      await AuthService.setToken(parsed['success']['token']);
      return "success";
    } else if(response.statusCode == 400) {
      String msg = "";
      Map errors = parsed["error"];
      errors.forEach((key, value) {
        for(int i = 0; i < value.length; i++) {
          msg += key + ": " + value[i] + "\r\n";
        }
      });
      return msg;
```

```
  } else if(await AuthService.checkUnauthenticated(response)) {
    return "somehow, the server says you are not authenticated. he should not say that.";
  }
  else {
    return "sorry, some unknown error occured during register";
  }
}


static bool logout() {
  http.post(ApiService.url + '/logout', headers: AuthService.getHeaders());
  AuthService.setToken(null);
  return true;
}


static bool unregister() {
  http.post(ApiService.url + '/unregister', headers: AuthService.getHeaders());
  AuthService.setToken(null);
  return true;
}


static Future<List<ShoppingList>> fetchShoppingLists() async {
  final response = await http.get(ApiService.url + '/lists', headers:
                                  AuthService.getHeaders());
  if (response.statusCode == 200) {
    List jsonResponse = json.decode(response.body);
    return jsonResponse.map((sl) => new ShoppingList.fromJson(sl)).toList();
  } else if(await AuthService.checkUnauthenticated(response)) {
    throw Exception("401"); //Unauthenticated
  }
  else {
    throw Exception("failed to load shopping lists from API");
  }
}


static Future<List<ShoppingEntry>> fetchShoppingEntries(int listId) async {
  final response = await http.get(ApiService.url + '/list/' +
      listId.toString() + '/entries', headers: AuthService.getHeaders());
  if (response.statusCode == 200) {
    List jsonResponse = json.decode(response.body);
    return jsonResponse.map((se) => new ShoppingEntry.fromJson(se)).toList();
  } else if(await AuthService.checkUnauthenticated(response)) {
    throw Exception("401"); //Unauthenticated
  }
  else {
    throw Exception("failed to load shopping entries from API");
  }
}


static Future<http.Response> deleteShoppingList(int id) async {
  final response = await http.delete(ApiService.url + '/list/' +
      id.toString(), headers: AuthService.getHeaders());
  AuthService.checkUnauthenticated(response);
  if(response.statusCode == 200 || response.statusCode == 401) return response;
  else throw Exception("failed to delete list over API");
}
```

```
static Future<http.Response> deleteListEntry(int id) async {
    final response = await http.delete(ApiService.url + '/entry/' +
        id.toString(), headers: AuthService.getHeaders());
    AuthService.checkUnauthenticated(response);
    if(response.statusCode == 200 || response.statusCode == 401) return response;
    else throw Exception("failed to delete entry over API");
  }


  static Future<http.Response> addShoppingList(String name) async {
    final response = await http.post(
      ApiService.url + '/list',
      headers: AuthService.getHeaders(),
      body: jsonEncode(<String, String>{
        'listname': name,
      }),
    );
    AuthService.checkUnauthenticated(response);
    if(response.statusCode == 201 || response.statusCode == 401) return response;
    else throw Exception("failed to save new list over API");
  }


  static Future<http.Response> addListEntry(int listId, int amount,
      String entryName) async {
    final response = await http.post(
      ApiService.url + '/list/' + listId.toString() +'/entry',
      headers: AuthService.getHeaders(),
      body: jsonEncode(<String, String>{
        'entryname': entryName,
        'amount': amount.toString()
      })
    );
    AuthService.checkUnauthenticated(response);
    if(response.statusCode == 201 || response.statusCode == 401) return response;
    else throw Exception("failed to save new entry over API");
  }
}
```

File: lib/Libs/AuthService.dart

Stellt den Views gekapselt die Methoden zur Authentifizierung zur Verfügung

```
import 'package:http/http.dart';
import 'package:flutter_secure_storage/flutter_secure_storage.dart';

class AuthService {

  static final storage = new FlutterSecureStorage();
  static String _token=null;
  static final Map<String, String> _headers = <String, String>{
    'Content-Type': 'application/json; charset=UTF-8',
    'Accept': 'application/json'
  };

  static String getToken() {
    return AuthService._token;
  }
```

```
  static setToken(String token, [bool skipSecureWriting]) async {
    AuthService._token = token;
    if(token != null) {
      AuthService._headers['Authorization'] = 'Bearer $token';
      if(skipSecureWriting != true) await AuthService.storage.write(key: 'token',
                                          value: token);
    } else {
      AuthService._headers.remove('Authorization');
      if(skipSecureWriting != true) await AuthService.storage
                                                      .delete(key: 'token');
    }
  }

  static Future<bool> init() async {
    String token = await storage.read(key: 'token');
    AuthService.setToken(token, true);
    return true;
  }

  static Map<String, String> getHeaders() {
    return AuthService._headers;
  }

  static Future<bool> checkUnauthenticated(Response response) async {
    if(response.statusCode == 401) {
      await AuthService.setToken(null);
      return true;
    }
    else return false;
  }

  static bool isAuth() {
    return AuthService._token != null;
  }
}
```

File: lib/Libs/MenuService.dart

Stellt den einzelnen Scaffolds der Sreens den Menu-Drawer zur Verfügung

```
import 'package:flutter/material.dart';
import 'package:material_design_icons_flutter/material_design_icons_flutter.dart';

import 'ApiService.dart';

class MenuService {

  static Drawer getDrawer(BuildContext context) {
    return Drawer(
        child: Builder(
          builder: (ctxt) =>Container(
            color: Theme.of(context).backgroundColor,
child: ListView(
    padding: EdgeInsets.zero,
    children: <Widget>[
      DrawerHeader(
        child: Container(
```

```
                    padding: const EdgeInsets.fromLTRB(16.0, 10.0, 16.0, 22.0),
                    child: Image.asset('assets/logosm.png',
                        filterQuality:FilterQuality.high,
                        fit: BoxFit.fitHeight),
                    ),
                    decoration: BoxDecoration(
                      color: Theme.of(context).primaryColor,
                    ),
                  ),
                  ListTile(
                    leading: Icon(MdiIcons.power,
                      color: Theme.of(context).primaryColor,
                    ),
                    title: Text('Logout', style: Theme.of(context)
                        .textTheme.subtitle1),
                    onTap: () {
                      ApiService.logout();
                      Navigator.pushNamedAndRemoveUntil(context,'/login',
                              (route) => false);
                    },
                  ),
                  ListTile(
                    leading: Icon(MdiIcons.accountMinus,
                      color: Theme.of(context).primaryColor,
                    ),
                    title: Text("Delete Account", style: Theme.of(context)
                        .textTheme.subtitle1),
                    onTap: () {
                      showDialog(
                        context: context,
                        builder: (BuildContext alertContext) {
                          return AlertDialog(
                            title: Text("Unregister"),
                            content: Text("Would you really like to delete "
                                "your account and all its data permanently from "
                                "our databases? You cannot undo this action."),
                            actions: [
                              FlatButton(
                                child: Text("Cancel"),
                                onPressed:  () {
                                  Navigator.pop(context);
                                },
                              ),
                              FlatButton(
                                child: Text("Delete Account"),
                                onPressed:  () {
                                  ApiService.unregister();
                                  Navigator.pushNamedAndRemoveUntil(context,
                                      '/login', (route) => false);
                                },
                              ),
                            ],
                          );
                        },
                      );
                    },
                  )
                ]
              )
            )
          )
        );
      }
    }
```

File: lib/Libs/ThemeService.dart

Stellt der MaterialDesign App das verwendete Theming zur Vefügung.

```
import 'package:flutter/material.dart';
import 'package:tinycolor/tinycolor.dart';
import 'package:google_fonts/google_fonts.dart';

class ThemeService {

  static final Color prime = const Color(0xff2A628F);
  static final Color back = const Color(0xffFCF9ED);
  static final Color accent = const Color(0xff13293D);
  static final Color text = const Color(0xff3B2D29);
  static final Color textAlt = const Color(0xff7A0F0F);

  static ThemeData getTheme(BuildContext context) {
    return ThemeData(
      primaryColor: ThemeService.prime,
      accentColor: ThemeService.accent,
      backgroundColor: ThemeService.back,
      scaffoldBackgroundColor: ThemeService.back,
      dialogBackgroundColor: ThemeService.back,
      buttonColor: ThemeService.accent,
      cursorColor: ThemeService.textAlt,
      focusColor: ThemeService.accent,
      highlightColor: ThemeService.text,
      indicatorColor: ThemeService.textAlt,
      dividerColor: ThemeService.prime,
      hintColor: ThemeService.text,
      //fontFamily: 'Georgia', //openSansTextTheme //ubuntuTextTheme //latoTextTheme
//robotoTextTheme
      primaryTextTheme: GoogleFonts.ubuntuTextTheme(
        TextTheme( //contrasts with primary color
          headline1: TextStyle(color: ThemeService.back),
          headline2: TextStyle(color: ThemeService.back),
          headline3: TextStyle(color: ThemeService.back),
          headline4: TextStyle(color: ThemeService.back),
          headline5: TextStyle(color: ThemeService.back, fontSize: 20.0, ),
          headline6: TextStyle(color: ThemeService.back),
          subtitle1: TextStyle(color: ThemeService.back),
          subtitle2: TextStyle(color: ThemeService.back),
          bodyText1: TextStyle(color: ThemeService.text),
          bodyText2: TextStyle(color: ThemeService.back),
          button: TextStyle(color: ThemeService.back),
          caption: TextStyle(color: ThemeService.back),
          overline: TextStyle(color: ThemeService.back),
        ),
      ),
      textTheme: GoogleFonts.ubuntuTextTheme(
        TextTheme(
          headline1: TextStyle(color: ThemeService.text),
          headline2: TextStyle(color: ThemeService.text),
          headline3: TextStyle(color: ThemeService.text),
          headline4: TextStyle(color: ThemeService.text),
          headline5: TextStyle(color: ThemeService.text),
          headline6: TextStyle(color: ThemeService.text),
          subtitle1: TextStyle(color: ThemeService.text),
          subtitle2: TextStyle(color: ThemeService.textAlt),
          bodyText1: TextStyle(color: ThemeService.back),
          bodyText2: TextStyle(color: ThemeService.text),
          button: TextStyle(color: ThemeService.back),
```

```
          caption: TextStyle(color: ThemeService.text),
          overline: TextStyle(color: ThemeService.text),
        ),
      ),
      floatingActionButtonTheme: FloatingActionButtonThemeData(
        backgroundColor: ThemeService.accent,
        foregroundColor: ThemeService.back,
      ),
      buttonTheme: ButtonThemeData(
        buttonColor: ThemeService.accent, // Background color (orange in my case).
        textTheme: ButtonTextTheme.accent,
        colorScheme: Theme.of(context).colorScheme.copyWith(secondary: ThemeService.back)
      ),
      inputDecorationTheme: InputDecorationTheme(
          enabledBorder: UnderlineInputBorder(
            borderSide: BorderSide(color:
TinyColor(ThemeService.accent).lighten(12).color),
          ),
          focusedBorder: UnderlineInputBorder(
            borderSide: BorderSide(color: ThemeService.accent),
          ),
          border: UnderlineInputBorder(
            borderSide: BorderSide(color:
TinyColor(ThemeService.accent).lighten(12).color),
          )
      )
    );
  }

  static TextTheme getAlternativeTextTheme() {
    return GoogleFonts.caveatTextTheme(
      TextTheme( //contrasts with primary color
        headline1: TextStyle(color: ThemeService.text),
        headline2: TextStyle(color: ThemeService.text),
        headline3: TextStyle(color: ThemeService.text),
        headline4: TextStyle(color: ThemeService.text),
        headline5: TextStyle(color: ThemeService.text),
        headline6: TextStyle(color: const Color(0xffd5d5d5), fontSize: 26.0),
        subtitle1: TextStyle(color: ThemeService.text),
        subtitle2: TextStyle(color: ThemeService.text),
        bodyText1: TextStyle(color: ThemeService.text),
        bodyText2: TextStyle(color: ThemeService.text),
        button: TextStyle(color: ThemeService.text),
        caption: TextStyle(color: ThemeService.text),
        overline: TextStyle(color: ThemeService.text),
      ),
    );
  }
}
```

File: lib/Views/AllListsView.dart

Der Screen mit allen Einkaufslisten aufgelistet

```
import 'package:intl/intl.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import '../Classes/ShoppingList.dart';
import '../Libs/ApiService.dart';
import '../Libs/MenuService.dart';
import '../Libs/ThemeService.dart';
```

```
class AllListsView extends StatefulWidget {
  @override
  AllListsViewState createState() => new AllListsViewState();
}

class AllListsViewState extends State<AllListsView> {

  final DateFormat myDFormat = DateFormat('yyyy-MM-dd HH:mm');

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      endDrawer: MenuService.getDrawer(context),
      appBar: AppBar(
        title: Text('Shopping Lists'),
        actions: <Widget>[Builder(
          builder: (ctxt) => Container(
            padding: const EdgeInsets.fromLTRB(0, 0, 15, 0),
            child: IconButton(
              icon: Icon(
                Icons.menu,
                color: Theme.of(context).backgroundColor,
              ),
              onPressed: () => Scaffold.of(ctxt).openEndDrawer(),
            )
          ))
        ],
      ),
      body: Center(
        child: FutureBuilder(
          future: ApiService.fetchShoppingLists(),
          builder: (context, snapshot) {
            if (snapshot.hasData) {
              if(snapshot.data.length == 0) {
                return Column(
                  children: <Widget>[
                    Expanded(
//                      padding: const EdgeInsets.fromLTRB(0, 0, 30, 50),
                      child: Container(
                          padding: const EdgeInsets.all(50.0),
                          child: Text("vast emptyness", style: ThemeService
                              .getAlternativeTextTheme().headline6)
                      )
                    ),
                    Container(
                        alignment: Alignment.bottomRight,
                        padding: const EdgeInsets.fromLTRB(0, 0, 30.0, 50.0),
                        child:Image.asset('assets/first_list_hint.png',
                            height: 300, filterQuality:FilterQuality.high)
                    )
                  ]
                );
              } else return _shoppingListView(snapshot.data);
            } else if (snapshot.hasError) {
              if(snapshot.error.toString() == '401') {
                Navigator.pushNamedAndRemoveUntil(context,'/login', (route) => false);
              }
              else {
                Scaffold.of(context).showSnackBar(SnackBar(content:
                Text('network error on loading lists')));
                return null; //Text("${snapshot.error}");
              }
            }
            // By default, show a loading spinner
            return CircularProgressIndicator();
          },
        )
```

```
      ),
      floatingActionButton: Builder(
        builder: (ctxt) => FloatingActionButton(
          onPressed: () async {
            final dynamic result = await Navigator.pushNamed(context, '/new-list');
            if(result == true) {
              Scaffold.of(ctxt).showSnackBar(
                  SnackBar(content: Text('List added successfully.')));
            }
            this.setState(() {});
          },
          tooltip: 'add new shopping list',
          child: const Icon(Icons.add),
        )
      )
    );
  }

  ListView _shoppingListView(data) {
    return ListView.builder(
        itemCount: data.length,
        itemBuilder: (context, index) {
          return _tile(data[index], Icons.list, context);
        });
  }

  ListTile _tile(ShoppingList shoppingList, IconData icon,
      BuildContext context) => ListTile(
    title: Text(shoppingList.listname,
        style: TextStyle(
          fontWeight: FontWeight.w500,
          fontSize: 20,
        )),
    subtitle: Text(myDFormat.format(DateTime.parse(shoppingList.created_at)
        .toLocal()), style: Theme.of(context).textTheme.subtitle2),
    leading: Icon(
      icon,
      color: Theme.of(context).primaryColor,
    ),
    trailing: IconButton(icon: Icon(Icons.delete, color: Theme.of(context)
        .indicatorColor), onPressed: () async {
      await ApiService.deleteShoppingList(shoppingList.id);
      Scaffold
          .of(context)
          .showSnackBar(SnackBar(content: Text('List deleted successfully.')));
      this.setState(() {});
    }),
    onTap: () async {
      Navigator.pushNamed(context, '/single-list', arguments: shoppingList);
      this.setState(() {});
    },
  );

}
```

File: lib/Views/LoginView.dart

Der Login-Bereich der App

```dart
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:material_design_icons_flutter/material_design_icons_flutter.dart';
import '../Libs/ApiService.dart';

class LoginView extends StatefulWidget {
  LoginView({Key key}) : super(key: key);

  @override
  LoginViewState createState() {
    return LoginViewState();
  }
}

class LoginViewState extends State<LoginView> {
  final _formKey = GlobalKey<FormState>();
  LoginViewState() : super();
  final myEMailController = TextEditingController();
  final myPasswordController = TextEditingController();
  bool _autoValidate = false;

  void _validateInputs (BuildContext ctxt) async {
    if (_formKey.currentState.validate()) {
      bool login = false;
      login = await ApiService.login(myEMailController.text, myPasswordController.text);
      if(!login) {
        Scaffold.of(ctxt).showSnackBar(SnackBar(content:
        Text('Please check username and password and try again')));
      } else {
        Scaffold.of(ctxt).showSnackBar(SnackBar(content:
        Text('Login Successful')));
        Navigator.pushNamedAndRemoveUntil(context,
            '/', (route) => false);
      }
    } else {
      //If all data are not valid then start auto validation.
      Scaffold.of(ctxt).showSnackBar(SnackBar(content:
      Text('Please check your input fields')));
      setState(() {
        _autoValidate = true;
      });
    }
  }

  String validateEmail(String value) {
    Pattern pattern =
        r'^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\'
        r'.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$';
    RegExp regex = new RegExp(pattern);
    if (!regex.hasMatch(value))
      return 'Enter Valid Email';
    else
      return null;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        backgroundColor: Theme.of(context).primaryColor,
        body: SingleChildScrollView(
          child:Builder(
            builder: (ctxt) => Center(
              child: Padding(
                padding: const EdgeInsets.all(35.0),
                child:Form(
                  key: _formKey,
```

```
                autovalidate: _autoValidate,
                child: Column(
                  children: <Widget>[
                    new SizedBox(
                      height: 40.0,
                    ),
                    Image.asset('assets/logosm.png', height: 200,
                        filterQuality:FilterQuality.high),
                    new SizedBox(
                      height: 40.0,
                    ),
                    TextFormField(
                      style: TextStyle(color: Theme.of(context).backgroundColor),
                      decoration: InputDecoration(labelText: 'E-Mail',
                          labelStyle: Theme.of(context).primaryTextTheme.bodyText2),
                      keyboardType: TextInputType.emailAddress,
                      controller: myEMailController,
                      validator: validateEmail,
                    ),
                    TextFormField(
                      style: TextStyle(color: Theme.of(context).backgroundColor),
                      decoration: InputDecoration(labelText: 'Password',
                          labelStyle: Theme.of(context).primaryTextTheme.bodyText2),
                      keyboardType: TextInputType.text,
                      obscureText: true,
                      controller: myPasswordController
                    ),
                    new SizedBox(
                      height: 20.0,
                    ),
                    ConstrainedBox(
                      constraints: const BoxConstraints(minWidth: double.infinity),
                      child: RaisedButton(
                        onPressed: () {
                          _validateInputs(ctxt);
                        },
                        child: Text('LOGIN'),
                      )
                    )
                  ]
                )
              )
            )
          )
        ),
        floatingActionButton: Builder(
            builder: (ctxt) => FloatingActionButton(
              onPressed: () {
                Navigator.pushNamed(context, '/register');
              },
              tooltip: 'Switch to Register',
              child: Icon(MdiIcons.accountPlus)
            )
        )
    );
  }
}
```

File: lib/Views/NewEntryView.dart

Der Screen zum Erstellen eines neuen Postens auf einer Einkaufsliste:

```dart
import 'package:http/http.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import '../Classes/ShoppingList.dart';
import '../Libs/ApiService.dart';
import '../Libs/MenuService.dart';

class NewEntryView extends StatefulWidget {
  NewEntryView({Key key}) : super(key: key);
  @override
  NewEntryViewState createState() {
    return NewEntryViewState();
  }
}

class NewEntryViewState extends State<NewEntryView> {
  final _formKey = GlobalKey<FormState>();
  ShoppingList shoppingList;
  NewEntryViewState() : super();
  final myNameController = TextEditingController();
  final myAmountController = TextEditingController(text: '1');

  bool _autoValidate = false;

  void _validateInputs (BuildContext ctxt) async {
    if (_formKey.currentState.validate()) {
      Scaffold.of(ctxt).showSnackBar(SnackBar(content:Text('Creating entry...')));
      Response r = await ApiService.addListEntry(shoppingList.id,
          int.parse(myAmountController.text), myNameController.text);
      if(r.statusCode == 401) {
        Navigator.pushNamedAndRemoveUntil(context, '/login', (route) => false);
      } else Navigator.pop(ctxt, true);
    } else {
      //If all data are not valid then start auto validation.
      Scaffold.of(ctxt).showSnackBar(SnackBar(content:
      Text('Please check your input fields')));
      setState(() {
        _autoValidate = true;
      });
    }
  }

  String validateName(String value) {
    RegExp regex = RegExp('^[a-zA-Z0-9_ ]*\$');
    if (!regex.hasMatch(value)) {
      return 'Only characters and digits are allowed';
    }
    else if(value.length > 60) return 'Reduce name to less than 60 characters';
    else if(value == "") return 'Name should not be empty';
    return null;
  }

  String validatorAmount(String value) {
    if(int.parse(value) > 9999) return "value should be less than 9999";
    else return null;
  }

  @override
  Widget build(BuildContext context) {
    this.shoppingList = ModalRoute.of(context).settings.arguments;
    return Scaffold(
        endDrawer: MenuService.getDrawer(context),
        appBar: AppBar(
          title: Text(shoppingList.listname + ": New Entry"),
          actions: <Widget>[Builder(
```

```
            builder: (ctxt) => Container(
                padding: const EdgeInsets.fromLTRB(0, 0, 15, 0),
                child: IconButton(
                  icon: Icon(
                    Icons.menu,
                    color: Theme.of(context).backgroundColor,
                  ),
                  onPressed: () => Scaffold.of(ctxt).openEndDrawer(),
                )
            ))
      ],
    ),
    body: SingleChildScrollView(
        child:Builder(
        builder: (ctxt) => Center(
            child: Padding(
              padding: const EdgeInsets.all(40.0),
              child:Form(
                key: _formKey,
                autovalidate: _autoValidate,
                child: Column(
                    children: <Widget>[
                      TextFormField(
                        decoration: InputDecoration(labelText: 'Entryname'),
                        controller: myNameController,
                        validator: validateName
                      ),
                      TextFormField(
                        decoration: InputDecoration(
                            labelText: 'Amount'
                        ),
                        inputFormatters: [WhitelistingTextInputFormatter.digitsOnly],
                        keyboardType: TextInputType.number,
                        controller: myAmountController,
                        validator: validatorAmount
                      ),
                      new SizedBox(
                        height: 20.0,
                      ),
                      ConstrainedBox(
                          constraints: const BoxConstraints(minWidth: double.infinity),
                          child: RaisedButton(
                            onPressed: () {
                              _validateInputs(ctxt);
                            },
                            child: Text('Save'),
                          )
                      )
                    ]
                ))
            )
          )
        )
    ));
  }
}
```

File: lib/Views/NewListView.dart

Der Screen, um eine neue Liste anzulegen

```dart
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart';
import '../Libs/ApiService.dart';
import '../Libs/MenuService.dart';

class NewListView extends StatefulWidget {
  @override
  NewListFormState createState() {
    return NewListFormState();
  }
}

class NewListFormState extends State<NewListView> {
  final _formKey = GlobalKey<FormState>();
  final myNameController = TextEditingController();
  bool _autoValidate = false;

  void _validateInputs (BuildContext ctxt) async {
    if (_formKey.currentState.validate()) {
      Scaffold.of(ctxt)
          .showSnackBar(SnackBar(content: Text('Creating list...')));
      Response r = await ApiService.addShoppingList(myNameController.text);
      if(r.statusCode == 401) {
        Navigator.pushNamedAndRemoveUntil(context,
            '/login', (route) => false);
      }
      else Navigator.pop(ctxt, true);
    } else {
      //If all data are not valid then start auto validation.
      Scaffold.of(ctxt).showSnackBar(SnackBar(content:
      Text('Please check your input fields')));
      setState(() {
        _autoValidate = true;
      });
    }
  }


  String validateName(String value) {
    RegExp regex = RegExp('^[a-zA-Z0-9_ ]*\$');
    if (!regex.hasMatch(value)) {
      return 'Only characters and digits are allowed';
    }
    else if(value.length > 60) return 'Reduce name to less than 60 characters';
    else if(value == "") return 'Name should not be empty';
    return null;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      endDrawer: MenuService.getDrawer(context),
      appBar: AppBar(
        title: Text('New List'),
        actions: <Widget>[Builder(
            builder: (ctxt) => Container(
                padding: const EdgeInsets.fromLTRB(0, 0, 15, 0),
                child: IconButton(
                  icon: Icon(
                    Icons.menu,
                    color: Theme.of(context).backgroundColor,
                  ),
                  onPressed: () => Scaffold.of(ctxt).openEndDrawer(),
                )
            ))
      ],
```

```
      ),
      body: SingleChildScrollView(
        child:Builder(
          builder: (ctxt) => Center(
            child: Padding(
            padding: const EdgeInsets.all(40.0),
            child:Form(
              key: _formKey,
              autovalidate: _autoValidate,
              child: Column(
                children: <Widget>[
                  TextFormField(
                    decoration: InputDecoration(
                      labelText: 'Listname'
                    ),
                    controller: myNameController,
                    // The validator receives the text that the user has entered.
                    validator: validateName,
                  ),
                  new SizedBox(
                    height: 20.0,
                  ),
                  ConstrainedBox(
                    constraints: const BoxConstraints(minWidth: double.infinity),
                      child: RaisedButton(
                        onPressed: () {
                          _validateInputs(ctxt);
                        },
                        child: Text('Save'),
                      )
                    )
                  ]
                )
              )
            )
          )
        )
      )
    );
  }
}
```

File: lib/Views/RegisterView.dart

Der Screen, um einen neuen User zu registrieren

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import '../Libs/ApiService.dart';

class RegisterView extends StatefulWidget {
  RegisterView({Key key}) : super(key: key);

  @override
  RegisterViewState createState() {
    return RegisterViewState();
  }
}
```

```
class RegisterViewState extends State<RegisterView> {
  final _formKey = GlobalKey<FormState>();
  RegisterViewState() : super();
  final myEMailController = TextEditingController();
  final myNameController = TextEditingController();
  final myPasswordController = TextEditingController();
  final myPasswordConfirmController = TextEditingController();
  bool _autoValidate = false;

  void _validateInputs (BuildContext ctxt) async {
    if (_formKey.currentState.validate()) {
      String register = await ApiService.register(myEMailController.text,
        myPasswordController.text, myPasswordConfirmController.text,
          myNameController.text);
      if(register == "success") {
        Scaffold.of(ctxt).showSnackBar(SnackBar(content:
        Text('Login and Register Successful')));
        Navigator.pushNamedAndRemoveUntil(context,
            '/', (route) => false);
      } else {
        Scaffold.of(ctxt).showSnackBar(SnackBar(content:
        Text(register)));
      }
    } else {
      //If all data are not valid then start auto validation.
      Scaffold.of(ctxt).showSnackBar(SnackBar(content:
      Text('Please check your input fields')));
      setState(() {
        _autoValidate = true;
      });
    }}

  String validateEmail(String value) {
    Pattern pattern =
        r'^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\'
        r'.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$';
    RegExp regex = new RegExp(pattern);
    if (!regex.hasMatch(value))
      return 'Enter Valid Email';
    else if(value == "") return 'E-Mail should not be empty';
    else return null;
  }

  String validatePassword(String value) {
      if(value.length < 8) return 'Password should at least be 8 characters';
      else if(value.length == 0) return 'Password should not be empty';
      else return null;
  }

  String validateName(String value) {
    RegExp regex = RegExp('^[a-zA-Z0-9_ ]*\$');
    if (!regex.hasMatch(value)) {
     return 'Only characters and digits are allowed';
   }
   else if(value.length > 60) return 'Reduce name to less than 60 characters';
   else if(value.length == 0) return 'Name should not be empty';
   return null;
}
@override
Widget build(BuildContext context) {
  return Scaffold(
      backgroundColor: Theme.of(context).primaryColor,
      body: SingleChildScrollView(
        child:Builder(
          builder: (ctxt) => Center(
```

```
child: Padding(
  padding: const EdgeInsets.fromLTRB(35.0, 50.0, 35.0, 35.0),
  child:Form(
    key: _formKey,
    autovalidate: _autoValidate,
    child: Column(
      children: <Widget>[
        RichText(
          textAlign: TextAlign.center,
          text: new TextSpan( style: Theme.of(context)
              .primaryTextTheme.headline5,
            children: <TextSpan>[
              TextSpan(text: 'Sign up for '),
              TextSpan(text: 'ME WANT THAT',
                  style: new TextStyle(fontWeight: FontWeight.bold)),
              TextSpan(text: ' and create an account:')
            ]
          )
        ),
        new SizedBox(
          height: 20.0,
        ),
        TextFormField(
          style: TextStyle(color: Theme.of(context).backgroundColor),
          decoration: InputDecoration(labelText: 'Name',
              labelStyle: Theme.of(context).primaryTextTheme.bodyText2),
          keyboardType: TextInputType.text,
          controller: myNameController,
          validator: validateName
        ),
        TextFormField(
          style: TextStyle(color: Theme.of(context).backgroundColor),
          decoration: InputDecoration(labelText: 'E-Mail',
              labelStyle: Theme.of(context).primaryTextTheme.bodyText2),
          keyboardType: TextInputType.emailAddress,
          controller: myEMailController,
          validator: validateEmail,
        ),
        TextFormField(
          style: TextStyle(color: Theme.of(context).backgroundColor),
          decoration: InputDecoration(labelText: 'Password',
              labelStyle: Theme.of(context).primaryTextTheme.bodyText2),
          keyboardType: TextInputType.text,
          obscureText: true,
          controller: myPasswordController,
          validator: validatePassword
        ),
        TextFormField(
          style: TextStyle(color: Theme.of(context).backgroundColor),
          decoration: InputDecoration(labelText: 'Password again',
              labelStyle: Theme.of(context).primaryTextTheme.bodyText2),
          keyboardType: TextInputType.text,
          obscureText: true,
          controller: myPasswordConfirmController,
          validator: validatePassword
        ),
        new SizedBox(
          height: 30.0,
        ),
        ConstrainedBox(
          constraints: const BoxConstraints(minWidth: double.infinity),
          child: RaisedButton(
            color: Theme.of(context).accentColor,
            textColor: Theme.of(context).backgroundColor,
            onPressed: () {
              _validateInputs(ctxt);
            },
```

```
                        child: Text('REGISTER'),
                      )
                    )
                  ]
                )
              )
            )
          )
        )
      ),
      floatingActionButton: Builder(
          builder: (ctxt) => FloatingActionButton(
            backgroundColor: Theme.of(context).accentColor,
            onPressed: () {
              Navigator.pop(context);
            },
            tooltip: 'Switch to Login',
            child: Icon(Icons.arrow_back),
          )
      )
    );
  }
}
```

File: lib/Views/SingleListView.dart

Der Screen, um alle Listenposten einer Liste anzuzeigen

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import '../Classes/ShoppingList.dart';
import '../Classes/ShoppingEntry.dart';
import '../Libs/ApiService.dart';
import '../Libs/MenuService.dart';
import '../Libs/ThemeService.dart';

class SingleListView extends StatefulWidget {
  SingleListView({Key key}) : super(key: key);
  @override
  SingleListViewState createState() => new SingleListViewState();
}
class SingleListViewState extends State<SingleListView> {

  ShoppingList shoppingList;
  SingleListViewState() : super();

  navigateNewEntry(BuildContext context, BuildContext snackbarContext) async {
    final dynamic result = await Navigator.pushNamed(context,
        '/new-entry', arguments: shoppingList);
    if(result == true) Scaffold
        .of(snackbarContext)
        .showSnackBar(SnackBar(content: Text('Entry added successfully.')));
    this.setState(() {});
  }
@override
Widget build(BuildContext context) {
  this.shoppingList = ModalRoute.of(context).settings.arguments;
  return Scaffold(
      endDrawer: MenuService.getDrawer(context),
```

```
      appBar: AppBar(
        title: Text('List: ' + shoppingList.listname),
        actions: <Widget>[Builder(
            builder: (ctxt) => Container(
                padding: const EdgeInsets.fromLTRB(0, 0, 15, 0),
                child: IconButton(
                  icon: Icon(
                    Icons.menu,
                    color: Theme.of(context).backgroundColor,
                  ),
                  onPressed: () => Scaffold.of(ctxt).openEndDrawer(),
                )
            ))
        ],
      ),
      body: Center(
      child: FutureBuilder(
        future: ApiService.fetchShoppingEntries(shoppingList.id),
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            if(snapshot.data.length == 0) {
              return Column(
                  children: <Widget>[
                    Expanded(
                        child: Container(
                            padding: const EdgeInsets.all(50.0),
                            child: Text("vast emptyness",
                                style: ThemeService.getAlternativeTextTheme().headline6)
                        )
                    ),
                    Container(
                        alignment: Alignment.bottomRight,
                        padding: const EdgeInsets.fromLTRB(0, 0, 30.0, 50.0),
                        child:Image.asset('assets/first_entry_hint.png',
                            height: 300, filterQuality:FilterQuality.high)
                    )
                  ]
              );
            } else return _singleListView(snapshot.data);
          } else if (snapshot.hasError) {
            if(snapshot.error.toString() == '401') {
              Navigator.pushNamedAndRemoveUntil(context,'/login', (route) => false);
            }
            else {
              Scaffold.of(context).showSnackBar(SnackBar(content:
                                        Text('network error on loading lists')));
              return null; //Text("${snapshot.error}");
            }
          }
          // By default, show a loading spinner
          return CircularProgressIndicator();
        },
      )
    ),
    floatingActionButton: Builder(
      builder: (ctxt) => FloatingActionButton(
        onPressed: () => navigateNewEntry(context, ctxt),
        tooltip: 'add new shopping list',
        child: const Icon(Icons.add),
      )
    )
  );
}

ListView _singleListView(data) {
  return ListView.builder(
      itemCount: data.length,
```

```
        itemBuilder: (context, index) {
          return _tile(data[index], Icons.local_offer, context);
        });
  }


  ListTile _tile(ShoppingEntry entry, IconData icon, BuildContext context) => ListTile(
    title: Text(entry.entryname,
        style: TextStyle(
          fontWeight: FontWeight.w500,
          fontSize: 20,
        )),
    subtitle: Text('Amount: '+entry.amount.toString() + 'x'),
    leading: Icon(
      icon,
      color: Theme.of(context).primaryColor,
    ),
    trailing: IconButton(icon: Icon(Icons.delete, color: Theme.of(context)
        .indicatorColor), onPressed: () async {
      await ApiService.deleteListEntry(entry.id);
      Scaffold.of(context).showSnackBar(SnackBar(content: Text
                                                ('Entry deleted successfully.')));
      this.setState(() {});
    }),
  );
} //end class SingleListViewState
```

File: pubspec.yaml

Konfigurationsfile für die Pakete für den Pub Paketmanager

```
name: ShoppingList_Flutter
description: Shopping List
publish_to: 'none' # Remove this line if you wish to publish to pub.dev
version: 1.0.0+1

environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  http: any
  intl: ^0.16.1
  flutter_secure_storage: ^3.3.3
  font_awesome_flutter: ^8.8.1
  google_fonts: 1.1.0
  material_design_icons_flutter: 4.0.5345
  tinycolor: 1.0.2
  flutter:
    sdk: flutter
  cupertino_icons: ^0.1.3

dev_dependencies:
  flutter_launcher_icons: 0.7.5
  flutter_test:
    sdk: flutter
```

```
flutter_icons:
  ios: true
  android: true
  image_path_ios: "assets/launcher/icon.png"
  image_path_android: "assets/launcher/icon.png"
  adaptive_icon_background: "assets/launcher/background.png"
  adaptive_icon_foreground: "assets/launcher/foreground.png"

flutter:
  uses-material-design: true
  assets:
    - assets/
```

Die Implementierung ist sehr einfach gehalten, um die Sourcecode-Abschnitte kurz und verständlich zu halten. Es wurden zwei Klassen angelegt, ShoppingList.dart und ShoppingEntry.dart, welche die Models repräsentieren.

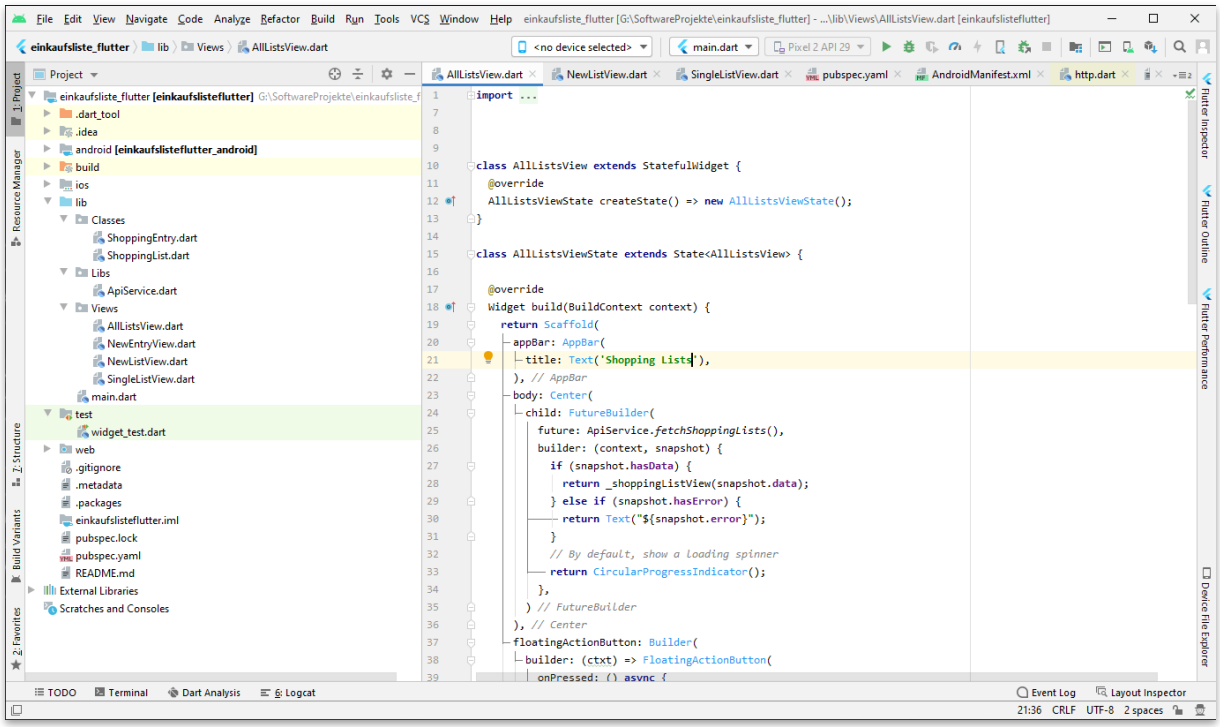
Einige Service-Libraries (ApiService, MenuService, AuthSerivce und ThemeService) kümmern sich um gekapselte Dienste wie den asynchronen Austausch der Daten mit der Server Rest API, Authentifizierung oder Theming.

Jeder Screen wird durch ein stateful Widget repräsentiert, welches in seiner State-Klasse mithilfe des ApiServices die Model-Instanzen und Collections handelt.

7.1.1.8 Accessoires und Hilfsmittel

Neben dem bereits angeführten **Paketmanager PUB**, der Sprachumgebung **DART** und dem **Framework FLUTTER** bietet Google eine großartige Integration in die Entwicklungsumgebung **Android Studio** an. Android Studio ist nicht nur gratis verfügbar, sondern auch der quasi Standard zur Entwicklung von Android Apps. Google arbeitet dafür mit der Firma *JetBrains* zusammen, auf deren Java IDE Produkt *„IntelliJ IDEA"* das Android Studio im Kern basiert.

Nach der Installation der entsprechenden Flutter und Dart Plugins (sehr einfach und intuitiv über den integrierten Plugin-Store) hat man im Android-Studio die Möglichkeit, Flutter Projekte zu öffnen, was die Umgebung in eine mächtige IDE (Integrated Development Environment) für Flutter verwandelt.



Sowohl **Code-Highlighting**, Starten von **Emulationsdevices**, **Debugging** mit Haltepunkten, **Builden** und **Deployment** funktionieren einwandfrei und ohne Fragen aufzuwerfen.

Der Größte Vorteil hier ist vermutlich, dass ein einziges Werkzeug für die gesamte Entwicklung ausreicht, geht man von einer Veröffentlichung nur im Android Play Store aus. Der Gegenentwurf anderer Hybrid-App Frameworks sieht vor, den tatsächlichen Entwicklungsprozess in einem externen Tool zu vollziehen und von dort ein Android-Studio Projekt zu exportieren, welches danach mit Android Studio geöffnet und für den finalen Build verwendet wird. Obwohl dieser Vorgang den meisten Entwicklern bereits aus vielen anderen Sprachen und Prozessketten geläufig ist, wirkt er im Vergleich zu der eleganten *Ein-System-Lösung* von Flutter klobig und unnötig.

Flutter funktioniert allerdings nicht ausschließlich mit dem Android Studio, sondern kann mit allen möglichen Werkzeugen programmiert werden. Weiters hervor getan haben sich der Editor **„Visual Studio Code“** mit seinen Flutter und Dart Plugins, sowie eine reine Entwicklung mit einem Texteditor und Builden, Debuggen und Inspizieren über CLI und Browser.

Die Stichworte *„Inspizieren“* und *„Browser“* sind hier besonders zu erwähnen, bietet Flutter doch mit den Flutter **Dev-Tools** (siehe Kapitel *Command Line Tools*) eine großartige Möglichkeit, seine App im Browser als WebApp laufen zu lassen und viele Manipulationen und Debugging Prozesse darauf auszuführen wie Performance-Profiling und Inspizieren des Component Trees.

7.1.1.9 Paketverwaltung

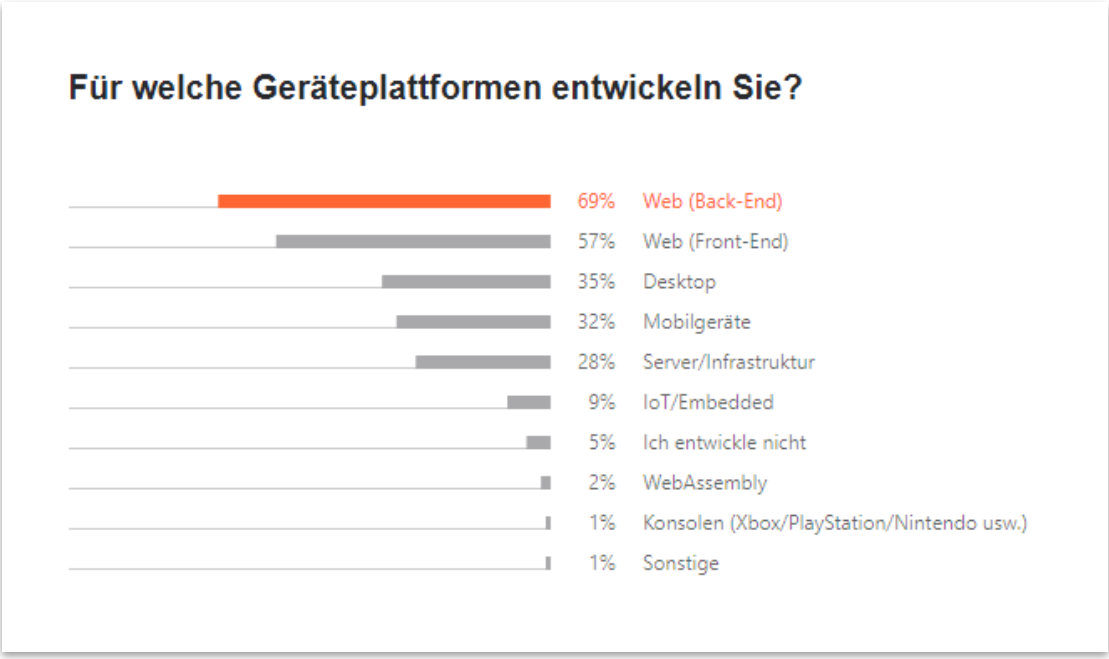
Das Paketmanagementsystem **PUB** (https://pub.dev/) bietet, entsprechend der Natur der zugrundeliegenden Sprache **DART**, sowohl Erweiterungen und Spezialfälle für UI Elemente sowie Skripte, Algorithmen, Logiken und Helferleins an. So vielseitig wie sich die Sprache selbst zeigt, so vielschichtig präsentiert sich auch das Paketmanagement dahinter. Kartenintegration über Maps? Komfortableres Ansprechen von REST Apis? Eine spezielle ListView Komponente? Ein alternatives Iconset? **PUB** hat meistens Antworten zu diesen Fragen.

7.1.1.10 Testimonials

- Alibaba
- Google
- Google Ads
- Tencent
- New York Times

7.1.2 Ionic Software Development Kit

In der JetBrains Developer Survey 2020 wurde den Entwicklern die Frage gestellt, für welche Geräteplattformen sie entwickeln. 57% davon gaben an, dass sie Web Frontend Entwickler sind. 32% deklarieren sich als Mobile Entwickler. Genau diese Personen sind die Zielgruppe von Frameworks wie Ionic, bieten sie doch dem Webentwickler ein Toolset, um eine großartige App entwerfen und implementieren zu können.



Quelle: JetBrains Developer Survey 2020

Ionic ist ein großartiges Cross Plattform Framework für die App-Entwicklung für Android, iOS und das Web. Außerdem kann damit auch für den Desktop und die Windows 10 Universal Plattform entwickelt werden.

Natürlich erfindet dieses Framework das Rad nicht komplett neu. Vielmehr stellt es eine Sammlung aus weit verbreiteten und akzeptierten Frameworks und Libraries dar, die zusammengenommen dieses leicht zu bedienende und sehr umfangreiche Software Development Kit ausmachen.

Der Autor Andreas Dorman schreibt dazu folgendes einprägsame Zitat:

> *The creators of Ionic didn't reinvent the wheel. They made it even rounder!*
>
> *(Dormann, 2019, p. 5)*

Gemeint ist damit ein Zusammenspiel aus Technologien wie Javascript und Angular (in der aktuellsten Version sind statt Angular auch React oder VueJS möglich), HTML5, CSS, eine Brückenschicht, welche die WebApp packt und die nativen APIs exponiert (früher Apache Cordova, heute in der aktuellsten Version wird dafür Capacitor verwendet), ein Toolset zum Transpilen des JS Codes, ein Ecosystem an verwendbaren Web-Paketen und verschiedene Build- und Pack-Tools.

7.1.2.1 Entwicklungsparadigmen

Grundsätzlich entwickelt man Ionic Apps mit JavaScript, HTML und CSS. Es wird allerdings für den JavaScript Part eine MVVM Library vorausgesetzt, welche die App-Logik beherbergt.

In früheren Versionen war diese Library verpflichtend Angular, in der aktuellsten Version wurde der Kern des Ionic Frameworks so exponiert, dass auch wahlweise React oder VueJS als Träger verwendet werden können.

Aber egal ob Angular, React oder Vue, die verwendete JavaScript-Library hat durchwegs „nur" die Aufgabe, die View upzudaten, den DOM zu manipulieren, Komponenten wiederverwendbar zu machen und gewisse Lifecycle Hooks anzubieten. (Dormann, 2019, p. 5) Deswegen sollte man mit mindestens einer dieser Technologien bereits im Web gearbeitet haben, will man sich die Lernkurve dieser Libraries ersparen.

In unserer Beispiel-App „Shopping List" werden wir anstatt Angular die Library VueJS verwenden, da sie meiner Meinung nach von den Dreien am einfachsten zu lesen und verstehen ist.

The core library of Vue.js, like React, is only for manipulating the view layer from the MVC architectural pattern.

(Gore, 2017, p. 10)

Vorteile von VueJS sind unter anderem Templates, Direktiven, Reaktivität, Komponenten (auch „Single File"), eine flache Lernkurve und eine mittlerweile weite Verbreitung unter vielen Web Developern.

Ein paar Worte, die Andreas Dormann über Angular schreibt sollten hier unbedingt erwähnt sein, da es auch vollinhaltlich auf VueJS zutrifft:

Client-side programming has become extremely complex in recent times. The requirements for a web application are now just as high as for a desktop application – if not higher. The desire for a framework that enables the developer to cover all these requirements on the web is thus getting bigger.

Angular is a client-side framework that makes it possible to create web applications. It helps the developer to bring known and new architectural concepts to the client and develop complex applications. The work can be aligned not only on the server, but via JavaScript on the client. The application can fulfill modern architectural concepts by separating the responsibilities and is thus more maintainable and testable.

(Dormann, 2019, p. 25)

Um die Rolle von Angular/React/Vue innerhalb von Ionic zu verstehen, hilft möglicherweise auch folgendes Zitat:

Angular helps with many emerging issues such as creating components, data binding to the UI/HTML, transformation of data against the UI with pipes, outsourcing of „work" in services, communication with an API, etc. Angular makes it possible to communicate with components, feed them with data and receive events from components; it makes components reusable and more isolated.

(Dormann, 2019, p. 26)

Apps, die mit Ionic erstellt werden, sind im Kern echte Websites oder Progressive Web Apps. Das bedeutet, eine WebView wird unsichtbar auf dem Phone geöffnet und die WebApp wird auf dieser WebView gerendert. Als völlig normale HTML5/JS/CSS Website. Ein großer Unterschied ist allerdings, dass der HTML-Code und die Assets nicht von einem Webserver geladen werden, sondern mit der App ausgeliefert werden, weswegen auch eine Offline-App ohne Probleme erstellt werden kann.

Da sich Ionic nicht der nativen UI Elemente des jeweiligen Betriebssystems bedienen kann (da es wie bereits erwähnt eine WebApp ist), sind in das Framework HTML UI Komponenten integriert, welche die Grundbausteine für die Apps darstellen.

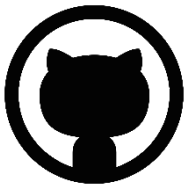
> *Ionic apps are made of high-level building blocks called Components, which allow you to quickly construct the UI for your app. Ionic comes stock with an umber of components, including cards, lists and tabs.*
>
> *(Dormann, 2019, p. 145)*

7.1.2.2 Android Package Kit (APK)

Die Android-Version dieser IONIC App kann downgeloadet und ausprobiert werden. Zusätzlich liegt sie dem Speichermedium dieser Arbeit bei. Zum Downloaden, einfach den QR Link scannen oder dem Link folgen.

https://me-want-that.com/downloads/MeWantThatIonic.apk

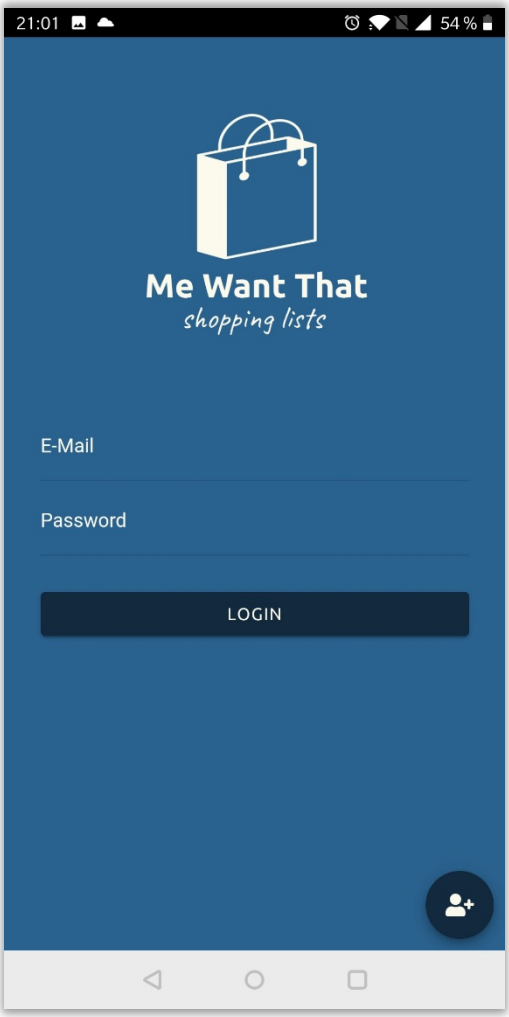
7.1.2.3 Github Repository

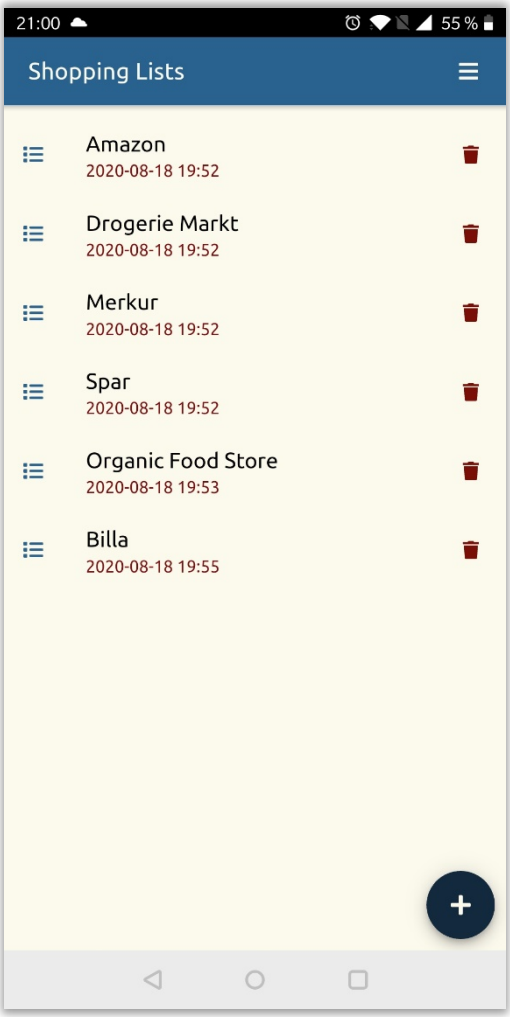


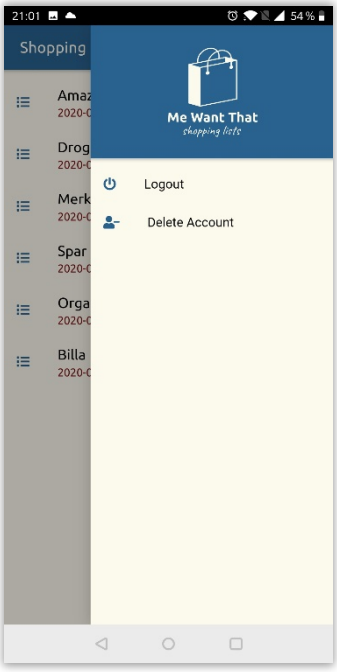
Der gesamte Sourcecode ist öffentlich unter folgendem GitHub Repository verfügbar:

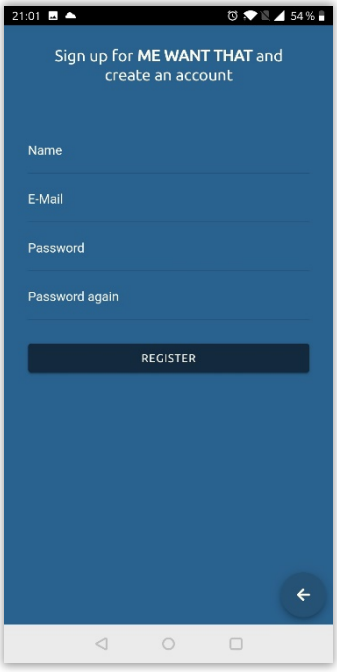
https://github.com/MikesDevCorner/MeWantThat-frontend-ionic.git

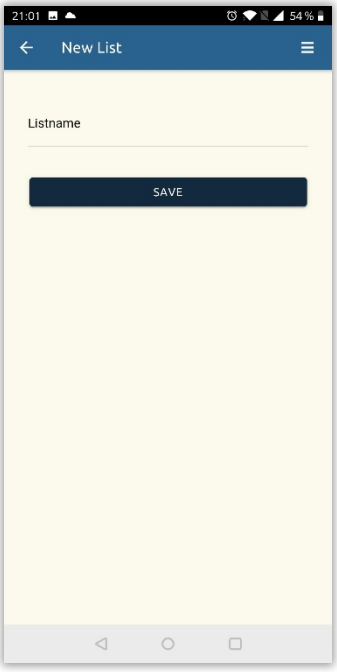
7.1.2.4 Screenshots der IONIC App

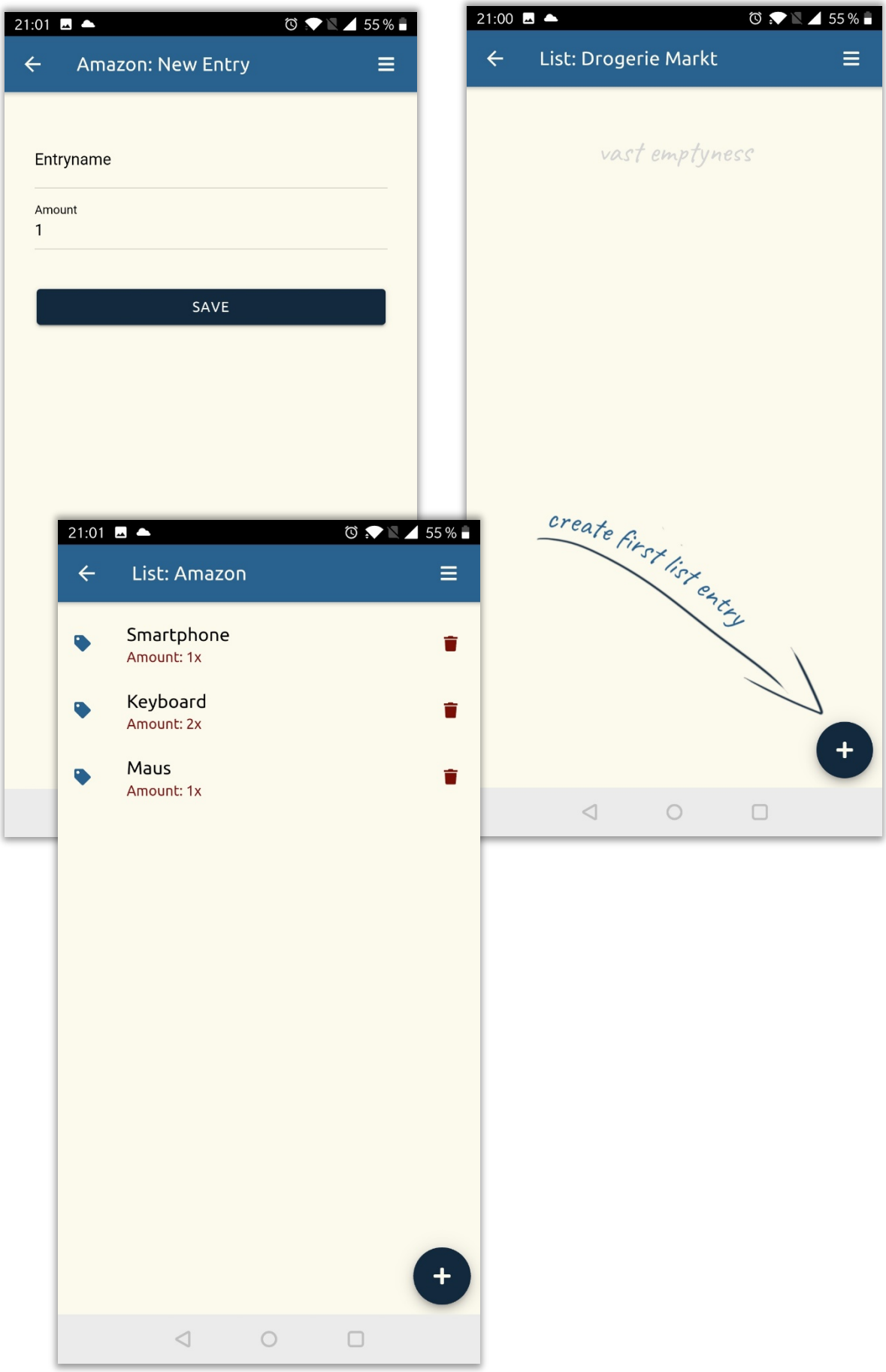











21:01
55 %
Amazon: New Entry
Entryname
Amount
1
SAVE
21:00
55 %
List: Drogerie Markt
vast emptyness
create first list entry
21:01
55 %
List: Amazon
Smartphone
Amount: 1x
Keyboard
Amount: 2x
Maus
Amount: 1x

7.1.2.5 Command Line Tools

Das IONIC Umfeld offeriert ein großartiges Command Line Interface, über welches der gesamte App-Lifecycle bewerkstelligt werden kann. Vom Anlegen der App, über ein direktes Ausliefern auf einen lokalen Webserver für Debug Zwecke (mit Hot Reload Fähigkeit, das bedeutet die App wird sofort neu geladen sobald man im Code etwas ändert), bis zum Builden erlaubt das CLI alle Facetten.

Hier einige der wichtigsten Commands aufgelistet. Eine umfassende Erklärung würde den Rahmen dieser Arbeit sprengen, allerdings sind die Befehle sehr sprechend:

```
npm install -g @vue/cli
```

```
npm install -g ionic@latest
```

```
vue create einkaufsliste-ionic
```

```
npm add @ionic/vue@0.0.9
```

```
npm add @ionic/core
```

```
npm add vue-router
```

```
npm run serve
```

```
npm install --save @capacitor/core @capacitor/cli
```

```
npx cap init --web-dir=dist
```

```
npm run build
```

```
npx cap add android
```

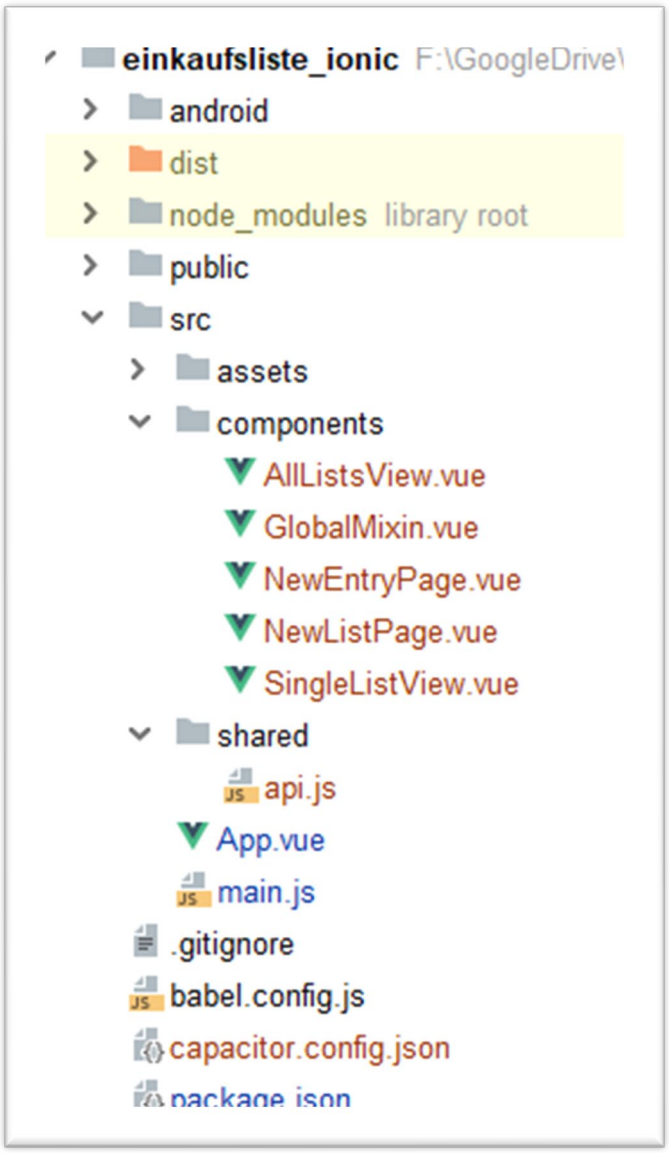
```
npx cap copy
```

```
npx cap open android
```

```
npm run build; npx cap sync
```

7.1.2.6 Ordnerstruktur

Erstellt man ein neues IONIC 5 Projekt, findet man eine vergleichbare Ordnerstruktur wie auf der folgenden Abbildung wieder:



Der Großteil der App, also Logik und Komponenten, werden im Ordner **src** angelegt. Außerhalb dieses Ordners befinden sich Konfigurationsdateien und Build-Ergebnisse. Der Ordner „android“ zum Beispiel kann mit dem Android-Studio geöffnet und so zu einem App Paket (APK) verpackt und in den Play Store deployed werden.

7.1.2.7 Implementierung

File: src/main.js

Einstiegspunkt für die Ionic Engine. Einrichten und Konfigurieren von VueJS und aller Libraries die global benötigt werden. Initialisieren des Routers, des Router-Guards, welcher uns zum Login befördert sollten wir nicht angemeldet sein, und der benannten Routes, die zum Navigieren zwischen den Screens notwendig sind.

```
import Vue from 'vue'
import App from './App.vue'

import Ionic from "@ionic/vue"
import "@ionic/core/css/core.css"
import "@ionic/core/css/ionic.bundle.css"
import { IonicVueRouter } from "@ionic/vue"

import { library } from '@fortawesome/fontawesome-svg-core'
import { faPlus, faTrash, faList, faTag, faPowerOff, faUserPlus, faArrowLeft,
         faUserMinus, faBars } from '@fortawesome/free-solid-svg-icons'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import GlobalMixin from './components/GlobalMixin.vue'
import apiLib from "./shared/api.js"
import auth from "./shared/auth.js"

async function initApp() {
  await auth.init()
  Vue.config.productionTip = false
  Vue.use(Ionic)
  Vue.use(IonicVueRouter)
  Vue.mixin(GlobalMixin)
  Vue.use(apiLib)
  library.add(faPlus, faTrash, faList, faTag, faPowerOff, faUserPlus,
  faUserMinus, faArrowLeft, faBars)
  Vue.component('font-awesome-icon', FontAwesomeIcon)

 const router = new IonicVueRouter({
    routes: [
      { path: "/", redirect: "/all-lists" },
      {
        path: "/login",
        name: "login",
        component: () =>
          import(/* webpackChunkName: "login" */ "@/components/LoginView"),
      },
      {
        path: "/all-lists",
        name: "all-lists",
        component: () =>
          import(/* webpackChunkName: "all-lists" */ "@/components/AllListsView"),
      },
```

```
      {
        path: "/new-list",
        name: "new-list",
        component: () =>
          import(/* webpackChunkName: "new-list" */ "@/components/NewListPage"),
      },
      {
        path: "/single-list-view",
        name: "single-list-view",
        props: true,
        component: () =>
          import(/* webpackChunkName: "single-list-view" */ "@/components/SingleListView"),
      },
      {
        path: "/new-entry",
        name: "new-entry",
        props: true,
        component: () =>
          import(/* webpackChunkName: "new-entry" */ "@/components/NewEntryPage"),
      }
    ]
  });

  router.beforeEach(function(to, from, next) {
    if (to.name == 'login') {
      router.app.$root.$emit('menu-off')
      next()
    } else {
      router.app.$root.$emit('menu-on')
      if(auth.getToken() === null) next({ name: 'login' })
      else next()
    }
  });

  new Vue({
    router,
    render: h => h(App),
  }).$mount('#app')

}
initApp()
```

File: src/shared/api.js

API Library zum asynchronen Kontaktieren der REST API

```
import axios from "axios"
import auth from "./auth.js"

const url = "https://me-want-that.com/api"

export default {
  isAuthenticated: false,
  install: function(Vue) {
    Vue.prototype.$api = {
      async login(email, password) {
        let response = null;
        try {
          response = await axios.post(url + "/login", {
            email: email,
```

```
        password: password
      });
      if(response.status === 202) {
        await auth.setToken(response.data.success.token)
      }
      return response
    } catch(e) {
      if(auth.checkUnauthenticated(e)) return e.response
    }
  },
  async logout() {
    let response = await axios.post(url + "/logout")
    auth.setToken(null)
    return response
  },
  async unregister() {
    let response = await axios.post(url + "/unregister")
    auth.setToken(null)
    return response
  },
  async register(name, email, password, password_confirmation) {
    try {
      let response = await axios.post(url + "/register", {
        email: email,
        password: password,
        password_confirmation: password_confirmation,
        name: name
      });
      if(response.status === 201) {
        await auth.setToken(response.data.success.token)
      }
      return response
    } catch(e) {
      if(auth.checkUnauthenticated(e)) return e.response
    }
  },
  async getLists() {
    try {
      let response = await axios.get(url + "/lists")
      return response
    } catch(e) {
      if(auth.checkUnauthenticated(e)) return e.response
    }
  },
  async addList(name) {
    try {
      let response = await axios.post(url + "/list", {
        listname: name
      })
      return response
    } catch(e) {
      if(auth.checkUnauthenticated(e)) return e.response
    }
  },
  async removeList(list) {
    try {
      let response = await axios.delete(url + "/list/" + list.id);
      return Promise.resolve(response)
    } catch(e) {
      if(auth.checkUnauthenticated(e)) return e.response
    }
  },
  async getEntries(listId) {
    try {
      let response = await axios.get(url + "/list/" + listId + "/entries");
      return Promise.resolve(response)
    } catch(e) {
```

```
          if(auth.checkUnauthenticated(e)) return e.response
        }
      },
      async addEntry(name, amount, listId) {
        try {
          let response = await axios.post(url + "/list/"+listId+"/entry", {
            entryname: name,
            amount: amount
          });
          return Promise.resolve(response)
        } catch(e) {
          if(auth.checkUnauthenticated(e)) return e.response
        }
      },
      async removeEntry(entry) {
        try {
          let response = await axios.delete(url + "/entry/" + entry.id)
          return Promise.resolve(response)
        } catch(e) {
          if(auth.checkUnauthenticated(e)) return e.response
        }
      }
    };
  }
};
```

File: src/shared/auth.js

Service-Library, welche die Authentifizierung über Token und Secure Storage verwaltet.

```
import 'capacitor-secure-storage-plugin'
import { Plugins } from '@capacitor/core'
import axios from 'axios'

const { SecureStoragePlugin } = Plugins;

export default {
    _token: null,
    setToken: async function(val) {
        this._token = val
        await SecureStoragePlugin.set({
            key: 'token',
            value: val
        })
        if(val !== null) axios.defaults.headers.common = {'Authorization': `Bearer ${val}`,
 'Accept': 'application/json'}
        else axios.defaults.headers.common = {'Accept': 'application/json'}
    },
    getToken: function() {
        return this._token
    },
    init: async function() {
        let t = {value: null};
        try {
            t = await SecureStoragePlugin.get({ key: 'token'})
```

```
        } catch(e) {
            console.log('no token read from storage')
        }
        if(t.value !== null && t.value !== 'null' && t.value !== undefined
            && t.value !== 'undefined') {
            this._token = t.value
            axios.defaults.headers.common = {'Authorization': `Bearer ${t.value}`,
                                             'Accept': 'application/json'}
        }
    },
    checkUnauthenticated(e) { //Error goes in here. Checks if Unauthenticated or
                              other error
        if(e.response && e.response.status === 401) {
            this.setToken(null)
            return e.response
        } else throw e
    }
}
```

File: src/components/GlobalMixin.vue

Mixin, um diverse Funktionen wie Alerts, Snackbars und Loadingindikatoren in allen Komponenten verfügbar zu machen.

```
<script>
export default {
  data() {
    return {
      loading: false,
      loadScreen: null
    };
  },
  methods: {
    showLoading() {
      let me = this;
      if (this.loading !== true) {
        this.loading = true;
        return this.$ionic.loadingController
          .create({
            showBackdrop: false,
            translucent: true,
            cssClass: 'my-loading-class'
          })
          .then(loadScreen => {
            me.loadScreen = loadScreen;
            return loadScreen.present();
          });
      }
    },
    hideLoading() {
      this.loading = false;
      this.loadScreen.dismiss();
    },
    presentAlert(header, message, subtitle, buttons) {
      return this.$ionic.alertController
        .create({
          header: header,
          subHeader: subtitle !== undefined ? subtitle : "",
```

```
          message: message,
          buttons: buttons !== undefined ? buttons : ["OK"]
        })
        .then(a => a.present());
    },
    presentToast(message) {
      return this.$ionic.toastController
        .create({
          message: message,
          duration: 2000
        })
        .then(a => a.present());
    }
  }
};
</script>
```

File: src/components/AllLists.vue

Screen zur Auflistung aller Einkaufslisten

```
<template>
  <ion-page>
    <ion-header>
      <ion-toolbar color="primary">
        <ion-buttons slot="end">
          <ion-menu-button menu="first" >
            <font-awesome-icon class="fa-xs" icon="bars"/>
          </ion-menu-button>
        </ion-buttons>
        <ion-title>Shopping Lists</ion-title>
      </ion-toolbar>
    </ion-header>
    <ion-content padding>
      <div v-if="lists.length === 0" class="img-cont">
        <div class="empty handwritten">vast emptyness</div>
        <img alt="Hint" src="../assets/first_list_hint.png" class="hint">
      </div>
      <ion-list v-if="lists.length > 0">
        <ion-item button v-for="(list, index) in lists" :key="index">
          <ion-ripple-effect></ion-ripple-effect>
          <font-awesome-icon class="text-color-primary list-icon" icon="list" />
          <ion-label @click="openList(list)">
            <h2>{{list.listname}}</h2>
            <ion-note>{{list.readableDate}}</ion-note>
          </ion-label>
          <ion-button fill="clear" size="large" slot="end" @click="removeList(list)">
            <font-awesome-icon class="fa-xs text-alt-color" icon="trash" />
          </ion-button>
        </ion-item>
      </ion-list>
      <ion-fab vertical="bottom" horizontal="end" slot="fixed">
        <ion-fab-button @click="addList">
          <font-awesome-icon class="fa-lg" icon="plus" />
        </ion-fab-button>
      </ion-fab>
    </ion-content>
  </ion-page>
</template>
```

```
<script>
import { Plugins } from '@capacitor/core'
import moment from 'moment'

export default {
  name: "AllListsView",
  mounted() {
    this.$root.$emit('menu-on')
    this.refreshList();
    if(this.alreadySubscribed === false) {
      Plugins.App.addListener('backButton', function() {
        Plugins.App.exitApp();
      });
      this.alreadySubscribed = true;
    }
  },
  data() {
    return {
      lists: [],
      alreadySubscribed: false
    };
  },
  methods: {
    async refreshList() {
      this.showLoading();
      let response = await this.$api.getLists();
      if(response.status === 401) {
        this.hideLoading();
        this.$router.replace({name: "login"});
      }
      this.lists = response.data;
      this.lists.forEach((list) => {
          list.readableDate = moment(list.updated_at).format("YYYY-MM-DD HH:mm")
      })
      this.hideLoading();
    },
    openList(list) {
      this.$router.push({
        name: "single-list-view",
        params: { listid: list.id, title: list.listname }
      });
    },
    addList() {
      this.$router.push({ name: "new-list" });
    },
    async removeList(list) {
      this.showLoading();
      let response = await this.$api.removeList(list);
      if (response.status === 200) {
        this.refreshList();
        this.presentToast("Deleted successfully");
      } else if(response.status === 401) {
        this.hideLoading();
        this.$router.replace({name: "login"});
      } else {
        this.hideLoading();
        this.presentAlert(
          "Error",
          "Some error occured during deletion"
        );
      }
    }
  }
};
</script>
```

```
<style scoped>
  .sc-ion-label-md-s h2 {
    font-size: 18px;
    font-weight: 500;
  }

  ion-list ion-button {
    --padding-end: 5px;
  }

  .img-cont {
    position:relative;
    height: 100%;
  }

  .empty {
    text-align: center;
    font-size: 30px;
    padding: 1.5em;
    color: #d5d5d5;
  }

  .list-icon {
    margin-right: 35px;
  }

  .hint {
    display: block;
    max-height: 40%;
    position: absolute;
    bottom: 50px;
    right: 30px;
  }
</style>
```

File: src/components/LoginView.vue

Kombinations-Screen für Login und Registrierung

```
<template>
  <ion-page>
    <ion-content color="primary">
      <ion-grid>
        <ion-row color="primary" class="ion-justify-content-center">
          <ion-col align-self-center size-md="6" size-lg="5" size-xs="12">
            <div class="ion-text-center hl">
              <img v-if="register===false" alt="Logo" src="../assets/logosm.png"
                        style="width: 60%; margin-top: 40px;">
              <h4 v-if="register===true">Sign up for <b>ME WANT THAT</b>
                                        and create an account</h4>
            </div>
            <div class="ion-padding" style="margin: 4px;">
              <ion-item v-if="register">
                <ion-label position="floating">Name</ion-label>
                <ion-input type="text"
                           :value="name"
                           inputmode="text"
                           @input="name=$event.target.value"></ion-input>
              </ion-item>
              <ion-item>
```

```
                <ion-label position="floating">E-Mail</ion-label>
                <ion-input type="email"
                           :value="email"
                           inputmode="email"
                           @input="email=$event.target.value"></ion-input>
              </ion-item>
              <ion-item>
                <ion-label position="floating">Password</ion-label>
                <ion-input type="password"
                           :value="password"
                           inputmode="text"
                           @input="password=$event.target.value"></ion-input>
              </ion-item>
              <ion-item v-if="register">
                <ion-label position="floating">Password again</ion-label>
                <ion-input type="password"
                           :value="passwordconfirm"
                           inputmode="text"
                           @input="passwordconfirm=$event.target.value"></ion-input>
              </ion-item>
              <div class="lbutton">
                <ion-button expand="block" @click="login">
                                {{ register ? 'Register' : 'Login' }}</ion-button>
              </div></div>
          </ion-col>
        </ion-row>
      </ion-grid>
      <ion-fab vertical="bottom" horizontal="end" slot="fixed">
        <ion-fab-button @click="changeToRegister">
          <font-awesome-icon class="fa-lg" v-if="register" icon="arrow-left" />
          <font-awesome-icon class="fa-lg" v-if="!register" icon="user-plus" />
        </ion-fab-button>
      </ion-fab>
    </ion-content>
  </ion-page>
</template>

<script>
export default {
  name: "LoginView",
  data() {
    return {
      register: false,
      email: "",
      password: "",
      passwordconfirm: "",
      name: ""
    };
  },
  methods: {
    changeToRegister() { this.register = !this.register },
    async login() {
      this.email = this.email.trim();
      if (this.email !== "" && this.password !== "") {
        this.showLoading();
        try {
          let response = null
          if(this.register === true) {
            response = await this.$api.register(
              this.name,
              this.email,
              this.password,
              this.passwordconfirm
```

```
            )
          } else {
            response = await this.$api.login(
              this.email,
              this.password
            )
          }
          if (response.status === 202) {
            this.$router.replace({name: "all-lists"});
            this.presentToast("Login successful")
          } else if(response.status === 201) {
            this.$router.replace({name: "all-lists"});
            this.presentToast("Registered successful and logged in")
          } else if(response.status === 401) {
            this.presentToast("Please check E-Mail and Password and try again");
          }
          this.hideLoading()
        } catch(e) {
          this.hideLoading();
          this.presentToast("Something went wrong on contacting the API: "+e.message);
        }
      } else {
        this.presentToast("Please check your input fields")
      }
    },
    goBack() {
      this.$router.replace({
        name: "single-list-view",
        params: { listid: this.listid, title: this.title }
      });
    }
  }
};
</script>

<style scoped>
  ion-content{
    --ion-background-color:#2a628f;
  }

  .lbutton {
    margin: -2px;
    margin-top: 30px;
  }

  ion-button {
    --background: #13293d;
  }

  ion-item {
      margin-bottom: 5px;
      --background: #transparent;
      --color: #fcf9ed;
      --highlight-color-focused : #13293d;
      --padding-end: 0px;
      --padding-start: 0px;
      --highlight-height: 1px;
      /*--show-full-highlight: true;*/
  }

  .hl {
    margin-bottom: 40px;
    padding: 0px 32px;
  }

  .hl > h1 {
    font-size: 43px;
```

```
    font-weight: normal;
    text-align: center;
    margin-top: 40px;
  }

  .hl > h3 {
    font-size: 30px;
    font-weight: normal;
    text-align: left;
  }
</style>
```

File: src/components/NewListPage.vue

Screen zum Anlegen einer neuen Liste

```
<template>
  <ion-page>
    <ion-header>
      <ion-toolbar color="primary">
        <ion-buttons slot="start">
          <ion-back-button default-href="/"></ion-back-button>
        </ion-buttons>
        <ion-buttons slot="end">
          <ion-menu-button menu="first" >
            <font-awesome-icon class="fa-xs" icon="bars"/>
          </ion-menu-button>
        </ion-buttons>
        <ion-title>New List</ion-title>
      </ion-toolbar>
    </ion-header>
    <ion-content>
      <div class="gpadding">
        <ion-item class="nameinput">
          <ion-label position="floating">Listname</ion-label>
          <ion-input type="text"
                     :value="listName"
                     maxlength="60"
                     required
                     @input="listName=$event.target.value"></ion-input>
        </ion-item>
        <div class="icenter">
          <ion-button color="tertiary" expand="block" @click="saveList">Save</ion-button>
        </div>
      </div>
    </ion-content>
  </ion-page>
</template>

<script>
export default {
  name: "NewListPage",
  data() {
    return {
      listName: ""
    };
  },
  mounted() {
    this.$root.$emit('menu-on')
  },
```

```
  methods: {
    async saveList() {
      this.listName = this.listName.trim();
      if (this.listName !== "") {
        this.showLoading();
        try {
          let response = await this.$api.addList(this.listName);
          this.hideLoading();
          if (response.status === 201) {
            this.$router.replace({ name: "all-lists" });
            this.presentToast("List created successfully");
          } else if(response.status === 401) {
            this.hideLoading();
            this.$router.replace({name: "login"});
          }
        } catch(e) {
          this.hideLoading();
          this.presentAlert(
            "Error",
            "Some error occured during creation of your list"
          );
        }
      } else {
        this.presentToast("Listname should not be empty");
      }
    }
  }
};
</script>

<style scoped>
  .nameinput {
    margin-top: 40px;
    margin-bottom: 20px;
  }
  .icenter {
    padding-top: 15px;
  }

  .gpadding {
    margin: 30px;
  }

  ion-item {
      margin-bottom: 5px;
      --padding-end: 0px;
      --padding-start: 0px;
      --highlight-height: 1px;
  }
</style>
```

File: src/components/SingleListView.vue

Screen zum Anzeigen aller Posten auf einer einzelnen Liste

```
<template>
  <ion-page>
    <ion-header>
      <ion-toolbar color="primary">
        <ion-buttons slot="start">
          <ion-back-button default-href="/"></ion-back-button>
```

```
        </ion-buttons>
        <ion-buttons slot="end">
          <ion-menu-button menu="first" >
            <font-awesome-icon class="fa-xs" icon="bars"/>
          </ion-menu-button>
        </ion-buttons>
        <ion-title>List: {{title}}</ion-title>
      </ion-toolbar>
    </ion-header>
    <ion-content padding>
      <div v-if="entries.length === 0" class="img-cont">
        <div class="handwritten empty">vast emptyness</div>
        <img alt="Hint" src="../assets/first_entry_hint.png" class="hint">
      </div>
      <ion-list v-if="entries.length > 0">
        <ion-item v-for="(entry, index) in entries" :key="index">
          <font-awesome-icon class="text-color-primary entry-icon" icon="tag"/>
          <ion-label>
            <h2>{{entry.entryname}}</h2>
            <ion-note>Amount: {{entry.amount}}x</ion-note>
          </ion-label>
          <ion-button  fill="clear" size="large" slot="end" @click="removeEntry(entry)">
            <font-awesome-icon class="fa-xs text-alt-color" icon="trash" />
          </ion-button>
        </ion-item>
      </ion-list>
      <ion-fab vertical="bottom" horizontal="end" slot="fixed">
        <ion-fab-button @click="addEntry">
          <font-awesome-icon class="fa-lg" icon="plus" />
        </ion-fab-button>
      </ion-fab>
    </ion-content>
  </ion-page>
</template>

<script>
import moment from 'moment'

export default {
  name: "SingleListView",
  mounted() {
    this.$root.$emit('menu-on')
    this.refreshEntries();
  },
  data() {
    return {
      entries: []
    };
  },
  props: ["listid", "title"],
  methods: {
    async refreshEntries() {
      this.showLoading();
      let response = await this.$api.getEntries(this.listid);
      if(response.status === 401) {
        this.hideLoading();
        this.$router.replace({name: "login"});
      }
      this.entries = response.data;
      this.entries.forEach((entry) => {
          entry.readableDate = moment(entry.updated_at).format("YYYY-MM-DD HH:mm")
      })
      this.hideLoading();
    },
    async removeEntry(entry) {
      this.showLoading();
      let response = await this.$api.removeEntry(entry);
```

```
      if (response.status === 200) {
        this.refreshEntries();
        this.presentToast("Deleted successfully.");
      } else if(response.status === 401) {
        this.hideLoading();
        this.$router.replace({name: "login"});
      } else {
        this.hideLoading();
        this.presentAlert(
          "Error",
          "Some error occured during deletion"
        );
      }
    },
    addEntry() {
      this.$router.replace({
        name: "new-entry",
        params: { listid: this.listid, title: this.title }
      });
    }
  }
};
</script>

<style scoped>
  .sc-ion-label-md-s h2 {
    font-size: 18px;
    font-weight: 500;
  }

  ion-list ion-button {
    --padding-end: 5px;
  }

  .img-cont {
    position:relative;
    height: 100%;
  }

  .empty {
    text-align: center;
    font-size: 30px;
    padding: 1.5em;
    color: #d5d5d5;
  }

  .entry-icon {
    margin-right: 35px;
  }

  .hint {
    display: block;
    max-height: 40%;
    position: absolute;
    bottom: 50px;
    right: 30px;
  }
</style>
```

File: src/components/NewEntryPage.vue

Screen zum Anlegen eines neuen Listenpostens

```
<template>
  <ion-page>
    <ion-header>
      <ion-toolbar color="primary">
        <ion-buttons slot="start">
          <ion-back-button @click.stop.prevent="goBack"></ion-back-button>
        </ion-buttons>
        <ion-buttons slot="end">
          <ion-menu-button menu="first" >
            <font-awesome-icon class="fa-xs" icon="bars"/>
          </ion-menu-button>
        </ion-buttons>
        <ion-title>{{title}}: New Entry</ion-title>
      </ion-toolbar>
    </ion-header>
    <ion-content>
      <div class="gpadding">
        <ion-item class="nameinput">
          <ion-label position="floating">Entryname</ion-label>
          <ion-input type="text"
                     :value="entryName"
                     required
                     maxlength="60"
                     @input="entryName=$event.target.value"></ion-input>
        </ion-item>
        <ion-item class="amountinput">
          <ion-label position="floating">Amount</ion-label>
          <ion-input type="number"
                     required
                     min="0"
                     max="9999"
                     :value="amount"
                     @input="amount=$event.target.value"></ion-input>
        </ion-item>
        <div class="icenter">
          <ion-button color="tertiary" expand="block" @click="saveEntry">Save</ion-button>
        </div>
      </div>
    </ion-content>
  </ion-page>
</template>

<script>
export default {
  name: "NewEntryPage",
  props: ["listid", "title"],
  data() {
    return {
      entryName: "",
      amount: 1
    };
  },
  mounted() {
    this.$root.$emit('menu-on')
  },
```

```
  methods: {
    async saveEntry() {
      this.entryName = this.entryName.trim();
      if (this.entryName !== "" && this.amount !== "" && this.amount > 0
          && this.amount < 9999) {
        this.showLoading();
        try {
          let response = await this.$api.addEntry(
            this.entryName,
            this.amount,
            this.listid
          );
          if (response.status === 201) {
            this.hideLoading();
            this.$router.replace({
              name: "single-list-view",
              params: { listid: this.listid, title: this.title }
            });
            this.presentToast("Entry created successfully");
          } else if(response.status === 401) {
            this.hideLoading();
            this.$router.replace({name: "login"});
          }
        } catch(e) {
          this.hideLoading();
          this.presentAlert(
            "Error",
            "There was an error creating your entry"
          );
        }
      } else {
        this.presentToast("Please check your input fields: entryname should be shorter"
          + "than 60 characters. amount should be more than 0 and less than 10000.", 6000);
      }
    },
    goBack() {
      this.$router.replace({
        name: "single-list-view",
        params: { listid: this.listid, title: this.title }
      });
    }
  }
};
</script>

<style scoped>
.nameinput {
  margin-top: 40px;
}
.amountinput {
  margin-bottom: 20px;
}

.icenter { padding-top: 15px;}
.gpadding {   margin: 30px;> }

ion-item {
    margin-bottom: 5px;
    --padding-end: 0px;
    --padding-start: 0px;
    --highlight-height: 1px;
}
</style>
```

File: src/App.vue

Vue-Einstiegspunkt sowie Styling und Theming für die App

```
<template>
  <div id="app">
    <ion-app>
      <ion-menu :disabled="menuDisabled" side="end" menu-id="first"
                type="overlay" content-id="thepage" ref="menu">
        <ion-header>
          <ion-toolbar color="primary" style="text-align:center; padding: 20px 0;">
            <img alt="Logo" src="./assets/logosm.png" style="width: 40%;">
            </ion-toolbar>
        </ion-header>
        <ion-content>
            <ion-list>
              <ion-item @click="logout">
                <font-awesome-icon class="text-color-primary"
                                   icon="power-off" style="margin-right: 35px;" />
                <ion-label>Logout</ion-label>
              </ion-item>
              <ion-item @click="deleteAccount">
                <font-awesome-icon class="text-color-primary"
                                   icon="user-minus" style="margin-right: 35px;" />
                <ion-label>Delete Account</ion-label>
              </ion-item>
            </ion-list>
        </ion-content>
      </ion-menu>
      <div id="thepage">
        <ion-vue-router />
      </div>
    </ion-app>
  </div>
</template>

<script>
export default {
  name: "app",
  data: function () {
    return {
      menuDisabled: true
    }
  },
  created() {
    this.$root.$on('menu-on', () => {
      this.menuDisabled = false;
    })
    this.$root.$on('menu-off', () => {
      this.menuDisabled = true;
    })
  },
  methods: {
    async logout() {
      this.$refs.menu.close();
      await this.$api.logout();
      this.$router.replace({name: "login"});
    },
    async deleteAccount() {
```

```
      this.presentAlert(
        "Unregister",
        "Would you really like to delete your account and all its data permanently
         from our databases? You cannot undo this action.",
        undefined, [
          {   text: 'Cancel',
              role: 'cancel',
              handler: () => { },
          },{
              text: 'Delete account',
              handler: async () => {
                this.$refs.menu.close();
                await this.$api.unregister();
                this.$router.replace({name: "login"});
              }
            }
        ]
      )
    }
  }
};
</script>

<style>

  @font-face {
    font-family: 'Ubuntu';
    src: url('assets/Ubuntu-Regular.ttf');
  }

  @font-face {
    font-family: 'Caveat';
    src: url('assets/Caveat-Regular.ttf');
  }

  * {
    font-family: 'Ubuntu';
  }

  .handwritten {
    font-family: 'Caveat';
  }

  #thepage {height: 100%;}

  .mytoast {
    opacity: 0.9;
    --end: 75px;
  }

  ion-title {
    font-size: 19px;
  }

  ion-item {
    --inner-border-width: 0;
  }

  ion-fab-button{
      --background: #13293D;
  }

  ion-loading.my-loading-class .loading-wrapper {
    background: transparent;
    box-shadow: none;
  }
```

```
  ion-note {
    color: #7A0F0F;
  }

  .text-alt-color {
    color: #7a0f07;
  }

  .text-color-primary {
    color: #2A628F;
  }

  ion-header ion-menu-button {
    --padding-end: 15px;
  }

  :root {
  --ion-color-primary: #2A628F;
  --ion-color-primary-rgb: 42,98,143;
  --ion-color-primary-contrast: #fcf9ed;
  --ion-color-primary-contrast-rgb: 252,249,237;
  --ion-color-primary-shade: #3679ae;
  --ion-color-primary-tint: #265276;
  --ion-color-secondary: #FCF9ED;
  --ion-color-tertiary: #13293D;
  --ion-color-light: #3B2D29;
  --ion-color-medium: #7A0F0F;
  --ion-background-color:#fcf9ed;
  }
</style>
```

File: package.json

Konfigurationsdatei mit Paket-Abhängigkeiten

```
{
  "name": "me-want-that",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
    "@capacitor/android": "^2.4.0",
    "@capacitor/cli": "^2.4.0",
    "@capacitor/core": "^2.4.0",
    "@fortawesome/fontawesome-svg-core": "^1.2.30",
    "@fortawesome/free-solid-svg-icons": "^5.14.0",
    "@fortawesome/vue-fontawesome": "^0.1.10",
    "@ionic/core": "^5.3.1",
    "@ionic/vue": "^0.0.9",
    "axios": "^0.19.2",
    "capacitor-secure-storage-plugin": "^0.4.0",
    "core-js": "^3.6.5",
    "moment": "^2.27.0",
    "vue": "^2.6.11",
    "vue-router": "^3.4.3"
  },
```

```
  "devDependencies": {
    "@vue/cli-plugin-babel": "^4.2.3",
    "@vue/cli-plugin-eslint": "^4.2.3",
    "@vue/cli-service": "^4.2.3",
    "babel-eslint": "^10.1.0",
    "eslint": "^6.7.2",
    "eslint-plugin-vue": "^6.2.2",
    "vue-template-compiler": "^2.6.11"
  },
  "eslintConfig": {
    "root": true,
    "env": {
      "node": true
    },
    "extends": [
      "plugin:vue/essential",
      "eslint:recommended"
    ],
    "parserOptions": {
      "parser": "babel-eslint"
    },
    "rules": {}
  },
  "browserslist": [
    "> 1%",
    "last 2 versions"
  ]
}
```

Auch hier wurde die Implementierung einfach gehalten, um die Sourcecode-Abschnitte kurz und verständlich zu belassen. Den Einstiegspunkt in die App stellt das **main.js** File dar, wo die globalen Abhängigkeiten für die Verwendung mit Vue registriert werden. Auch die nötigen Routen zur Navigation werden hier festgelegt.

Wie auch bei Flutter gibt es eine Service-Library **api.js**. Sie kümmert sich um den asynchronen Austausch der Daten mit der Server Rest API.

Zusätzlich wurde ein globales Mixin für Vue geschaffen. Mixins sind Snippets, welche in allen Komponenten zur Verfügung stehen und bieten somit eine gute Möglichkeit für geteilte Funktionalität an. Das **GlobalMixin.vue** stellt den Komponenten eine Loadingmaske und Dialogmöglichkeiten zur Verfügung.

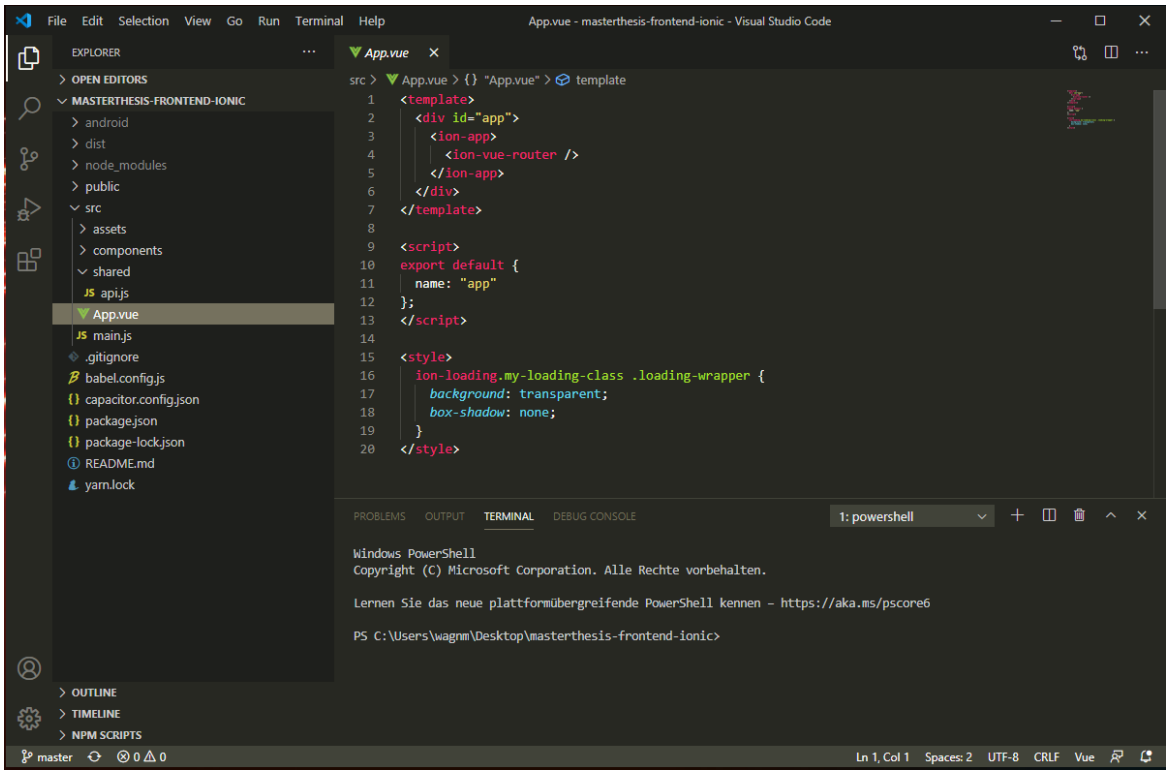
Die fünf Screens sind je als einzelne Vue Komponenten realisiert, welche ihren State, also die Daten, selbstständig verwalten und mithilfe der Service API-Bibliothek abrufen. Eine Navigation von Screen zu Screen wird mit dem Vue Router bewerkstelligt.

7.1.2.8 Accessories und Hilfesmittel

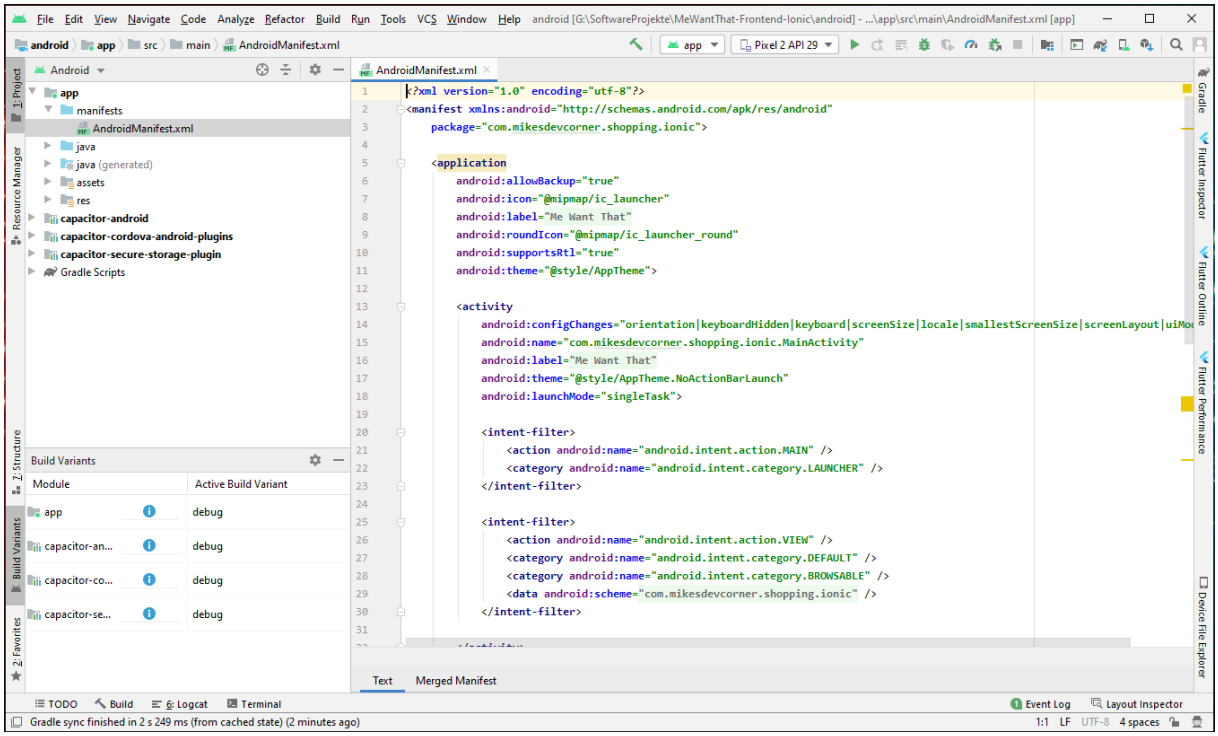
Ionic bietet, wie bereits erwähnt ab Version 5 die Auswahl für *Angular*, *React* oder *VueJs* an. Letzteres befindet sich zum Zeitpunkt der Veröffentlichung dieser Arbeit im Beta Stadium und ist noch nicht besonders gut dokumentiert. Die Implementierung mit Vue fühlte sich trotzdem angenehm an, da Vue unter den zur Verfügung stehenden Kandidaten über die geringste Komplixität und Lernkurve verfügt.

Die integrierte Komponenten-Library ist ausreichend gut dokumentiert und lässt kaum Wünsche offen, und sollte doch ein Sonderwunsch auftreten kann man über den ***IONIC Marketplace*** von Usern designte Komponenten downloaden. Dort gibt es auch fertige Layouts und Screens zum Download.

Das wichtigste Tool für die Programmierung mit IONIC stellt mit Sicherheit die *Commandline* dar. Ein guter Editor, oder eine gute Web-IDE mit Plugin-Unterstützung für VueJS und IONIC (wie ***Visual Studio Code*** oder ***JetBrains WebStorm***) erleichtern das Arbeiten.



Anders als bei *Flutter*, welches primär in der IDE Android Studio programmiert wird, wird im Falle von Ionic mit dem Tool „Capacitor“ ein Android Studio bzw xCode Projekt generiert, welche danach für die plattformspezifischen Buildvorgänge verwendet werden.



7.1.2.9 Paketverwaltung

Da das Ionic Framework eine Web-App Lösung ist und die Ressourcen nicht zu Binärcode kompiliert werden müssen, ist ein Einbinden aller Plugins möglich, die auch im Web funktionieren. Diese Tatsache multipliziert die Vielfalt der verfügbaren Lösungen um ein Vielfaches, da für Web Apps eine Unzahl an Code Snippets existiert.

Die verwendete Programmiersprache ist Javascript und die Anwendung wird im Kontext einer WebView auf mobilen Endgeräten ausgeführt. Das erlaubt unter anderem das im Webbereich stark verbreitete Paketmanagement Tool NPM (Node Package Manager) und seine unbegrenzt erscheinende Anzahl an Lösungen, Paketen und Tools zu verwenden.

Ionic bietet mit seinen hauseigenen Userinterface Elementen, welche die meisten aktuell bekannten UI Widgets und Bedienelemente umfasst, bereits einen großartigen Startpunkt für kreatives App-Design.

Trotzdem gibt es unter market.ionicframework.com auch einen prall gefüllten Marketplace, auf dem Entwickler ihre eigenen Lösungen anbieten.

Um aus dem Rahmen einer Webapp auszubrechen und Funktionen zu erlauben, die im Normalfall nur nativen Apps erlaubt sind, verwendet man Plugins für Apache Cordova beziehungsweise das neuere Tool Capacitor. Diese Plugins injizieren Funktionen wie Push Notifications, Taschenlampe, Screenshots und viele mehr in die WebApp.

7.1.2.10 Testimonials

Ein populäres App-Framework definiert sich nicht zuletzt über die dadurch entstandenen, bekannten und prominenten Apps, die in den Stores dieser Welt verfügbar sind. Im Jahr 2019 wurden mit dem IONIC Framework unter anderem folgende großartige Apps entwickelt:

- IBM
- ING
- SAP
- NASA
- Sworkit
- JustWatch
- McDonalds Türkiye
- McLaren Automotive
- Honeyfi
- Diesel

8 Zusammenfassung und Beurteilung

Die Forschungsfrage beantwortet sich also wie folgt:

Um eine App mit zentralisierter Datenverwaltung und Persistierung umzusetzen, benötigt man ein weitreichendes Vermögen über verschiedene Themenbereiche der Softwareentwicklung.

Die Kernaspekte dabei stellen folgende Gebiete dar:

- Wissen über Datenmodellierung und Datenbankdesign
- Wissen über Backend Development und die verfügbaren Frameworks und Optionen
- Wissen über REST API-Design
- Wissen über Security und Authentifizierung
- Wissen über Frontend Development und die verfügbaren Frameworks und Optionen
- Wissen über asynchronen Datenaustausch zwischen Frontend und Backend
- Verständnis für Design, Datenvisualisierung, User Interfaces und Usability

Im direkten Vergleich der Cross Platform Development Paradigmen **Native Hybrid App** (repräsentiert durch *Flutter*) mit **Web Hybrid App** (repräsentiert durch *Ionic*) sind mehrere Kriterien zu berücksichtigen. Bevor die herausgearbeiteten Unterschiede aufgezeigt werden, muss erwähnt werden, dass es keinen „Verlierer" und „Gewinner" dieser Ausarbeitung gibt.

Beide Frameworks haben dieselbe Vision, allerdings könnten die Ansätze und Philosophien dahinter oft nicht unterschiedlicher sein. Während Ionic auf offene Standards und Web Plattformen setzt, entscheidet sich Flutter für ein in sich geschlossenes Ecosystem und neuen Wegen zu Gunsten von Performance.

Je nach Projektanforderung muss anhand der bekannten Parameter aufs Neue entschieden werden, welches Framework zu bevorzugen ist. Die Erfahrung zeigt, dass oft auch der Kunde bereits eine Wunschvorstellung über die verwendete Technologie ins Projekt mitbringt. Am Ende des Tages muss man sein Ziel kennen und die möglichen Wege, dorthin zu gelangen. Welcher Ansatz der *„richtige"* ist, entscheidet sich von Projekt zu Projekt. Oft wird es vermutlich auch weiterhin eine nativ programmierte App sein.

Die Implementierung der Beispiel-App „Shopping List" stellte sich mit beiden verwendeten Frameworks als unproblematisch dar.

Stellt man die Frameworks Gegenüber, sind folgende Aspekte aufgefallen:

- Die **Ausführungsgeschwindigkeit** der Flutter Applikation ist wesentlich höher gegenüber Ionic. Der Unterschied ist im Vergleich deutlich spürbar, obwohl Ionic keine *„schlechte"* Performance bietet, allerdings eine schlechtere. Das Ahead-Of-Time Kompilieren bei Flutter macht sich in diesem Punkt bezahlt.

- Die **Startzeit**, also die Zeit, die vom Tappen des Icons bis zum Verschwinden des Splash-Screens vergeht, ist bei Ionic wesentlich höher. Nach dem Öffnen wartet man hier einige Sekunden bis der Splash Screen verschwindet währen bei Flutter ohne Verzögerung die App geöffnet ist. Auch hier macht sich das Interpretieren der App durch eine *WebView* negativ bemerkbar.

- Vergleicht man die **UI Komponenten-Libraries** mitsammen, finde ich die Flutter Komponenten mit Material bzw. Cupertino Design schöner gestaltet und sie wirken reaktiver und dynamischer.

- Das Aufbauen, Ineinander Verschachteln und Layouten der Screens und Komponenten funktioniert bei Ionic durch HTML viel einfacher und unkomplizierter. Außerdem ist eine saubere Trennung (bei Verwendung von Vue) zwischen Präsentationsschicht (HTML, CSS) und Programmlogik (JS, Vue) möglich. Flutter hingegen wirkt hier umständlich, verzweigt sich der Komponentenbaum bereits bei einfachen Hierarchien stark und macht ein sauberes Lesen der Parent-Child Beziehungen in der klammerintensiven C-Syntax schwer.

- Da Flutter seine eigene Runtime, also die in C++ geschriebene Engine, welche unsere vorkompilierte Application Library zur Laufzeit ausführt, mit in die Projekte verpackt, ist die **Größe des gepackten Builds** deutlich höher als bei Ionic. Ionic verpackt zwar einen dünnen Layer über das Tool Capacitor, um der WebView den Zugriff auf die Betriebssytem-APIs zu gewährleisten, das benötigt allerdings deutlich weniger Speicherplatz. Im direkten Verlgeich anhand unserer Beispiel-App war Ionic mit knapp 5Mb nur ein Drittel so groß wie das APK File, welches der Flutter Buildprozess ausspuckte mit rund 15Mb.

- Der **Entwicklungsprozess** war in beiden Fällen ausgezeichnet. Beide Frameworks bieten einen lokalen Development Server an, welcher die App

zur Laufzeit im Browser anzeigt und über eine Hot-Reload Fähigkeit verfügt. Beide Apps können auch entwickelt werden, während sie auf dem Emulator oder einem echten Device ausgeführt werden, auch dort wird hot reloaded.

- **Debugging und Fehlersuche** war, trotz der guten DevTools von Flutter, mit Ionic einfacher. Die Chrome Browser DevTools können sich remote mit dem Android-Gerät, dass die App betreibt verbinden und so können nicht nur Haltepunkte ausgezeichnet und im Livebetrieb untersucht, sondern auch der konkrete Zustand des DOMs und der Komponentenbaum betrachtet werden. Bei Flutter hingegen setzt man die Haltepunkte direkt im Android-Studio. Auch eine gute Möglichkeit, allerdings fehlt im Vergleich die Möglichkeit, die View und ihre Komponenten zu inspizieren sowie der asynchrone Netzwerktraffic mit einer Online API lässt sich kaum aufzeichnen und sichtbar machen. Alles in Allem waren aber beide Kandidaten nicht hoffnungslos und der Debug-Prozess ging flott und effizient von der Hand.

- **Tooling**: Beim Thema der verwendeten Tools und Komponenten hat Flutter eindeutig die Nase vorne. Bei Flutter reicht es im Prinzip, das Flutter SDK zu installieren und mit der IDE *Android Studio* loszulegen. Mehr wird nicht benötigt. Im Falle von Ionic ist die Landschaft der (benötigten oder empfohlenen) Tools um Einiges weiter gefasst. Es wird eine Installation von NodeJS, das Vue CLI, das Ionic CLI, Google Chrome, Android Studio zum Builden und eine IDE zum Arbeiten, wie Visual Studio Code oder JetBrains Webstorm (inklusive Plugins), benötigt.

- Wenn man eher ein Freund von Assistenten ist und sich noch nicht mit der Commandline angefreundet hat, ist Flutter eine bessere Lösung, da fast alle Tasks über Buttons im Android Studio erledigt werden können. Die meisten entwicklungsrelevanten Aufgaben werden bei Ionic über die

Commandline abgehandelt. Allerdings sehe ich das nicht als Nachteil, da fast alle bekannten Entwicklerwerkzeuge sich über CLIs bedienen lassen.

- Die Sprache an sich, also **Dart oder JavaScript**, bleibt eine Geschmacksfrage. Mir, als erfahrener Web Entwickler, liegt JavaScript im ersten Moment näher. Über das Tooling bei Ionic ist es zum Beispiel auch möglich, anstatt JavaScript zB TypeScript, anstatt HTML mit PUG und anstatt CSS mit SASS zu arbeiten. Alle diese Dinge werden transpiled, übersetzt und im Buildprozess wieder zu validem HTML, JS und CSS gerechnet. Flutter hingegen bietet mit Dart eine *„One for All"* Lösung. Es ist auch ein Vorteil, eine einzige Sprache innerhalb eines App-Projektes zu programmieren.

- Wenn man sich die **Online Entwicklerdokus** im Vergleich ansieht, schneidet Flutter meiner Meinung nach besser ab. Es gibt eine gute Komponentenbeschreibung und eine sauber dokumentierte API-Beschreibung. Das soll nicht bedeuten, dass Ionic eine schlechte Dokumentation hat, allerdings ist die Dokumentation der Version 5 zum Zeitpunkt der Veröffentlichung dieser Arbeit nicht nur unvollständig sondern auch verwirrend, da Rücksicht auf die verschiedenen Implementierungsvarianten (Angular, React, Vue) genommen werden muss.

- Das ist auch der Grund, warum es schwerer erscheint, konkrete Hilfe für eine Problemstellung in den **Communities** zu erhalten. Ionic ist in sich selbst fragmentiert. Die meisten Entwickler entwickeln nach wie vor Angular, weswegen es einfacher ist hier Hilfe zu erhalten als bei den neueren Geschmacksrichtungen Vue und React.

9 VERWENDETE TOOLS

- Balsamiq Mockups (Wireframes)
- Coolers Color Theme Generator
- Adobe XD (Screendesign)
- Wix Logo Maker (Logo)
- Adobe Photoshop (Logo)
- ApeTools ImageGorilla (Icon, Splashscreen
- Laravel/PHP (Backend Development)
- WAMP Server (Backend Development)
- Postman (API Testing)
- Swagger (API Documentation)
- Dia (Data Modeling)
- NodeJS, NPM (Frontend Development)
- mySQL + mySQL Workbench (Backend Development)
- Google Chrome (Debugging und Development)
- Visual Studio Code (Development)
- Visual Studio Code Laravel Plugin (Development)
- Visual Studio Code Ionic Plugin (Development)
- Visual Studio VueJS Plugin (Development)
- Flutter SDK (Development)
- Android Studio + Android SDK (Development)
- Android Studio Flutter Plugin (Development)
- VueJS CLI (Development)
- Ionic CLI (Development)

10 LITERATURVERZEICHNIS

Dormann, A., 2019. Ionic 4+: Creating awesome apps for iOS, Android, Desktop and Web. D&D Verlag, Bonn, Germany.

Freeman, Eric, Freeman, Elisabeth, Sierra, K., Bates, B., 2005. Entwurfsmuster von Kopf bis Fuß, 1 edition. ed. O'Reilly Verlag GmbH & Co. KG, Beijing.

Gore, A., 2017. Full-Stack Vue.js 2 and Laravel 5: Bring the frontend and backend together with Vue, Vuex, and Laravel. Packt Publishing, Birmingham, UK.

Hahn, M., 2017. Webdesign: Das Handbuch zur Webgestaltung, 2 edition. ed. Rheinwerk Design, Bonn.

Jetbrains Developer Survey 2020 [WWW Document], 2020. URL https://www.jetbrains.com/de-de/lp/devecosystem-2020/

Liebel, C., 2018. Progressive Web Apps: Das Praxisbuch. Plattformübergreifende App-Entwicklung mit Angular und Workbox. Für Browser, Windows, macOS, iOS und Android, 1 edition. ed. Rheinwerk Computing, Bonn.

M.A. Dirk Srocke / Florian Karlstetter, 2017. Was ist eine REST API. Cloudcomputing-Insid. URL https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/

Schmidt, S., 2007. PHP design patterns: Entwurfsmuster für die Praxis ; deckt PHP 5.1 ab, 2., korr. Nachdr. ed. O'Reilly, Beijing.

Semler, J., Tschierschke, K., 2019. App-Design: Das umfassende Handbuch. Alles zur Gestaltung, Usability und User Experience von iOS-, Android- und Web-Apps, 2 edition. ed. Rheinwerk Design.

Sturgeon, P., Bohill, L., 2015. Build APIs You Won't Hate: Everyone and their dog wants an API, so you should probably learn how to build them, 1 edition. ed. Philip J. Sturgeon.

Zammetti, F., 2019. Practical Flutter: Improve your Mobile Development with Google's Latest Open-Source SDK, 1st ed. edition. ed. Apress.

Zammetti, F., 2018. Practical React Native: Build Two Full Projects and One Full Game using React Native, 1st ed. edition. ed. Apress, Chennai, NY.

11 ABKÜRZUNGSVERZEICHNIS

API	Application Programmable Interface
APK	Android Package Kit
CLI	Command Line Interface
CORS	Cross Origin Resource Sharing
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DOM	Document Object Model
IDE	Integrated Development Environment
JS	JavaScript
JSON	Java Script Object Notation
MVC	Model View Controller
MVVM	Model View View Model
NFC	Near Field Communication
NPM	Node Package Manager
ORM	Object Relation Mapping
OS	Operating System
PWA	Progressive Web App
REST	Representational State Transfer
SDK	Software Development Kit
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator

12 GLOSSAR

Ahead-Of-Time Compiling	Code wird kompiliert, damit er zur Laufzeit schnell ausgeführt werden kann und nicht von einem Interpreter interpretiert werden muss
App	Auf mobile Endgeräte verteilbare Applikation
App Screens	Die verschiedenen Ansichten einer App
Asynchronität	Kommunikation zwischen Frontend und Backend erfolgt im Hintergrund, ohne das User Interface zu blockieren
Authentifizierung	Über ein sicheres Verfahren seine Personalien nachweisen
Backend	Server-Gegenstelle, auf welchem Programmlogik ausgeführt und Daten gespeichert werden
Build	Ein optimiertes Kompilat, welches auf dem Endgerät ausgeführt werden kann
Businesslogic	Die Datenverarbeitung, die auf dem Backend, also auf dem Server passiert
Client/Server Applikation	Verteilte Applikationen, die eine zentralisierte Server-Schnittstelle für die Geschäftslogik aufweisen
Controller	Ein Element, welches Geschäftslogik auf Daten in einer Software anwendet
CORS Preflight	Ein Pre-Request über das Options-Verb welcher am Server ein Anfragen einer Ressource anmeldet
Cross Platform Development	Entwicklung für verschiedene Zielplattformen wie Android und iOS

Datenbank-Migration	Eine Anweisung für ein ORM Mapping Tool, Model-Daten in die Datenbank upzudaten
Datenpersistierung	Speichern der Daten auf dem Server
Debugging	Suchen von Bugs und Fehlern
Deployment	Verteilung, zB einer App über einen App-Store
Design Patterns	Bekannte Entwurfsmuster in der Software Entwicklung, welche bekannte Probleme auf eine intelligente Art lösen
Device	Mobiles Endgerät
DevTools	Toolset zum Entwickeln. Hilft beim Debuggen.
Dumb Phone	Endgerät, welches keine Apps installieren kann
Emulation	die Nachahmung eines mobilen Endgerätes durch Software auf einem PC für Debugging
Entity Relations	Die Beziehungen der verschiedenen Daten-Entitäten zueinander
Frameworks	Sammlung von Technologien, Services, Bibliotheken und/oder Methodiken um einheitliche Rahmenbedingungen, einen „Frame“ für eine angenehme Softwareentwicklung zu schaffen
Frontend	Oberfläche einer Website oder einer App
Full-Stack Development	Die Summe von Frontend und Backend Entwicklung
Hot Reload	Währen dem Entwickeln läuft die Applikation und ändert sich direkt beim Ändern des Source-Codes mit
HTTP Verben	GET, POST, PUT, DELETE → Die Verben (Methods), welche semantische Bedeutung

	zur Kommunikation mit einer REST API geben
Kernel	Der Kern einer Applikation, einer Library
Library	Programmbibliothek. Sammlung von Funktionen
Localhost	Die eigene Entwicklermaschine, der PC auf dem Entwickelt wird
Lock-In Effekt	Der Effekt der auftritt, wenn man bereits viel Energie in ein System gesteckt hat und deswegen nur schwer auf ein anderes System wechseln kann
Lifecycle Hooks	Eventbasierte Zeitpunkte im Lebenszyklus einer Applikation auf die man reagieren kann
Middleware	Eine Mittelschicht, welche zwischen zwei Schichten vermittelt
Model	Die Repräsentation der Daten in einer Software
Multi User Fähigkeit	Mehrere User können einen eigenen Datenstand in dieser App anlegen
Native	Bedeutet, etwas wird kompiliert und direkt vom Prozessor des Endgerätes ausgeführt, ohne Zwischenschichten
Object-Relational Mapping	Die automatische Zuordnung von Datenfeldern eines Models auf Tabellen und Spalten in eine relationale Datenbank
Paketmanagementsystem	Online Bibliothek um Komponenten, Libraries oder Logik nachzuladen
Prerequisites	Voraussetzungen, zB für eine Installation
Profiling	Suchen von Geschwindigkeitsbremsen
Reaktivität	Ein Datenfeld ist an ein UI Element gebunden. Ändern sich die Daten, ändern

	sich auch alle gebundenen UI Elemente mit und umgekehrt
Response	Die Antwort eines Servers auf einen Request
Request	Die Anfrage an einen Server
Review	Bewertung
Router	Kümmert sich (client- oder serverseitig) um das korrekte Ansteuern einer angefragten Ressource
Same Origin Policy	Regel, welche verhindert, dass eine URI außerhalb der Domain kontaktiert wird
Scaffolding	Das automatische Erstellen einer Ordnerstruktur beim Erstellen eines Projektes durch das Command Line Interface
Screendesign	Eine ausgearbeitete grafische Version des Endproduktes, welches man einem Kunden zeigen und mit ihm besprechen kann. Farben, Schriften, Layout, Fotos sind hier bereits integriert
Seeden	Ein initiales Befüllen der Datenbank mit Daten
Service	Eine Klasse, welche Methoden bündelt
Smartphone	Endgerät, welches Apps installieren kann
Stateless Widgets	Widgets, welche keinen Zustand annehmen können
Statefull Widgets	Widgets, welche einen Zustand annehmen können, wie zum Beispiel Checkboxen
Terminal	Eine Commandline Umgebung
Transpilen	Sourcecode von einer Sprache in eine andere Sprache übersetzen
View	Eine Ansicht auf Daten in einer Software

WebView	Ein unsichtbarer Brwoser-Frame, welcher einer Website ermöglicht als App gestartet werden zu können
Widget	User Interface Element in Flutter
Widget-Tree	Die Verschachtelung aller Widgets auf einem Screen.
Wireframe	Ein konzeptionelles Mockup, also eine erste rohe Version des Layouts ohne grafische Elemente