universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Realisierung einer Cross-developed Ecological Momentary Intervention App für Herz-Risiko-Patienten

Abschlussarbeit an der Universität Ulm

Vorgelegt von:
Enes Atar
enes.atar@uni-ulm.de
942778

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Prof. Dr. Rüdiger Pryss

2021

Fassung 6. September 2021

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Smartphones gewinnen durch den zunehmenden Einsatz im Alltag an Bedeutung. Ein möglicher Einsatzbereich im Alltag ist das Gesundheitswesen, welche *mobile Health (mHealth)* genannt wird. Hierbei können Patienten in Anwendungen Fragebögen auf mobilen Geräten beantworten, wodurch die Automatisierung der Auswertung und Datenerfassung für größere Benutzergruppen ermöglicht wird.

Im Rahmen dieser Arbeit wurde ein generischer Fragebogen für Herz-Risiko-Patienten erstellt. Diesen Fragebogen können die Patienten mit einer mobilen iOS und Android Anwendung beantworten. Die Patienten erhalten anschließend eine automatisierte Auswertung der Fragen in Form eines Feedbacks. Der Datenaustausch erfolgt dabei mit einer REST API, die vom *Institut für Datenbanken und Informationssysteme (DBIS)* der Universität Ulm zur Verfügung gestellt wird. Zusätzlich bietet die entwickelte Applikation weitere Funktionen an, welche die REST API zur Verfügung stellt.

Danksagung

An dieser Stelle möchte ich mich bei Prof. Dr. Manfred Reichert für die Begutachtung dieser Arbeit bedanken.

Ein besonderer Dank gilt an meinen Betreuer Prof. Dr. Rüdiger Pryss für seine ausgiebige Unterstützung und hilfreichen Anregungen im Verlauf der Arbeit. Des Weiteren bedanke ich mich bei Eray Özhan Kaya und Johannes Vedder für das sorgfältige Korrekturlesen und Altug Paycin für die hilfreichen Anregungen während der Implementierung der Applikation. Abschließend bedanke ich mich bei meiner Familie und meinen Freunden, welche mich während der gesamten Abschlussarbeit unterstützt und motiviert haben.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel wird zunächst die Wichtigkeit der Smartphone-Branche im Bereich des Gesundheitswesens beschrieben. Danach wird die Zielsetzung der zu entwickelnden mobilen Anwendung verdeutlicht. Anschließend wird auf die Struktur der Arbeit eingegangen.

1.1 Motivation

Die Digitalisierung gewinnt zunehmend an Bedeutung im Alltag. Vor allem durch die COVID-19-Pandemie wurde die Wichtigkeit der Digitalisierung in verschiedenen Branchen in den Vordergrund gestellt. Diese betrifft ebenfalls die Gesundheitsbranche, dort betrug beispielsweise der Umsatz im Jahr 2019 106 Milliarden US-Dollar und soll sich laut Prognose von *Global Market Insights* bis zum Jahre 2026 sich mehr als versechsfachen, wie Abbildung 1.1 darstellt [31].

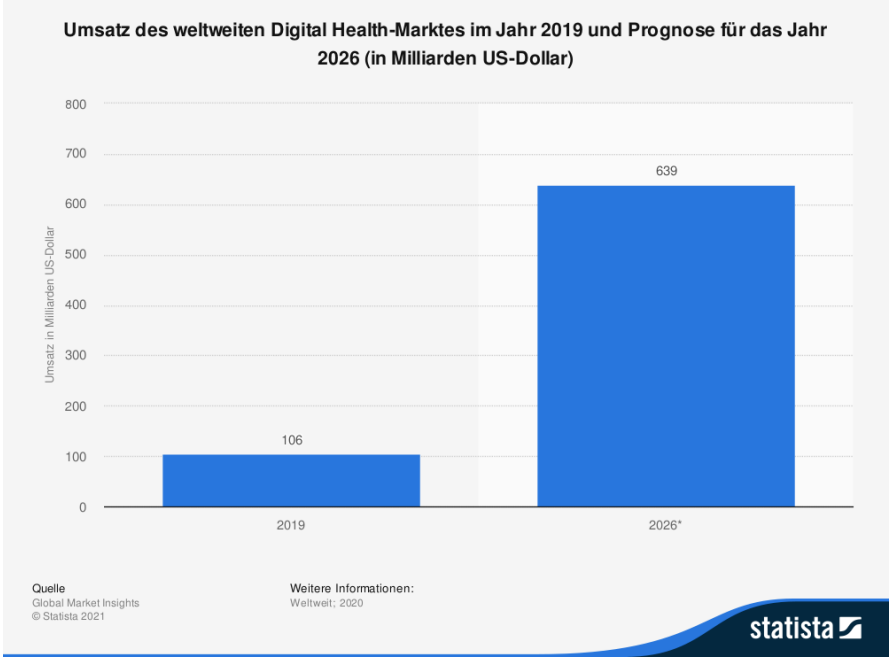


Abbildung 1.1: Umsatz des weltweiten Digital Health-Marktes im Jahr 2019 und Prognose für das Jahr 2026 (in Milliarden US-Dollar) [31]

Einen Teil vom Digitalen-Health-Markt bildet der E-Health-Bereich ab. Der E-Health-Bereich umfasst Anwendungen, in denen Patienten in ihrer Behandlung und Betreuung durch *Informations- und Kommunikationstechnologien (IKT)* unterstützt werden [20]. Allein im Jahr 2020 stieg die Zahl von Smartphone Nutzern binnen eines Jahres um mehr als 412 Millionen [14]. Die genaue Entwicklung der Smartphone-Nutzer weltweit aus den Vorjahren und einer Prognose für die laufenden Jahre sind in Abbildung 1.2 von *Ericsson*, einem Diensteanbieter für *IKT* zu finden. In Anbetracht der jährlich weltweit wachsenden Smartphone-Nutzerrate spielt der Einsatz von mobiler und drahtloser Technologie eine wesentliche Rolle im Gesundheitswesen. Dabei haben sich auf dem Smartphone-Markt die Betriebssysteme *iOS* von Apple und *Android* von Google im Laufe der Jahre durch gesetzt. Nach einer Analyse von StatCount betrug im Jahre 2020 der Android-Anteil aller Geräte 73%, iOS hingegen 26% [51].

Der Einsatz von mobilen Geräten in der Gesundheitsbranche gehört dem Bereich des *mHealth* an. Diese wird von der World Health Organization (WHO) als *medizinisches und öffentliche Gesundheitspraxis, die durch mobile Geräte wie Mobiltelefone, Patienten Überwachungsgeräte, persönliche digitale Assistenten (PDAs) und*

andere drahtlose Geräte definiert [38]. mHeatlh bietet den Vorteil, Patientendaten automatisiert zu sammeln und auszuwerten, ohne die Anwesenheit von Fachkräften zu benötigen. Diese Arbeit beschäftigt sich mit dem Entwicklungsprozess einer Smartphone-Anwendung mit einem Cross-Platform-Framework, welcher mit einem REST API kommuniziert. Zusätzlich kann der Benutzer einen elektronischen Fragebogen über koronare Herzkrankheiten ausfüllen und die jeweilige Auswertung begutachten.

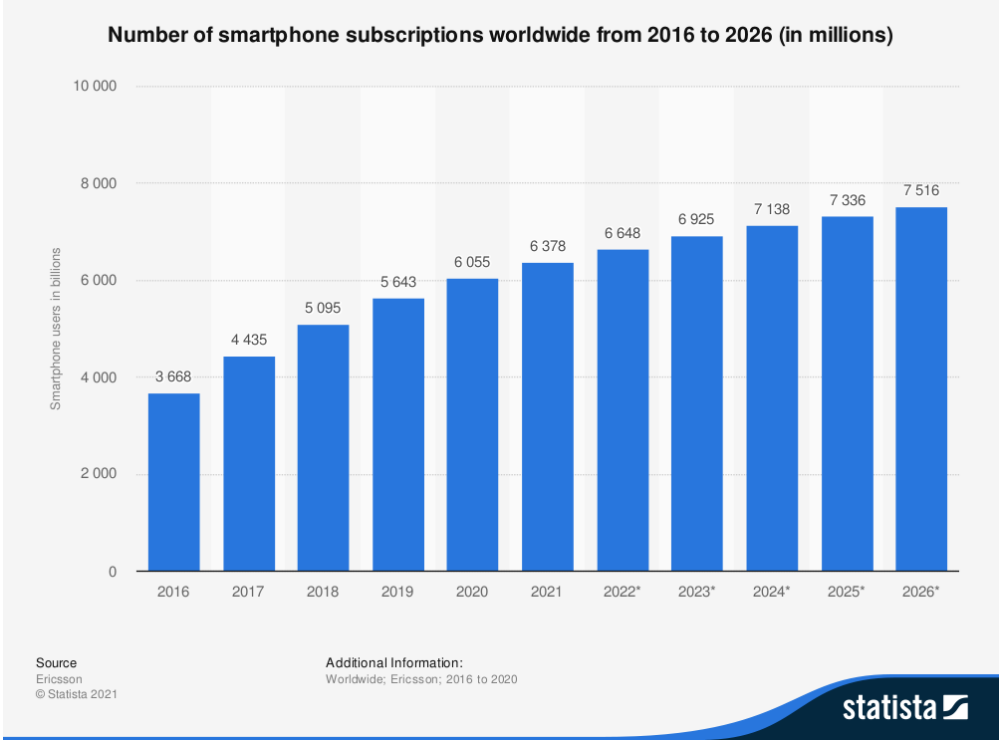


Abbildung 1.2: Prognose zur Anzahl der Smartphone-Nutzer weltweit von 2016 bis 2026 (in Millionen) nach Daten von Ericsson [14]

1.2 Zielsetzung

Das Ziel der Arbeit ist das Erstellen eines Fragebogens für koronare Herz-Risiko-Patienten und einer Smartphone-Anwendung, in der Studien mit jeweiligen Fragebögen zur Verfügung gestellt werden. Der Benutzer soll sich dabei mit einem Benutzerkonto anmelden können. Nach der Anmeldung ist es möglich, sich in Studien ein/-auszuschreiben. Unter anderem soll auch das Ausfüllen des selbst erstellten Fragebogens über die Risikofaktoren in Bezug auf koronare Herzkrankheiten möglich sein. Aufgrund des hohen Anteils von iOS und Android Nutzern soll die Anwendung mit dem *Flutter* Cross-Platform Framework implementiert werden. Das Flutter Framework hat den Vorteil, die Entwicklungs- und Wartungszeit zu minimieren, da für beide Plattformen nur eine Codebasis benötigt wird.

1.3 Struktur der Arbeit

Diese Arbeit bietet einen Überblick des Entwicklungsprozesses und der Fertigstellung einer plattformübergreifenden App für Herz-Risiko-Patienten in Flutter. In Kapitel 2 werden für das bessere Verständnis die Grundlagen erläutert, die in den nachfolgenden Kapiteln vorausgesetzt werden. Bereits ähnlich existierende Arbeiten werden in Kapitel 3 vorgestellt. Anschließend werden in Kapitel 4 die funktionalen und nicht-funktionalen Anforderungen an die zu entwickelnde Applikation definiert. Die Designentscheidungen für das User-Interface und die resultierenden Prototypen werden Kapitel 5 vorgestellt. Die Architektur vom System mit der möglichen Navigation, dem genutztem Pattern und der Datenaustausch zwischen Applikation und Server werden im Kapitel 6 genauer erläutert. Im Anschluss werden in Kapitel 7 die genutzten Werkzeuge sowie einige Implementierungsaspekte der realisierten Applikation vorgestellt. Ein Abgleich der in Kapitel 4 definierten Anforderung mit der erstellten Applikation ist in Kapitel 8 enthalten. Abschließend wird in Kapitel 9 die Arbeit zusammengefasst und ein Ausblick auf die zukünftige Weiterentwicklung der entstandenen Applikation eingegangen.

2 Grundlagen

In diesem Kapitel werden Grundlagen und Begriffe erläutert, die im späteren Verlauf der Arbeit genutzt werden.

2.1 Koronare Herzkrankheit

Eine Person mit einer koronaren Herzkrankheit kann einen unerwarteten Myokardinfarkt (Herzinfarkt) oder einem Herztod erleiden [3]. Die Erkrankung wird meist durch Arteriosklerose oder Thrombose verursacht, wodurch die Herzkranzgefäße (Koronararterien) verengt oder verstopft werden. Dies führt zu einem Sauerstoffmangel der Herzmuskulatur [3, 52].

Das Risiko, an einer koronaren Herzkrankheit zu erkranken, kann von insgesamt 8 Risikofaktoren zusammenhängen [1, 3]. Diese Faktoren werden in beeinflussbaren und nicht beeinflussbaren Risikofaktoren unterteilt.

Mit diesen 8 Risikofaktoren wurde in Münster von 1979 bis 1985 die *Prospective Cardiovascular Münster Study (PROCAM-Study)* durchgeführt. Das Ergebnis der Studie ist ein einfaches, aber genaues Punktesystem zur Abschätzung des Risikos, einen Herzinfarkt innerhalb 10 Jahren zu erleiden. In dieser sind die Risikofaktoren nach Wichtigkeit sortiert, zugleich wird jedem Risikofaktor in Abhängigkeit der Antwortmöglichkeit eine Punktzahl zugeordnet. Das Punktesystem mit den jeweiligen Risikofaktoren ist in Tabelle 2.1 aufgeführt. Anzumerken ist, dass dies lediglich ein Überblick der Punkte darstellt und nicht die genauen Punktzahlen mit den jeweiligen Risiken [1]. Ferner ist zu beachten, dass die Studie nur bei Männern im Alter von 35–65 Jahren durchgeführt worden ist. Deshalb ist das Punktesystem außerhalb dieser Altersgruppen und Geschlechtern nur bedingt aussagekräftig. Durch die Gesamtpunktzahl der Risikofaktoren kann der globale Risikofaktor abgeschätzt

werden. Die Gesamtpunktzahl mit den Wahrscheinlichkeiten einer akuten koronaren Herzkrankheit kann in Tabelle 2.2 abgelesen werden.

Risikofaktor	Antwort	Punkte
Alter	35-39	0
	40-44	6
	45-49	11
	50-54	16
	55-59	21
	60-65	26
LDL-Cholesterinwert, mg/dL	<100	0
	100-129	5
	130-159	10
	160-189	14
	$\geq$ 190	20
HDL-Cholesterinwert, mg/dL	<35	11
	35-44	8
	45-54	5
	$\geq$ 55	0
Triglyceridwert, mg/dL	<100	0
	100-149	2
	150-199	3
	$\geq$ 200	4
Raucher	Nein	0
	Ja	8
Diabetes Mellitus	Nein	0
	Ja	4
Systolischer Blutdruck, mm Hg	<120	0
	120-129	2
	130-139	3
	140-159	5
	$\geq$ 160	8

Tabelle 2.1: Punktesystem mit den Risikofaktoren aus der PROCAM-Studie [1]

Punktzahl	10-Jahres-Risiko für akute koronare Ereignisse (in %)
0-20	<1.0
21-25	1.1-1.6
26-30	1.7-2.4
31-35	2.8-4.0
36-40	4.2-6.1
41-45	7.0-7.10.2
46-50	10.5-15.5
51-55	16.8-22.2
56-59	23.8-29.4
$\geq$ 60	$\geq$ 30.0

Tabelle 2.2: Überblick vom PROCAM-Punktesystem für das Risiko von akute koronare Ergeignisse [1]

2.2 REST APIs

Um den Datentransfer zwischen Systemen unabhängig von ihrer Implementierung zu erleichtern, wurde das sogenannte *Application Programming Interface (API)* entwickelt. APIs sind eine Programmierschnittstelle, in der Produkte und Services untereinander kommunizieren können [37]. Da unterschiedliche Arten von APIs existieren, jedoch für diese Arbeit die *Web APIs* relevant sind, wird nur auf diese genauer eingegangen.

Web APIs oder auch *Web-Services* genannt, ermöglichen den Datenaustausch von Maschine-zu-Maschine. Der Datenaustausch erfolgt dabei über das Internet durch HTTP-Anfragen, welche die XML-Serialisierung und andere Web-Standards nutzen [27]. Diese HTTP-Anfragen und Web-Standards werden in den nächsten Abschnitten genauer erläutert.

Bei der Entwicklung von Web-APIs wird als bevorzugter Standard die *Representa-*

tional State Transfer (REST) Architektur eingesetzt. Web-APIs, die auf diese Architektur basieren, werden *RESTful APIs* genannt [37]. Der REST Architektur-Stil ist in der Dissertation von *Roy Fielding* im Jahre 2000 entwickelt und konzipiert worden. In dieser legt er sechs Beschränkungen fest, die diesen Architektur-Stil beschreiben [16]:

1. **Client-Server:** Die Trennung von Client und Server. Durch diese Trennung wird die Portabilität der Benutzeroberfläche verbessert, da die Datenspeicherung von der Benutzeroberfläche getrennt ist. Vor allem können somit beide Komponenten unabhängig voneinander weiterentwickelt werden [16]. Der Datenaustausch erfolgt über einen *Anfrage-Antwort (Request-Response)* Zyklus. Dabei stellt der Client die Request-Nachricht an den Server, dieser wiederum verschickt eine Response-Nachricht zurück.
2. **Zustandslosigkeit:** Damit der Request vom Client auf den Server richtig bearbeitet wird, muss jede Request alle nötigen Informationen beinhalten [16]. Dadurch hat jede Request den erforderlichen Zustand und der Server muss nicht den Zustand der Daten im Zusammenhang der Sitzung speichern.
3. **Cache:** Die Response Nachricht vom Server wird explizit als cachefähig oder nicht cachefähig deklariert. Falls die Response cachefähig ist, kann der Client für spätere gleichwertige Request diese Responsedaten wiederverwenden [16]. Das hat den Vorteil, dass der Request nicht nochmals an den Server verschickt werden muss, was die Leistung und Verfügbarkeit in der Interaktion vom Client verbessert.
4. **Einheitliche Schnittstelle:** Die Schnittstelle ist einheitlich und ermöglicht, dass die Informationen in einer standardisierten Form übermittelt werden, unabhängig vom Gerät oder der Art der Anwendung. Diese einheitliche Schnittstelle wird wiederum durch weitere Beschränkungen erreicht. Diese Beschränkungen sind die *Identifizierung* von Ressourcen, *Manipulation* von Ressourcen durch Darstellungen, *selbstbeschreibende Nachrichten* und der *Anwendungszustand* über Hypermedia [16].
5. **Mehrschichtiges System:** Die Anwendungsarchitektur muss aus mehreren hierarchischen Schichten zusammengesetzt sein. Dabei hat jede Schicht keine Information über die andere Schicht. Die Kapselung reduziert die Komplexität des Systems und erhöht die Sicherheit vor Angriffen von Clients [16].

6. **Code-On-Demand:** Die Code-On-Demand Beschränkung beschreibt das Ausführen von Programmteilen über Skripte oder Applets, die vom Server bereitgestellt werden. Der Client lädt somit die Skripte herunter und führt diese clientseitig aus. Diese Einschränkung ist jedoch optional [16].

2.3 HTTP

Die meisten Dienste und Anwendungen kommunizieren über das Internet. Dabei kontrollieren Protokolle das Senden und Empfangen von Nachrichten. REST APIs nutzten dabei hauptsächlich das HTTP Protokoll, welches in diesem Kapitel genauer vorgestellt wird.

2.3.1 HTTP Protokoll

HTTP steht für *Hypertext Transfer Protocol*. Dieses HTTP ist ein Transferprotokoll, welches die *Formate* und *Reihenfolgen* beim Austausch von Nachrichten zwischen Client und Server festlegt. HTTP ist *zustandslos (stateless)*, womit keine früheren Client-Anfragen auf dem Server gespeichert werden. Das HTTP Protokoll benutzt *TCP/IP* zum Verbindungsaufbau. Dabei initiiert der Client eine TCP-Verbindung über einem Socket zum Server. Der Server wiederum akzeptiert den TCP Verbindungsaufbau. Im Anschluss werden die HTTP Nachrichten in Form von Request- und Response-Nachrichten zwischen Client und dem Web-Server ausgetauscht. Den Request stellt der Client über *URLs (Uniform Resource Locator)* um die jeweiligen Ressourcen an den Server anzufragen, dieser sendet wiederum eine Response-Nachricht [17]. Das Prinzip ist in Abbildung 2.1 schematisch verdeutlicht.

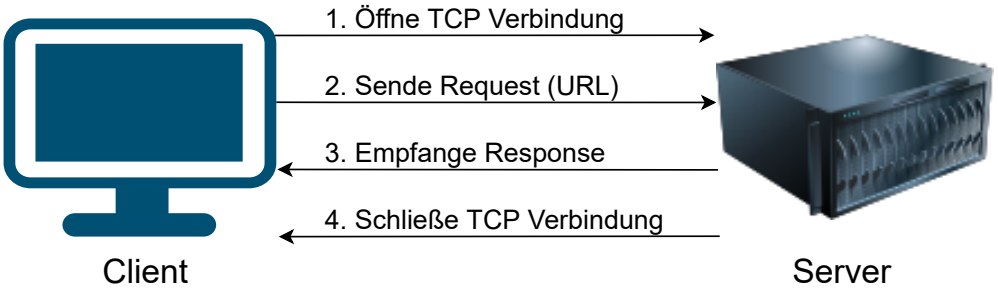


Abbildung 2.1: HTTP-Datenaustausch zwischen Client und Server, eigene Darstellung nach [30]

Eine HTTP Nachricht besteht dabei aus den folgenden drei Komponenten:

1. **Methode/Status:** Eine HTTP-Response besteht immer aus einem Statuscode, welches im Unterabschnitt 2.3.3 genauer vorgestellt wird. Die HTTP-Request hingegen besitzt keinen Statuscode, sondern besteht immer aus einer Methode. Diese Methoden bestimmen die Aktion der Anfrage. Dabei können folgende Methoden implementiert werden [17]:
   - **GET** fordert eine Ressource an (muss implementiert werden).
   - **HEAD** fragt nur den Header einer Ressource ohne den Body an (muss implementiert werden).
   - **OPTIONS** liefert Informationen zur Kommunikationskette wie bspw. die unterstützenden Methoden vom Server.
   - **POST** schickt eine Request mit Übertragung von Parametern.
   - **PUT** übertragt eine Ressource zum Server.
   - **DELETE** löscht eine Ressource auf dem Server.
   - **TRACE** liefert Request vom Server zurück an den Client zur Überprüfung, ob es Änderungen bei der Übertragung gab.
   - **CONNECT** um einen SSL Tunnel zu öffnen.
2. **Header:** Durch HTTP-Header (Kopfzeilen) können Client und Server zusätzliche Metainformationen oder Einstellungen einer Nachricht anhängen. Diese enthalten unter anderem Informationen der verwendeten Codierung im Body [17]. Zusätzlich ist das Einfügen von *Content Negotationen* möglich, welche im nächsten Abschnitt vorgestellt wird.

3. **Body:** Der Body (Nachrichtenrumpf) ist der eigentliche Nachrichtentext in Datenbytes [17].

Die Nachrichten werden dabei in ASCII-kodiert, jedoch seit HTTP/2 binär [7, 17]. Die grundlegenden Operationen mit gespeicherten Daten sind das Erstellen, Lesen, Aktualisieren und Löschen. Das Konzept wird *CRUD* genannt und steht für *C*reate, *R*ead, *U*pdate und *D*elete. Das CRUD Konzept wird auch in RESTful APIs eingesetzt und ist durch folgende HTTP-Methoden abgedeckt [6]:

- **Create** *POST* oder *PUT*
- **Read** *GET*
- **Update** *PUT* oder *UPDATE*
- **Delete** *DELETE*

2.3.2 Content Negotiation

Der Client hat die Möglichkeit durch das Eingeben von *Content Negotationen* im HTTP-Header, verschiedene Darstellungen einer Ressource unter derselben URL anzufordern. Dabei kann die unterschiedliche Darstellung der Ressource aus sprachspezifischen, Kodierung und Qualität bestehen. Die Content Negotation wird hierbei in *Server-driven* oder *Agent-driven* unterteilt. Beim Server-driven Content Negeotation wird die passende Ressource basierend auf die Anfrage geliefert. Dabei sind folgende Content Negotationen im Header-Feld möglich:

- **Accept**
  - *Funktion:* Gibt an welche Inhaltstypen (MIME-Typen) akzeptiert werden können.
  - *Syntax:* Accept: <MIME_typ>/<MIME_subtyp>
  - *Beispiel:* Accept: text/html
- **Accept-Language**
  - *Funktion:* Liste der Sprachen die akzeptiert werden können.
  - *Syntax:* Accept-Language: <sprache>

  - *Beispiel:* Accept-Language: de

- **Accept-Encoding**
  - *Funktion:* Art des Kompressionsalgorithmus bzw. Kodierung des Inhalts.
  - *Syntax:* Accept-Encoding: <kompression_format>
  - *Beispiel:* Accept-Encoding: gzip

- **User-Agent**
  - *Funktion:* Ermöglicht Informationen über das Betriebssystem, den Hersteller oder die Version des Clients zu übergeben.
  - *Syntax* User-Agent: <produkt> / <produkt-version> <kommentar> <plattform> (<plattform-details>) <erweiterungen>
  - *Beispiel mit Browser Chrome:* Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36

Im Agent-driven Content Negeotation hingegen schickt der Server als Antwort zur ersten Anfrage eine Auswahlliste *(300 multiple choices)* der verfügbaren Darstellungen. Mit einer zweiten Anfrage entscheidet sich der User für die jeweilige Darstellung. Falls beide Arten in Kombination genutzt werden, wird diese *Transparent Negotation* genannt. In dieser besitzt die Proxy eine Auswahlliste (Agent-driven) und entscheidet für den Client die jeweilige Darstellung aus (Server-driven) [17].

Eine REST API nutzt, um den Zustand der Objekte einer Ressource im Body in sinnvolle Felder darzustellen, als Datenformat am häufigsten die *JavaScript Object Notation (JSON)* oder *Extensible Markup Language (XML)* [37]. JSON basiert auf die Datentypen der Programmiersprache JavaScript und hat den Vorteil, dass sie von Entwicklern als auch von Maschinen leicht verstanden werden kann [39]. Die Daten werden dabei mit geschweiften Klammern hierarchisch durch Schlüssel-/Wert-Paare (key-value pairs) strukturiert [37, 39]. Ein einfaches JSON-Objekt ist in Listing 2.1 gelistet.

```
{
  "university": "Universität Ulm",
  "institute": "Institut für Datenbanken und
     Informationssysteme",
  "active": true,
  "members": [
    {
      "firstname": "Manfred",
      "lastname": "Reichert",
      "title": "Prof. Dr.",
    },
    {
      "firstname": "Rüdiger",
      "lastname": "Pryss",
      "title": "Prof. Dr.",
    },
  ]
}
```

Listing 2.1: Einfaches JSON-Objekt

Das JSON-Objekt aus dem Listing 2.1 besteht aus den Attributen `university`, `institute`, `active` und einer Liste `members`. Die Liste `members` enthält wiederum Objekte aus den Attributen *firstname*, `lastname` und `title`. Mit diesem JSON-Format können auch in Post-Methoden Daten vom Client dem Server übergeben werden. Die Darstellungsform wird dabei im Header durch das `Accept:application/json` Feld angegeben.

2.3.3 Statuscodes

Wie in Unterabschnitt 2.3.1 erwähnt, besitzt die HTTP-Response-Nachricht keine Methode, sondern einen *Statuscode*. Anhand dieses Statuscode, wird der Client über den Zustand seiner Request-Nachricht informiert. Dieser befindet sich direkt in der ersten Zeile der Response-Nachricht. Der Statuscode besteht aus einer dreistelligen Zahl und lässt sich in fünf Kategorien unterteilen, die in Tabelle 2.3 gelistet

sind [17].

Kategorie	Statuscode	Beschreibung
Informal	1xx	Der Request wurde übermittelt und die Informationen werden bearbeitet.
Success	2xx	Der Request wurde erfolgreich empfangen und bearbeitet.
Redirection	3xx	Der Client muss zusätzliche Anweisungen oder Aktionen tätigen, da der *Request* aufgrund mangelnder Informationen nicht bearbeitet werden konnte.
Client Error	4xx	Der Request ist fehlerhaft, entweder beinhaltet diese eine falsche Syntax oder kann nicht auf dem Server bearbeitet werden.
Server Error	5xx	Der Server hat einen Fehler, da der Server für die gültige Request-Nachricht keine entsprechende Response-Nachricht generieren konnte.

Tabelle 2.3: Unterteilung der Response Statuscodes in Kategorien [17]

Über diese Statuscodes kann der Entwickler effizient mit der Response-Nachricht arbeiten und je nach Statuscode die weiteren Bearbeitungsphasen einleiten. Beispielsweise wird an eine REST API eine GET-Request gestellt. Dann ist es nur bei einem Success-Statuscode erforderlich, den Response-Body einzulesen. Bei einem Fehler hingegen kann der Entwickler anhand des Statuscodes dem Client mitteilen, weswegen der Fehler entstanden ist.

2.4 Mobile Anwendungen

Mobile Anwendungen sind heutzutage auf unseren Smartphones und Tablets im alltäglichen Gebrauch. Diese werden kurz *App* genannt, welcher sich aus dem englischen Begriff *Application* ableitet. Dabei laufen Apps auf unterschiedlichen Betriebssystemen und lassen sich in drei Gruppen kategorisieren.

Die erste Kategorie sind die *native Apps* [8]. Zu dieser zählen Apps, die explizit in Programmiersprachen für ein bestimmtes Betriebssystem entwickelt werden [33]. Die Entwicklung einer nativen App erfolgt für iOS ausschließlich in den Programmiersprachen Swift[1] oder Objective-C[2]. Für Android hingegen können verschiedene Programmiersprachen eingesetzt werden, dabei sind Java und Kotlin[3] die bevorzugten Sprachen [22]. Native Apps haben den Vorteil, den vollen Zugriff auf die Gerätefunktionen wie Sensor-APIs zu haben, aber können auch Änderungen der Systemeinstellungen vornehmen [33].

Die weiteren Kategorien bilden die Web- und Hybrid-Apps [8]. Im Gegensatz zu nativen Apps unterscheiden sie sich in ihrem Ansatz, da sie plattformunabhängig sind.

Web-Apps basieren auf Webanwendungen, die Webtechnologien wie HTML5, CSS3 und Javascript nutzen. Die App muss beim Benutzer nicht installiert werden, da der Inhalt auf einem Server gehostet wird und der Zugriff auf die Daten über Standardprotokolle wie HTTP stattfindet. Dementsprechend wird für den Zugriff nur die jeweilige URL benötigt um darauf zugreifen zu können. Damit sind Web-Apps Webseiten, die für Mobilgeräte optimiert sind und in einem installierten Browser ausgeführt werden [36]. Die Nutzung der Standardsprachen von Webtechnologien ermöglichen den Entwickler, eine einzige Anwendung mit konsistenter Erfahrung über alle Plattformen hinweg zu entwickeln. Zusätzlich bieten sie eine schnelle Entwicklung, einfache Wartung sowie eine gute Anwendbarkeit [36]. Gegenüber nativen Apps haben Web-Apps nur begrenzten Zugriff auf die Gerätefunktionen und damit nicht das Potenzial, die volle Leistung der grafischen Oberfläche auszunutzen. Zusätzlich benötigen diese eine permanente Internetverbindung [8]. Um diese Lücke zwischen nativen und Web-Apps zu füllen, gibt es die *progressive Web-Apps (PWAs)* [4] [36]. Diese sind ähnlich wie Web-Apps und laufen auf einem Server mit dem Unterschied, dass sie auf dem Gerät installiert werden können. Dadurch ist der Zugriff auf Gerätefunktionen möglich oder das Unterstützen einer Offline-Nutzung durch Verwaltung des Zwischenspeichers [36].

Ähnlich wie Web-Apps basieren *Hybride-Apps* auf Webtechnologien, jedoch sind

[1] `https://developer.apple.com/swift/` (05.08.2021)
[2] `https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html` (05.08.2021)
[3] `https://developer.android.com/kotlin` (05.08.2021)
[4] `https://web.dev/progressive-web-apps/` (05.08.2021)

Hybride-Apps zugleich eine installierbare native App. Die Anwendung läuft dabei in einem Container, dieser bildet eine Brücke zwischen Anwendung und Betriebssystem. Der Container ermöglicht den Zugriff auf die Gerätefunktionen und die Kommunikation zwischen Gerät und Anwendung [8, 33]. Somit ist es möglich, die Vorteile einer Web-App zu nutzen und zugleich auf die spezifischen Gerätefunktionen zuzugreifen.

2.4.1 Cross-Platform-Umgebungen

Wie in Abschnitt 1.1 erwähnt, haben die Betriebssysteme iOS[5] von Apple und Android[6] von Google im Laufe der Jahre sich in der Smartphone-Branche durchgesetzt. Aufgrund der großen Nutzergruppen beider Plattformen ist es für Unternehmen relevant, ihre Applikationen für die beiden Plattformen abzudecken. Somit müssen zwei separate Projekte für die entsprechenden Plattformen realisiert werden. Dies bedeutet, dass für die jeweilige Plattform jeweils ein Projekt erfolgt, welche mit Kostenfaktoren wie Planung, Entwicklungszeit, Test-und Wartungskosten zusammenhängt. Um die eben genannten Faktoren zu senken, bieten sich *Cross-Platform Tools* an. Diese ermöglichen es einem durch eine einzige Codebasis mehrere plattform-spezifische Apps zu erstellen, zu welchen ebenfalls Webbrowseranwendungen zählen. Es gibt verschiedene Frameworks, die den Arbeitsaufwand durch unterschiedliche Ansätze verringern. Die bekanntesten Frameworks sind dabei *Xamarin*[7] (2011) von Microsoft, *Ionic*[8] (2013) von Drifty Co, *React Native*[9] (2015) von Facebook und *Flutter*[10] (2017) von Google.

Um eine native App für Herz-Risiko-Patienten zu entwickeln, wurde in dieser Arbeit sich für das Cross-Platform Tool Flutter entschieden. Flutter nutzt die Programmiersprache Dart, die es ermöglicht, aus einer Codebasis native Apps für iOS und Android zu entwickeln. In Flutter kann auf die direkten Programmierschnittstellen der Geräte zugegriffen werden. Somit haben diese Apps keine wesentlichen Nachteile

[5] `https://www.apple.com/ios` (05.08.2021)
[6] `https://www.android.com/` (05.08.2021)
[7] `https://dotnet.microsoft.com/apps/xamarin` (05.08.2021)
[8] `https://ionicframework.com/` (05.08.2021)
[9] `https://reactnative.dev/` (05.08.2021)
[10] `https://flutter.dev/` (05.08.2021)

im Vergleich zu Apps, die mit einem *Software Development Kit (SDK)* vom Hersteller entwickelt werden.

2.4.2 Dart

Dart ist eine Open Source objektorientierte Programmiersprache, die von Google entwickelt wurde. Die Motivation für die Entwicklung der neuen Sprache hatte zwei Hauptgründe. Die erste Motivation war, eine besser strukturierte Sprache im Vergleich zu *JavaScript*[11] für Web-Apps zu entwickeln. Durch die Struktur soll es möglich sein, die *Virtual Machine (VM)* von Dart im Vergleich zu JavaScript leichter zu optimieren, um die Performance zu verbessern [53]. Nachfolgend haben sich die Ziele für ein App-Framework entwickelt, welcher das schnelle Entwickeln einer App für plattformübergreifende Geräte, gepaart mit einer flexiblen Laufzeitplattform ermöglichen soll [24].

Ein weiterer Grund war es, die Produktivität der Entwickler zu steigern. Ein essenzieller Bestandteil ist das Zusammenarbeiten mit anderen Entwicklern. Die Zusammenarbeit kann durch das Wiederverwenden von Codeteilen untereinander erfolgen, aber auch durch das Unterstützen von *Libraries (Bibliotheken)* und APIs [53]. Die APIs werden in Dart durch *Packages (Softwarepaketen)* bereitgestellt und auf der offiziellen Seite von Google unter *pub.dev*[12] veröffentlicht. Ebenfalls ist das Veröffentlichen von eigenen Packages möglich, da Dart ein Open-Source-Projekt ist.

Die Sprache an sich verfügt über Kernbibliotheken, welche die meist genutzten Programmieraufgaben wie verkettete Listen, Hashmaps, eingebaute Typen, Sammlungen und andere Kernfunktionen enthalten. Die Annotation der Typen ist in Dart optional, aber die Sprache selber ist *type safe*. Das bedeutet, dass der Wert einer Variablen mit dem statischen Typ übereinstimmen muss. Die Kombination von dynamischen Typen ist dabei auch möglich. Eine Besonderheit ist, dass nur Werte *null* sein dürfen, die explizit so deklariert werden. Damit soll in der Laufzeit der Compiler vor Null-Ausnahmen geschützt werden. Dieses Konstrukt wird *null safety* genannt [24]. Der erstellte Code kann durch den Compiler auf zwei unterschiedlichen Arten kompiliert und ausgeführt werden:

[11] `https://www.javascript.com/` (06.08.2021)
[12] `https://pub.dev/` (06.08.2021)

1. **Dart Native** enthält eine VM mit *Just-in-time-Kompilierung (JIT)* und *Ahead-of-time-Kompilierung (AOT)* [24]. AOT-Kompilierung bedeutet, dass der Programmcode unabhängig vor der Ausführung in eine native Maschinensprache übersetzt wird. Bei einer JIT-Kompilierung hingegen wird der Programmcode zur Laufzeit in Maschinencode übersetzt. Durch diesen Ansatz ist es möglich, Anwendungen nativ auf Mobil- und Desktop Geräten zu nutzen.

2. **Dart Web** ermöglicht den in Dart geschriebene Programmcode in JavaScript zu übersetzen. Dadurch kann das geschriebene Programm im Webbrowser ausgeführt werden [24].

Wie am Anfang erwähnt, ist Dart eine objektorientierte Sprache. Somit werden Konzepte wie Klassen und Vererbung unterstützt. Die Sprache ist an die Programmiersprache *C* angelehnt und unterstützt *funktionale* und *reaktive* Programmierparadigmen [23]. Aufgrund dieser Charakteristiken ähnelt die Sprache in vielen Aspekten *Java*. Zur Demonstration der Ähnlichkeit der beiden Sprachen ist ein *Hello World* Programm für Java in Listing 2.2 und für Dart in Listing 2.3 aufgeführt.

```
1 public class Hello {
2     public static void main ( String [] args ) {
3     System.out.println("Hello, World!");
4     }
5 }
```

Listing 2.2: Programm *Hello World* in Java

```
1 // Falls die öffnende geschweifte Klammer durch "=>" ersetzt wird, können
      beide Klammern weggleassen werden.
2 void main() {
3   print('Hello, World!');
4 }
```

Listing 2.3: Programm *Hello World* in Dart

2.4.3 Flutter

Flutter ist eine Open-Source (auf Github) Cross-Platform Software Development Kit (SDK) von Google, in welcher native 2D Anwendungen in der Programmiersprache

Dart implementiert werden können. In Flutter können somit native Anwendungen durch eine einzelne Codebasis für iOS, Android, Web, Linux, macOS und Windows entwickelt werden. Das Flutter Team bietet für die bekanntesten IDEs (IntelliJ IDEA, VisualStudio Code, Android Studio) eigene *Plugins (Erweiterungen)* an, welche die Entwicklung in Dart und Flutter erleichtern sollen. Wie in anderen Programmiersprachen können iOS Apps nur auf einem Mac mit *Xcode* kompiliert werden [25].
Die erstellten Anwendungen laufen nicht auf allen Endgeräten, da Flutter eine Systemanforderung voraussetzt. Die aktuellen Anforderungen können aus der Tabelle 2.4 entnommen werden.

Plattform	Version
Android	über Android SDK 15
iOS	über iOS Version 8
Windows	ab Windows 7
macOS	über Yosemite

Tabelle 2.4: Systemanforderungen von Flutter [25]

Flutter besteht aus zwei Komponenten, der Engine und dem Framework. Die Engine beinhaltet eine Reihe von Programmierwerkzeugen, die es vor allem ermöglicht, dass der geschriebene Code in einen nativen Code kompiliert wird. Dadurch kann die Applikation auf unterschiedlichen Plattformen ausgeführt werden. Die genauere Funktionsweise wird im nächsten Abschnitt genauer erläutert.
Das mitgelieferte Framework ist in *C* und *C++* implementiert und verfügt über eine Reihe von Werkzeugen. Zu diesen zählen unter anderem die Programmiersprache Dart und die verschiedenen UI-Elemente in Form von Widgets. Flutter nutzt nicht die *Original Equipment Manufacturer (OEM)* Widgets des Betriebssystems, sondern bietet selber ein Set von Widgets an. Diese werden im *Material Design* (Android) und *Cupertino* (iOS-Style) zur Verfügung gestellt [25]. In Abbildung 2.2 sind die Komponenten visuell dargestellt.

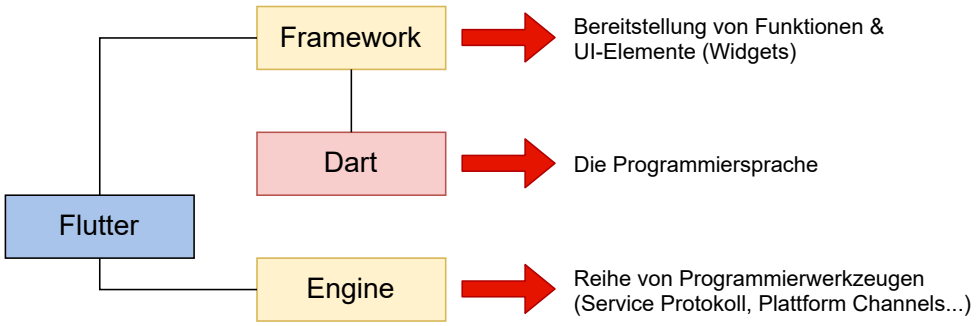


Abbildung 2.2: Bestandteile der Flutter SDK, eigene Darstellung nach [11]

Engine: Die Flutter Engine ist eine portable Laufzeitumgebung, bestehend aus unterschiedlichen Technologien. Dazu gehört die 2D-Grafik *Rendering-Engine* namens *Skia*, die ebenfalls von Google entwickelt wurde [11]. Die Engine ermöglicht jedes einzelne Pixel auf dem Bildschirm anzusteuern. Dadurch wird die Benutzeroberfläche einer Applikation auf die UI gezeichnet und nutzt nicht die nativen Widgets. Des Weiteren besteht die Engine aus einer *Shell*, die die Dart Virtuell Maschine (VM) beherbergt. Die Shell ist plattformspezifisch, da sie die Plattform APIs und *Application-Lifecycle Events* implementiert hat [10]. Die Dart Programmbibliothek und die zusätzliche *dart:ui* Bibliothek sind in die VM integriert. Diese ermöglicht den Zugriff auf die Shell und auf die Funktionen der Skia Render-Engine [11].

Die Engine besteht unter anderem aus C und C++ Code. Damit diese auf Android und iOS laufen, werden diese je nach Betriebssystem mit Androids *Native Development Kit (NDK)*[13] oder *LLVM*[14] kompiliert [25]. Da Dart wie schon in Unterabschnitt 2.4.2 erwähnt eine AOT-Kompilierung verwendet, wird der in der Engine vorhandene und für die Applikation geschriebene Dart Code für iOS in ARM für Android in ARM und x86 Bibliotheken übersetzt. Die entstandenen Bibliotheken werden in einem *Runner*-Projekt verpackt. Dieses Projekt wird in die Installationsdateien *APK* (Android) und *IPA* (iOS) eingebaut [25]. In Abbildung 2.3 ist dieser Ablauf vereinfacht zu sehen.

[13] `https://developer.android.com/ndk` (06.08.2021)
[14] `https://llvm.org/` (06.08.2021)

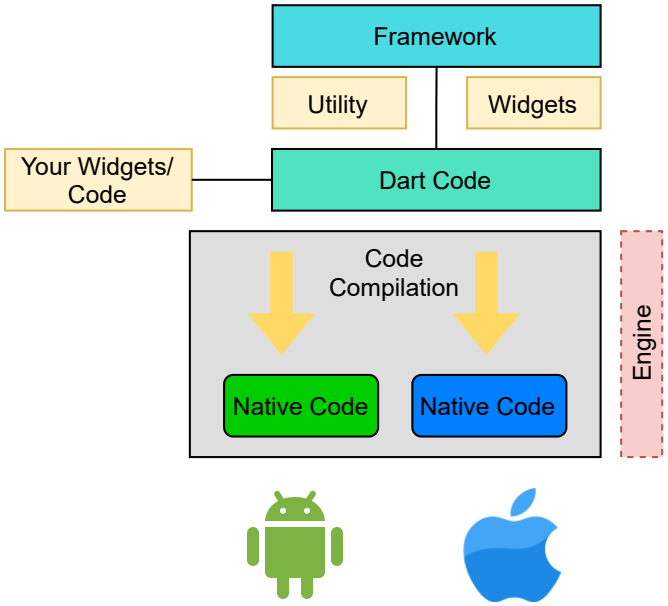


Abbildung 2.3: Ablauf der Kompilierung in nativen Plattformen, eigene Darstellung nach [11]

Da Dart über einen integrierten JavaScript-Compiler verfügt, wird die Engine und der geschriebene Dart Code in eine Web-Applikation kompiliert. Die Web-Applikation basiert dabei auf Webtechnologien wie HTML, CSS und JavaScript [25].

Widgets: Flutter selbst bietet keinen UI-Design-Editor, sondern die UI-Elemente werden über Code beschrieben. Wie weiter oben erwähnt, verwendet Flutter nicht die nativen UI-Komponenten der Betriebssysteme vom Gerät, sondern bietet eigene Widgets an. Die Widgets sind eine Beschreibung jedes UI-Elements mit ihrem Aussehen und ihrem Status (State). Der Status beinhaltet die Daten bzw. Informationen, die ein Benutzer eingibt oder die geladen werden. Die Entwickler können die in dem Framework vorgefertigten Widgets nutzen. Die Designs von den Widgets sind dem Android Material- und iOS Cupertino-Design angelehnt [25]. Das Hauptwidget wird dabei durch die Komposition grundlegender anderer Widgets aufgebaut. Die untergeordneten Widgets werden in der *build()*-Methode definiert. Der Benutzer baut somit eine Widget-Baumstruktur in Form von einem Code auf.

Das Framework baut anschließend rekursiv ein einziges Widget, welcher durch die Render-Engine dargestellt wird. Sobald sich der Zustand eines Widgets ändert,

baut sich die Widget-Baumstruktur in seiner Beschreibung neu auf und das Framework bestimmt die Unterschiede zum vorherigen Zustand. Diese Information ermöglicht der Render-Engine den Status von einem Zustand in den nächsten zu wechseln und darzustellen [25]. Die Widgets sind dabei entweder *Stateless* oder *Stateful*. Stateless Widget können ihren Zustand nur durch externen Dateninput ändern, Stateful Widgets hingegen haben die Möglichkeit, ihren Zustand mit jedem Aufruf zu ändern.

Generell können Widgets in drei Kategorien unterteilt werden. Diese Unterteilung erfolgt in die sichtbaren, nicht sichtbaren und dem Container. Die sichtbaren Widgets ermöglichen die Interaktion mit dem Benutzer durch Input und Output. Dazu gehören unter anderem *Text*, *AppBar* und *ElevatedButton* Widgets. Die nicht sichtbaren Widgets wie *Row*, *Column* und *ListView* ermöglichen das Layout und die Kontrolle der Positionen der sichtbaren Widgets. Die letztere Art ist das *Container*-Widget, mit dieser können rechteckige Elemente mit einem Rahmen, Schatten, Hintergrund aber auch Ränder und Einschränkungen mit der Größe des Inhaltes erstellt werden [25]. Eine Übersicht aller Widgets sind auf der Flutter Internetseite[15] zu finden. Im Listing 2.4 ist ein Programmbeispiel in Flutter zu sehen.

[15] `https://flutter.dev/docs/development/ui/widgets` (06.08.2021)

```
1  import 'package:flutter/material.dart';
2
3  void main() {
4    runApp(
5      Container(
6        decoration: BoxDecoration(
7          border: Border.all(color: Colors.blue, width: 3),
8          borderRadius: BorderRadius.circular(12),
9        ),
10       child: const Image(
11         image: NetworkImage(
12             'https://flutter.github.io/assets-for-api-docs/assets/widgets
                   /owl.jpg'),
13       ),
14     ),
15   );
16 }
```

Listing 2.4: Programmbeispiel in Flutter

Beim Starten der Applikation wird die `main` Funktion ausgeführt (Zeile 3). Diese beinhaltet eine vordefinierte `runApp` Funktion (Zeile 4), die es ermöglicht, dass das geschriebene Widget als App gestartet wird. In diesem Fall ist das Widget ein Container (Zeile 5-14) mit einem abgerundeten blauen Rahmen (Zeile 6-9), welches ein Bild aus dem Internet (Zeile 10-13) beherbergt.

Besonderheiten: Außerhalb der vorgefertigten Marterial und Cupertino Widgets, bietet Flutter den Entwicklern die Möglichkeit, eigene Widgets mit personalisiertem Design zu erstellen. Das Framework ermöglicht zugleich auf die *Runtime* des Betriebssystems zuzugreifen, wodurch dem Entwickler die Macht gegeben wird, je nach Betriebssystems des Gerätes (iOS, Android) unterschiedliche Programmteile auszuführen [25]. Damit kann der Entwickler beispielsweise mit einem Projekt eine Applikation mit unterschiedlichem Design entwickeln. Die Embedder API erlaubt es unter anderem dem Entwickler die Engine als Bibliothek zu verwenden [11]. Der Debugger Modus verwendet die VM, um den Dart-Code auszuführen. Dadurch ist die Applikation etwas langsamer, aber das Framework ermöglicht, dass die Widget-Baumstruktur von Stateful Widgets automatisch erneuert wird, ohne die Applikation neu starten zu müssen. Diese Funktion wird in Flutter *Stateful Hot-Reload* ge-

nannt [25].

Flutter im Vergleich mit anderen Cross-Platform Frameworks: Eine Umfrage aus dem Jahr 2020 von *JetBrains* ergab, dass Entwickler im Bereich plattformübergreifende mobile Frameworks zu 42% React Native, 39% Flutter und 28% Ionic nutzen [32]. Aus diesem Grund wird ein Vergleich nur mit diesen beiden Frameworks als Referenz genommen.

Das *Ionic* Framework ist wie Flutter eine Open-Source UI Cross-Platform Software. Anders als bei Flutter werden in dieser Hybride-Webanwendungen für iOS, Android, Desktop und Web entwickelt. Ionic bietet eine Reihe von UI-Webkomponenten, die auf Webtechnologien wie HTML, CSS und JavaScript basieren, mit deren Hilfe mit Ionic Benutzeroberflächen erstellt werden können [12]. Ebenso ermöglicht Ionic durch den Compiler *Stencil*[16] eigene Webkomponenten zu erstellen [40]. Ionic gibt dem Entwickler die Freiheit, sich für Font-End-Frameworks wie *Angular*, *React* oder *Vue* zu entscheiden. Für mobile Geräte wird die Applikation nicht wie in Flutter durch die Engine gerendert, sondern wird in einen nativen Container mithilfe von Frameworks wie *Apache Cordova*[17] oder *Capacitor*[18] verpackt [12]. In Abbildung 2.4 ist die Architektur von Apache Cordova abgebildet, welche die Funktionsweise der Kommunikation zwischen Gerät und Anwendung verdeutlicht. Die Frameworks bilden eine WebView in der es möglich ist, die Webapplikation zu hosten. Zugleich bieten sie native Plugins an, die den Zugriff auf die API der Geräte ermöglichen.

Desktopanwendungen hingegen werden meist in *Electron* verpackt [12]. Da es sich hierbei um eine Webanwendung handelt, kann die Anwendung im Browser genutzt werden. Ähnlich wie beim Hot-Reload von Flutter kann der Entwickler Änderungen direkt im Browser durch die *Live-Reload* Funktion anzeigen [12].

[16] `https://stenciljs.com/` (06.08.2021)
[17] `https://cordova.apache.org/` (06.08.2021)
[18] `https://capacitorjs.com/` (06.08.2021)

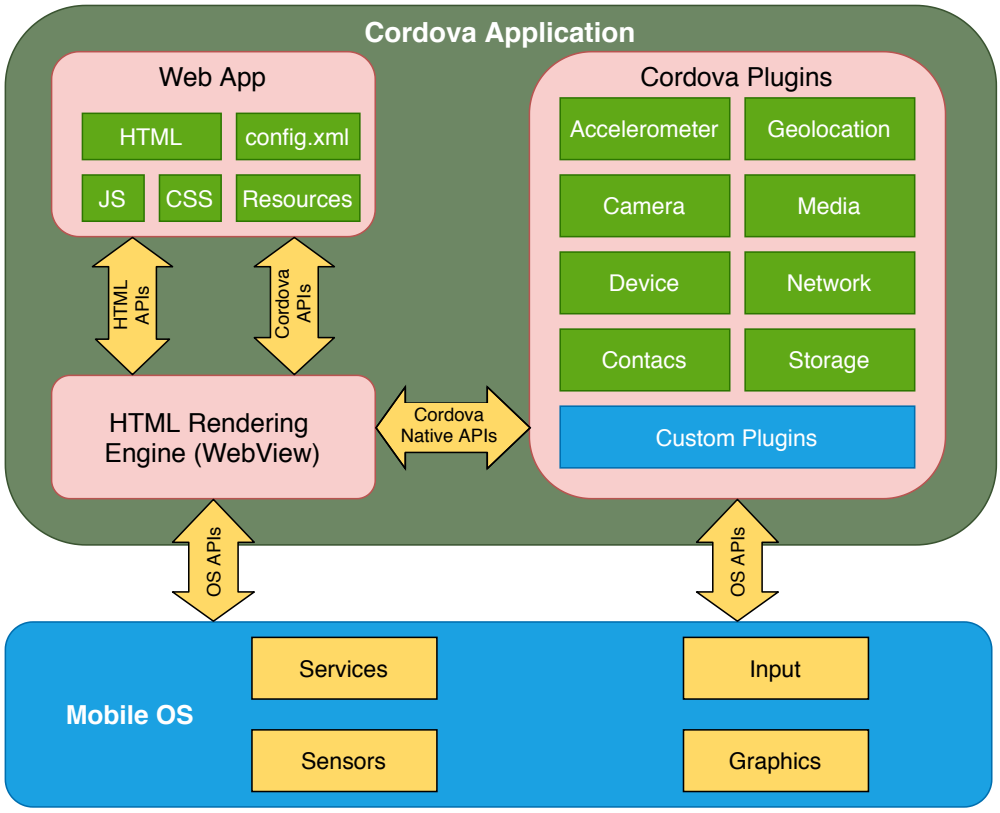


Abbildung 2.4: Apache Cordova Architektur [19]

Das Open-Source UI-Framework *React Native* nutzt JavaScript und die Bibliothek *React.js* um native iOS und Android Applikationen zu entwickeln. Die UI wird ähnlich wie Flutter durch die Komposition von UI-Komponenten erstellt. Das Framework bietet eine Reihe von Komponenten an, welche mit *JSX* Syntax erstellt werden und vom Design her angepasst werden können [15]. Die JSX Syntax ist die Erweiterung der JavaScript Sprache mit XML für React.js. Die Erweiterung ermöglicht das Einbauen von HTML in die JavaScript Syntax.

Mithilfe von APIs wird der Zugriff auf die nativen Funktionen wie die Kamera des Gerätes ermöglicht [15]. Bei der Kompilierung der APK- und IPA Datei wird nicht wie in Flutter das gesamte Projekt nativ kompiliert, sondern nur die erstellten UI-Komponenten. Diese werden je nach Plattform in die dementsprechenden nativen Widgets kompiliert [2]. Die APK- und IPA Datei zum Installieren der Applikation bestehen somit aus den nativen Widgets der entsprechenden Plattform und dem restlichen Code, die der Funktionslogik entspricht. Die Applikation läuft auf den Ge-

räten in zwei Threads. Im ersten Thread werden die nativen Widgets angezeigt und zugleich durch die Event-Listener abgehört. Der restliche Programmcode läuft im zweiten Thread auf einer JavaScript VM. Als VM wird die auf iOS vorhandene JavaScriptCore Engine benutzt. Auf Android ist diese nicht vorhanden, wird jedoch mit der APK zusammen mitgeliefert. Beim Auslösen eines Events wird der Nachrichtenaustausch zwischen den Threads durch die React Native Bridge realisiert. Die Bridge kompiliert den Code des ausgelösten Events (Java/C++) in JS-Code und leitet es an den anderen Thread weiter. Umgekehrt wird auch der JS-Code in Java/C++ kompiliert und an den ersten Thread weiter geleitet [2]. In Abbildung 2.5 ist der Ablauf visuell zu sehen.

Abbildung 2.5: Funktionsweise von React Native [2]

3 Verwandte Arbeiten

In diesem Kapitel werden zwei ähnliche Projekte des *Instituts für Datenbanken und Informationssysteme (DBIS)* der Universität Ulm vorgestellt.

3.1 Track Your Tinnitus

Im Rahmen der Diplomarbeit von *Jochen Herrmann* wurde die mobile Applikation *Track Your Tinnitus*[1] entwickelt. Diese wurde von Studenten im weiteren Verlauf in Projekten[2] weiterentwickelt und angepasst.

Für Tinnitus Patienten ermöglicht das Projekt, die eigene Wahrnehmung von Tinnitus anhand von digitalen Fragebögen zu erfassen und zu analysieren. Dies wird umgesetzt, in dem Patienten Fragen wie der Lautstärke vom Tinnitus, die empfundene Tonhöhe und negative Auswirkungen auf das alltägliche Leben beantworten. Anschließend erhält der Patient eine personalisierte grafische Visualisierung als Rückmeldung seiner Angaben. Zusätzlich können Erinnerungen in Form von Benachrichtigungen (Notifications) für repetierende Fragebögen eingestellt werden [29]. Zugleich ermöglicht die Applikation die Ansammlung von Daten, welche im Bereich der Forschung von Tinnitus nützlich sein kann. Unter anderem wurde die Applikation in verschiedenen Studien eingesetzt [41, 42, 43, 44, 45, 46, 48].

Die Applikation wurde für die Betriebssysteme iOS[3] und Android[4] in den jeweiligen App Stores veröffentlicht.

[1] Jochen Herrmann, `https://www.trackyourtinnitus.org/de/` Version 2013 (09.08.2021)
[2] `https://www.uni-ulm.de/in/iui-dbis/forschung/laufende-projekte/trackyourtinnitus/` (09.08.2021)
[3] `https://apps.apple.com/de/app/track-your-tinnitus/id787178122` (09.08.2021)
[4] `https://play.google.com/store/apps/details?id=com.jochenherrmann.trackyourtinnitus` (09.08.2021)

3.2 Track Your Stress

Das Projekt *Track Your Stress* ähnelt dem Projekt *Track Your Tinnitus* und ist eine *Crowdsensing Plattform*[5]. Mit diesem ist es möglich, den Stresslevel des Benutzers zu erfassen und anschließend die Daten auszuwerten und zu speichern.

Die ersten Projekte wurden in Form einer mobilen Applikation für die Betriebssysteme iOS und Android im Rahmen zwei Bachelorthesen entwickelt [28, 49]. Später wurde eine Web-Applikation[6] (2019) im Rahmen einer Masterthesis entworfen [13]. In diesen können Benutzer an einer Stress-Studie teilnehmen, um die entsprechenden Fragebögen auszufüllen. Nach einer anschließenden Evaluierung wird dem Benutzer das Ergebnis visuell dargestellt. Ebenfalls ist es dem Benutzer möglich, für wiederholende Fragebögen sich Erinnerungen in Form von Benachrichtigungen einzustellen [49].

[5] Datenerfassung und -auswertung einer Personengruppe die voneinander unabhängig und geografisch verteilt sind. Die Daten werden durch die Sensoren von mobilen Geräten gesammelt.
[6] `https://www.trackyourstress.org/home` (09.08.2021)

4 Anforderungsanalyse

Die zu realisierende Applikation heißt *Track Your Health (TYH)*. In diesem Kapitel werden die Anforderungen an die Applikation spezifiziert und erläutert. Dabei unterteilt sich das Kapitel in die Abschnitte Anwendungsfälle, funktionale und nichtfunktionale Anforderungen.

4.1 Anwendungsfälle

Ein Überblick über die wesentlichen Systemfunktionen aus der Benutzersicht ist in Abbildung 4.1 anhand eines Anwendungsfalldiagramms dargestellt.

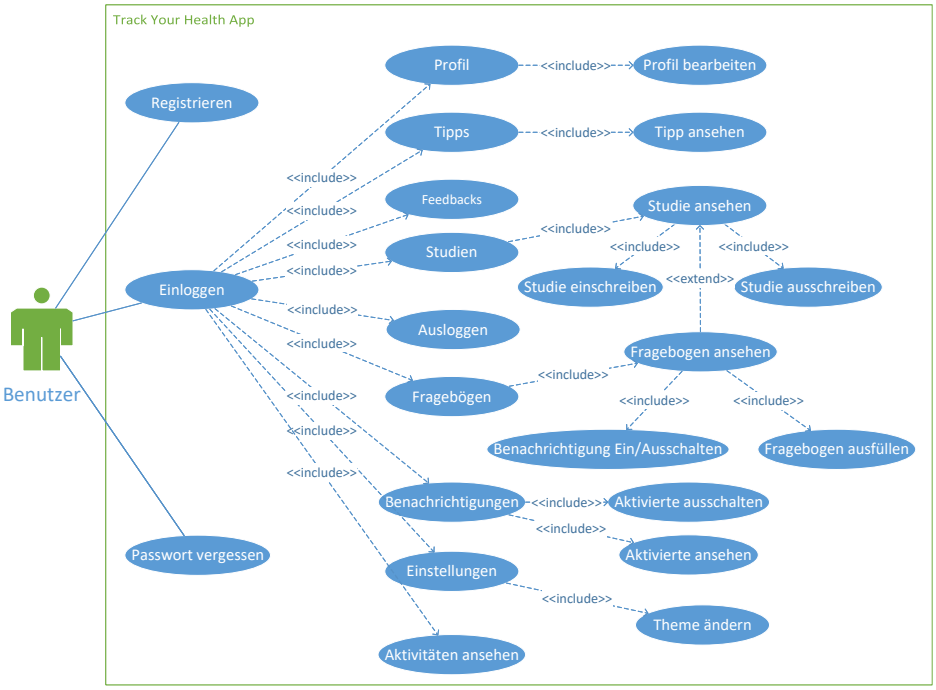


Abbildung 4.1: Anwendungsfalldiagramm der TYH Applikation

Beim Starten der Applikation hat der Benutzer die Möglichkeiten, sich ein Benutzerkonto zu erstellen, sich anzumelden oder durch die Angabe der E-Mail-Adresse das Passwort zurücksetzen. Nach der erfolgreichen Anmeldung kann der Benutzer sich abmelden und auf die restlichen Systemfunktionen zugreifen. Dazu gehören unter anderem das Ansehen und Bearbeiten vom Benutzerprofil. Laufende Studien werden unter *Studien* aufgelistet werden. Durch das Anklicken einer Studie werden die ausführlichen Informationen über diese angezeigt. Dadurch hat der Benutzer in der detaillierten Ansicht folgende Optionen:

- Sich Informationen über die Studie verschaffen
- Ein- und Ausschreiben aus der Studie
- Zusammenhängende Fragebögen erreichen

Um die Fragebögen ausfüllen zu können, muss der Benutzer in der jeweiligen Studie eingeschrieben sein. Durch das Einschreiben sind die entsprechenden Fragebögen im Reiter *Fragebögen* gelistet. Der Reiter ermöglicht den direkten Zugriff auf die Fragebögen, was schneller und einfacher ist, als diese über die entsprechenden Studien zu erreichen. Nach dem Auswählen eines Fragebogens kann der Benutzer folgende Aktionen in der detaillierten Ansicht durchführen:

- Detaillierte Informationen erhalten
- Im Falle eines wiederkehrenden Fragebogens Benachrichtigungen aktivieren
- Ausfüllen des Fragebogens beginnen

Die aktivierten Benachrichtigungen sind unter *Benachrichtigungen* zu finden und können von dort aus deaktiviert werden. Unter *Feedbacks* hat der Benutzer die Möglichkeit, Evaluationen über seine bereits ausgefüllten Fragebögen zu erhalten. Die Historie der Aktivitäten wie das Ändern von Profilinformationen oder Ein- und Ausschreiben von Studien sind unter *Meine Aktivitäten* zugänglich. In den Einstellungen steht dem Benutzer die Möglichkeit, das Farbdesign (Theme) der Anwendung anzupassen.

4.2 Funktionale Anforderungen

Die *funktionalen Anforderungen (FR)* spezifizieren die Funktionen, die das System erfüllen muss. Diese beinhaltet unter anderem die genaue Definition der möglichen Eingaben mit den jeweiligen Einschränkungen [21]. In diesem Abschnitt wird mit API die *Track Your Health API*[1] bezeichnet, welche die Daten zur Verfügung stellt. Diese wird in Abschnitt 6.1 ausführlicher vorgestellt. Mit den Anwendungsfällen aus Kapitel 4 ergeben sich folgende funktionale Anforderungen.

FR01 **Registrierung**

In der App kann der Benutzer ein Konto erstellen. Dabei werden folgende Eingaben benötigt:

- Benutzername
- Gültige E-Mail-Adresse
- Passwort mit mindestens 8 Zeichen
- Passwort Wiederholung

Falls die Eingaben fehlerhaft sind, wird dem Benutzer eine Fehlermeldung angezeigt.

FR02 **Passwort vergessen**

Der Benutzer kann durch Eingabe der E-Mail-Adresse das Passwort vom erstellten Benutzerkonto zurücksetzen. Bei einer ungültigen oder nicht verifizierten E-Mail wird der Fehler durch eine entsprechende Fehlermeldung mitgeteilt.

[1] https://api.dummy.trackyourhealth.net/dingodocs/v1.html (15.08.2021)

FR03 **Anmeldung**

In der App kann sich mit einem Benutzerkonto angemeldet werden. Dafür wird die E-Mail-Adresse und das Passwort benötigt. Bei falscher Eingabe oder einem Serverfehler wird eine entsprechende Fehlermeldung angezeigt. Nach einer erfolgreichen Anmeldung wird der empfangene *JSON Web Token (JWT)* im System gespeichert und bei weitere Anfragen verwendet. Falls der Benutzer sich nicht abmeldet, wird der gespeicherte JWT-Token bei einem Neustart der Applikation wiederverwendet.

FR04 **Abmeldung**

Der Benutzer kann sich aus seinem Benutzerkonto abmelden. Bei einer Abmeldung gelangt er zurück auf die Startseite und das System löscht das gespeicherte JWT-Token vom Gerät.

FR05 **Aktualisierung vom Token**

Das System aktualisiert alle 8 Minuten den JWT-Token, dabei wird der neue Token auf dem Gerät gespeichert und für die weiteren Anfragen eingesetzt.

FR06 **Vergabe von Bewertungen (Likes)**

Die App ermöglicht dem Benutzer das Bewerten (Likes) von einzelnen Tipps, Studien und Fragebögen. Dabei hat die Bewertung die Stufen 1 bis 5, wobei 5 die beste Bewertung ist.

FR07 **Tipps**

Das System zeigt dem Benutzer die auf der API liegenden Tipps an. Dabei wird zuerst eine Übersicht aller Tipps dargestellt. Der Benutzer hat durch das Auswählen eines Tipps die Möglichkeit, genauere Details darüber zu erhalten.

FR08 **Tipp-Detailansicht**

In der Tipp-Detailansicht wird dem Benutzer genauere Informationen über den jeweiligen Tipp angezeigt. Die Informationen gliedern sich dabei in das Ziel, die Beschreibung und die Erklärung. Zusätzlich hat der Benutzer in dieser Ansicht die Möglichkeit, den jeweiligen Tipp zu bewerten.

FR09 **Studien**

Das System zeigt dem Benutzer alle laufenden Studien auf der API an. Zusätzlich ist es möglich, eine einzige Studie mit näheren Details sich anzeigen zu lassen. Ebenfalls kann das System nur die eingeschriebenen Studien in einer Ansicht anzeigen.

FR10 **Studie einschreiben und verlassen**

Der Benutzer kann sich nur in Studien einschreiben, in welche er nicht eingeschrieben ist. Das System ermöglicht das Verlassen der Studie, falls der Ersteller der Studie dies zulässt.

FR11 **Studie-Detailansicht**

In der Studie-Detailansicht werden dem Benutzer genauere Informationen über die jeweilige Studie angezeigt. Diese beinhalten die Beschreibung, den Zeitraum und die entsprechenden Fragebögen. Zusätzlich kann der Benutzer die Studie bewerten. Ebenfalls wird der Einschreibungsstatus des Benutzers in der Studie angezeigt. Das Einschreiben und Verlassen erfolgt in dieser Ansicht.

FR12 **Fragebögen**

Das System zeigt alle aktiven Fragebögen auf der API an. Dabei hat das System zwei Arten, diese anzuzeigen. Die erste Ansicht beinhaltet alle Fragebögen für alle laufenden Studien, für die der Benutzer derzeit eingeschrieben ist. In der zweiten Ansicht findet der Benutzer die Fragebögen in den jeweiligen Studien. Zusätzlich ist es möglich, einen einzigen Fragebogen mit näheren Details anzeigen zu lassen.

FR13 **Benachrichtigungen aktivieren (Notifications)**

Das System ermöglicht für wiederkehrende Fragebögen dem Benutzer, sich Erinnerungen in Form von Benachrichtigungen zu aktivieren. Ebenfalls können aktivierte Benachrichtigungen deaktiviert werden.

FR14 **Fragebogen-Detailansicht**

Dem Benutzer werden in Fragebogen-Detailansicht die Beschreibung, der ausfüllbare Zeitraum, Anzahl der übrigen Versuche mit der Periode und die Berechtigungen, die benötigt werden, angezeigt. Das Bewerten eines Fragebogens ist ebenfalls möglich. Falls der Benutzer in der dazugehörenden Studie eingeschrieben ist und übrige Ausfüllversuche hat, kann das Ausfüllen des Fragebogens gestartet werden. Zusätzlich kann der Benutzer für wiederkehrende Fragebögen sich Erinnerungen in Form von Benachrichtigungen aktivieren. Aktivierte Benachrichtigungen können wiederum deaktiviert werden.

FR15 **Fragebogen ausfüllen**

Die App ermöglicht das Ausfüllen vom Fragebogen. Dabei werden die Fragen mit Antwortmöglichkeiten dem Benutzer angezeigt. Nach dem Ausfüllen aller Fragen erlaubt das System das Abschicken des Antwortbogens.

FR16 **Benachrichtigung Übersicht**

Das System zeigt dem Benutzer alle aktivierten Benachrichtigungen für die jeweiligen Fragebögen an. In der Übersicht ist es dem Benutzer möglich, einzelne Benachrichtigungen zu deaktivieren oder auf den entsprechenden Fragebogen zu gelangen.

FR17 **Profilverwaltung**

Das System ermöglicht dem Benutzer, Änderungen am Profil vorzunehmen. Diese beinhalten das Anpassen vom `Benutzernamen`, `Vornamen`, `Nachnamen`, `Geschlecht` und das Ändern vom `Passwort`. Zusätzlich ermöglicht das System das Löschen vom Konto.

FR18 **Aktivitäten anzeigen**

Die Applikation zeigt dem Benutzer in einer Übersicht die auf der API liegenden persönlichen Aktivitäten an.

FR19 **Feedbacks**

Alle verfügbaren Feedbacks zu ausgefüllten Fragebögen können in einer Übersicht angesehen werden. Zusätzlich ist es ihm möglich, ein einziges Feedback auszuwählen, um die jeweilige Evaluation zu betrachten.

FR20 **Evaluationen**

Dem Benutzer wird die Evaluation des ausgewählten Feedbacks angezeigt. Falls die Evaluation keine Informationen enthält, wird der Benutzer darauf aufmerksam gemacht.

FR21 **Einstellungen**

Die App erlaubt dem Benutzer, die Darstellung aus einer vordefinierten Menge an Optionen in Bezug auf die Farben (Theme) zu wechseln. Die Änderung wird dabei auf dem Gerät gespeichert.

FR22 **Über die App**

Dem Benutzer werden Hintergrundinformationen sowie Entwicklerinformationen der App angezeigt.

4.3 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen (NFR) definieren die Qualitätsattribute der gewünschten Funktionen an das System. Dazu zählt das Ausführungsverhalten in Bezug auf Zeit oder Speicher, aber auch die Wartbarkeit, Zuverlässigkeit, Ausfallsicherheit und Robustheit des Systems [21]. Daraus ergeben sich folgende nicht-funktionalen Anforderungen an die mobile Anwendung.

NFR01 **Benutzerfreundlichkeit**

Die Applikation muss intuitiv sein, damit unerfahrene Benutzer die Applikation ebenfalls benutzen können. Diese Anforderung kann durch das Ermöglichen von wichtigen Funktionalitäten wie das Navigieren über unterschiedlichem Wege und dem visuellen Feedback nach relevanten Aktionen erreicht werden.

NFR02 **Effizienz**

Die Anwendung muss effizient sein, damit das Gerät nicht zu sehr belastet wird und der Benutzer nicht langen Wartezeiten ausgesetzt wird.

NFR03 **Verlässlichkeit**

Das System muss dem Benutzer jederzeit korrekte Daten zur Verfügung stellen, unabhängig der Systemlaufzeit. Ebenfalls muss auf jede Eingabe reagiert werden. Bei einem Fehler muss die Ursache in einer klar verständlichen Meldung angezeigt werden.

NFR04 **Erweiterbarkeit**

Die Anwendung muss leicht zu erweitern sein. Die Erweiterung kann das Implementieren von zusätzlichen Funktionen oder das Hinzufügen von verschiedenen Sprachen sein, ohne die Grundarchitektur ändern zu müssen.

NFR05 **Portierbarkeit**

Die Applikation muss für den Benutzer auf iOS und Android Geräten einfach installierbar sein.

NFR06 **Wartbarkeit**

Die Applikation muss ohne größeren Aufwand zu warten sein. Dazu zählen das Ändern und Anpassen von Komponenten ohne weitere Hindernisse.

NFR07 **Systemunabhängige Benutzung**

Die Applikation muss auf unterschiedlichen Geräten unabhängig vom Betriebssystem identisch aussehen und dieselben Funktionen unterstützen.

5 User-Interface-Entwurf

Wie in Unterabschnitt 2.4.3 erwähnt, werden in Flutter fertige Widgets im Design von Android *Material Design* und iOS *Cupertino* mitgeliefert. In diesem Projekt wurde sich für das Material Design entschieden, da diese im Vergleich zu Cupertino eine größere Anzahl an unterschiedlichen Widgets anbietet. Dementsprechend werden in diesem Kapitel Designentscheidungen und die damit entstandenen Prototypen der App vorgestellt.

5.1 Styleguide und Themes

Das Ziel eines guten Designs einer Anwendung ist die maximale Benutzerfreundlichkeit (Usability) für den Benutzer. Dabei sollten Entwickler Gestaltungsregeln berücksichtigen, um die Usability zu erhöhen. Diese Gestaltungsregeln können aus der praktischen Erfahrung, formale Studien, Forschung und persönlicher Meinung entstehen [47]. Damit Entwickler eine detaillierte Leitlinie für Gestaltungsregeln haben, sind unterschiedliche User-Interface-Guidelines entstanden. Durch die Einhaltung der Guidelines wird den Entwicklern ermöglicht, Applikationen mit einer einheitlichen Oberfläche zu entwickeln. Diese Guidelines sind eine Ansammlung von Gestaltungsregeln, die sich mit der Zeit behauptet haben. Eines dieser Guidelines sind die *iOS Human Interface Guidelines*[1] von Apple für die Plattformen macOS, iOS, watchOs und tvOS. Google hingegen hat sein *Material Design*[2] für Android, iOS, Flutter und Webseiten. Beide dieser Guidelines werden mit der Zeit aktualisiert, wodurch sich Änderungen oder Erweiterungen vorfinden. Dementsprechend sollten Entwickler immer wieder die Guidelines evaluieren.

[1] `https://developer.apple.com/design/human-interface-guidelines/` (17.08.2021)
[2] `https://material.io/design` (17.08.2021)

Wie am Anfang erwähnt, wurde in dieser Arbeit sich für die Widgets im Android Design entschieden, deshalb wurde das Material Design als Grundlage für Designentscheidungen genutzt. Die Guideline wurde nicht konsequent eingehalten, da Flutter das Erstellen von eigenen Widgets, aber auch das Einsetzen von Widgets von anderen Entwicklern ermöglicht.

Farben/Themes: Um Farben sinnvoll auf die Benutzeroberfläche anzuwenden, bietet das Material Design ein Farbsystem mit unterschiedlichen Farbpaletten an. Dabei sind die Farben in der Palette harmonisch miteinander und lassen sich in Primär- und Sekundärfarben untergliedern. Die Primärfarben sind die am meisten eingesetzten Farben, welche in den Komponenten wie beispielsweise in *App Bars* oder *Buttons* auftauchen. Die Sekundärfarben hingegen bieten die Möglichkeit, bestimmte Elemente hervorzuheben. Die Sekundärfarben eignen sich unter anderem für Auswahlkontrollen wie *Schieberegler (Slider)* und *Schalter (Switches)*, aber auch für *Fortschrittsbalken (progress bars)* [26].

In dieser Arbeit wurden 5 verschiedene Themes realisiert, eines davon ist das fertige *Dark Theme*[3] des Material Design. Die restlichen Farbpaletten wurden ebenfalls aus dem Material Design[4] entnommen. Ein Beispiel der eingesetzten Farbpalette ist in Abbildung 5.1 abgebildet. Auf der rechten Seite ist die jeweilige Farbdefinition im Hexadezimalsystem zu sehen und wird mit einem # eingeleitet. Die Werte auf der linken Seite repräsentieren die relative Helligkeit/Dunkelheit oder den Farbton der Farbe, wobei 50 der hellste und 900 der dunkelste Wert ist. Zugleich repräsentieren die genutzten Schriftfarben die Schriftfarbe, die in Komposition mit den Hintergrundfarben eingesetzt werden. Die erstellten Themes sind in Tabelle 5.1 gelistet.

[3] `https://material.io/design/color/dark-theme.html#anatomy` (16.08.2021)

[4] `https://material.io/design/color/the-color-system.html#color-usage-and-palettes` (16.08.2021)

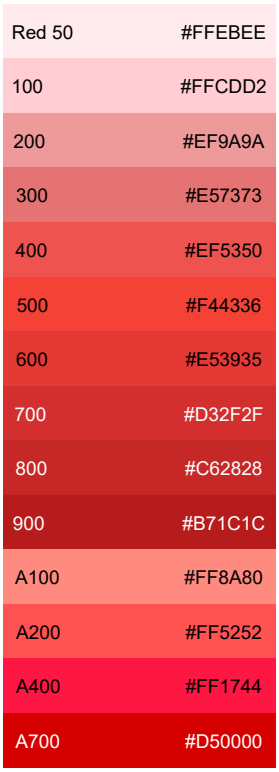


Abbildung 5.1: Material Design Farbpalette Red [26]

Name	Farbpalette	Hintergrund	Primärfarbe	Sekundärfarbe
CLASSIC	Red	#FFEBEE	#B71C1C	#FFCDD2
OCEAN BLUE	Light Blue	#E1F5FE	#0277BD	#4FC3F7
LIME	Lime	#F9FBE7	#CDDC39	#DCE775
WEED GREEN	Green	#E8F5E9	#2E7D32	#66BB6A
MATERIAL DARK	Dark theme	#121212	#BB867FC	#B875E3

Tabelle 5.1: Erstellte Themes, Farben nach [26]

Typografie: Die Typografie ist ein wichtiger Bestandteil der Usability. Dabei sollen die Texte für den Benutzer ästhetisch und lesbar gestaltet sein. Dies kann durch Verwendung von verschiedenen Schriftarten, -größen und -stile sowie Layout Einschränkungen erfolgen [34]. Google veröffentlichte 2014 mit dem Material De-

sign zusammen mit der überarbeiteten Version der hauseigenen Schriftart *Roboto*[5]. Diese empfiehlt sich besonders für mobile Geräte aufgrund der Buchstabenbreite, da diese den Lesefluss des Benutzers unterstützt. Deshalb wurde die Roboto-Schriftart im Projekt eingesetzt. In Tabelle 5.2 sind die eingesetzten Schriftgrößen mit den Typskalen gelistet. Die Werte wurden aus dem Material Design Guideline[6] entnommen. Dabei ist die *Spationierung* ein typografischer Begriff, welche die horizontalen Buchstabenabstände angibt. Diese wird auch Schriftlaufweite genannt.

Text-Element	Schriftstärke	Größe	Spationierung
Headline 1	Light	96	-1.5
Headline 2	Light	60	-0.5
Headline 3	Regular	48	0
Headline 4	Regular	34	0.25
Headline 5	Regular	24	0
Headline 6	Medium	20	0.15
Subtitle 1	Regular	16	0.15
Subtitle 2	Medium	14	0.1
Body 1	Regular	16	0.5
Body 2	Regular	14	0.25
Button	Medium	14	1.25
Caption	Regular	12	0.4
Overline	Regular	10	1.5

Tabelle 5.2: Eingesetzte Typskalen für die Roboto-Schriftart nach [26]

5.2 Prototypen

Nach der Anforderungsanalyse kann durch Prototypen eine frühzeitige, einfache und kostengünstige Erstellung einer Benutzeroberfläche (UI) realisiert werden. Da

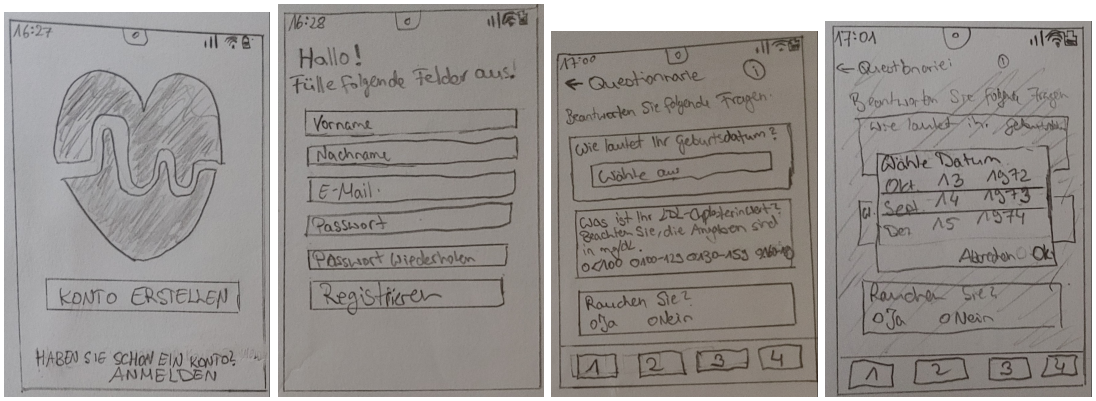
[5] `https://fonts.google.com/specimen/Roboto` (16.08.2021)
[6] `https://material.io/design/typography/the-type-system.html#type-scale` (16.08.2021)

die Erstellung von Prototypen schneller erfolgt als das Implementieren, können Änderungen im Entwurf leichter überarbeitet werden. Somit müssen keine zeitaufwendigen Überarbeitungen im implementierten System unternommen werden. Zusätzlich werden Lösungen und Ideen der Anforderungen konkretisiert, wodurch frühzeitige Probleme und Unklarheiten der Anforderungsanalyse aufgedeckt werden können. Dieser Aspekt ist vor allem bei kundenorientierten Produkten wichtig, da die Prototypen als Zwischenergebnisse vorgezeigt werden können. Somit bekommt der Kunde ein *look and feel* vom Produkt und kann sich aktiv am Entwurf beteiligen [34].

Bei der Erstellung von Prototypen gibt es zwei unterschiedliche Ansätze. Erstere sind die *Low-Fidelity Prototypen*. Bei diesen steht die Design-Erkundung und Kommunikation mit dem Kunden im Vordergrund, um die genauen Anforderungen an das System auszuarbeiten. Die formale Bewertung des Designs wird somit vernachlässigt [35]. In Unterabschnitt 5.2.1 ist ein Low-Fidelity Prototyp in Form von Papierskizzen zu sehen. Mithilfe von diesen entstanden die ersten Konzepte der UI. Den zweiten Ansatz bieten die *High-Fidelity Prototypen*. Diese sind detailreicher im Design, zusätzlich kann der Benutzer mit dem System bis zu einem gewissen Grad interagieren, womit ein Usabilityfeedback eingeholt werden kann [35]. Der erstellte High-Fidelity Prototyp wird in Unterabschnitt 5.2.2 vorgestellt. Durch den erstellten High-Fidelity Prototypen konnte die Funktionalität und die Anforderung der Anwendung konkretisiert und zugleich ein entsprechendes UI konzipiert werden.

5.2.1 Papierskizzen

In Abbildung 5.2 ist der allgemeine Aufbau vom ersten Entwurf der UI zu sehen. Die Anmeldung und Registrierung (Abbildung 5.2b) sind dabei über den Startbildschirm (Abbildung 5.2a) erreichbar. Dieser Ansatz wurde jedoch im High-Fidelity Prototypen aufgrund von Usability-Aspekten verworfen. Nach einer erfolgreichen Anmeldung gelangt der Benutzer auf ein Fenster mit vier Tabs. Diese Tabs ermöglichen das Navigieren in die verschiedenen Funktionen bzw. Fenster, diese ist in Abbildung 5.2c zu sehen. Einer der Funktionen ist das Ausfüllen von Fragebogen, welches in Abbildung 5.2c dargestellt ist. Dabei besitzt der Fragebogen verschiedenen Fragetypen mit den jeweiligen Antwortmöglichkeiten. Eine Antwortmöglichkeit stellt die Eingabe des Geburtsdatums mithilfe eines Datum-Picker dar (Abbildung 5.2d).



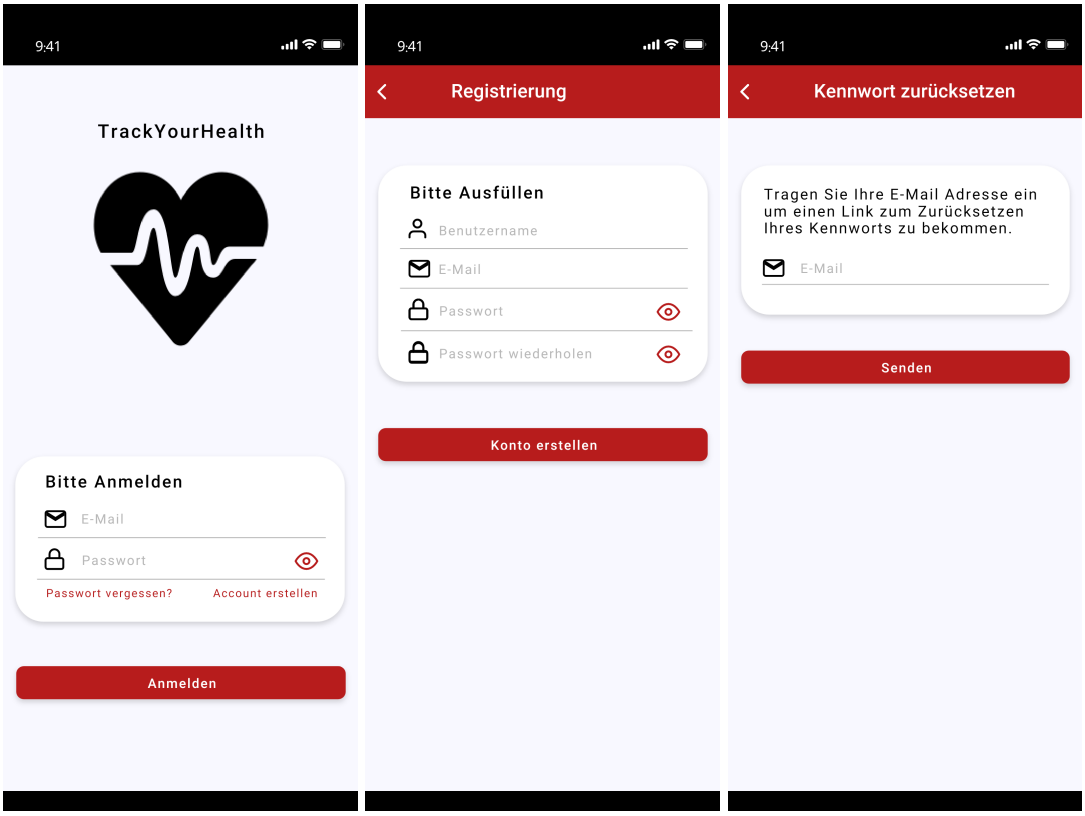
(a) Startbildschirm (b) Registrierung (c) Fragebogen (d) Date-Picker

Abbildung 5.2: Papierskizzen

5.2.2 High-Fidelity Prototyp

Der High-Fidelity Prototyp wurde mit Grafiksoftware *Adobe XD*[7] erstellt. Diese Software ermöglicht es, grafische Oberflächen für Webseiten oder Apps zu errichten. Die erstellten Oberflächen lassen sich dabei verknüpfen, wodurch die Interaktion in Form von Klicks simuliert wird. Das Ziel dieses Prototyps war es, ein nahezu vollständiges Design entsprechend der Anforderungen zu erstellen. Ebenfalls sollten die Anforderungen konkretisiert und mögliche Unklarheiten aufgeklärt werden. Durch die vorläufige Evaluation sollten Änderungen in der Implementierungsphase vorgebeugt oder viel mehr vermieden werden, damit während der Implementierung ein größerer Fokus auf die qualitative Umsetzung gelegt werden kann. Im Folgenden werden Ausschnitte vom erstellten Prototypen vorgestellt. Der vollständige interaktionsfähige Prototyp ist unter `https://xd.adobe.com/view/87a394de-26ba-40b1-8620-f3b0e4092982-d2f0/?fullscreen&hints=off`[8] zu finden.

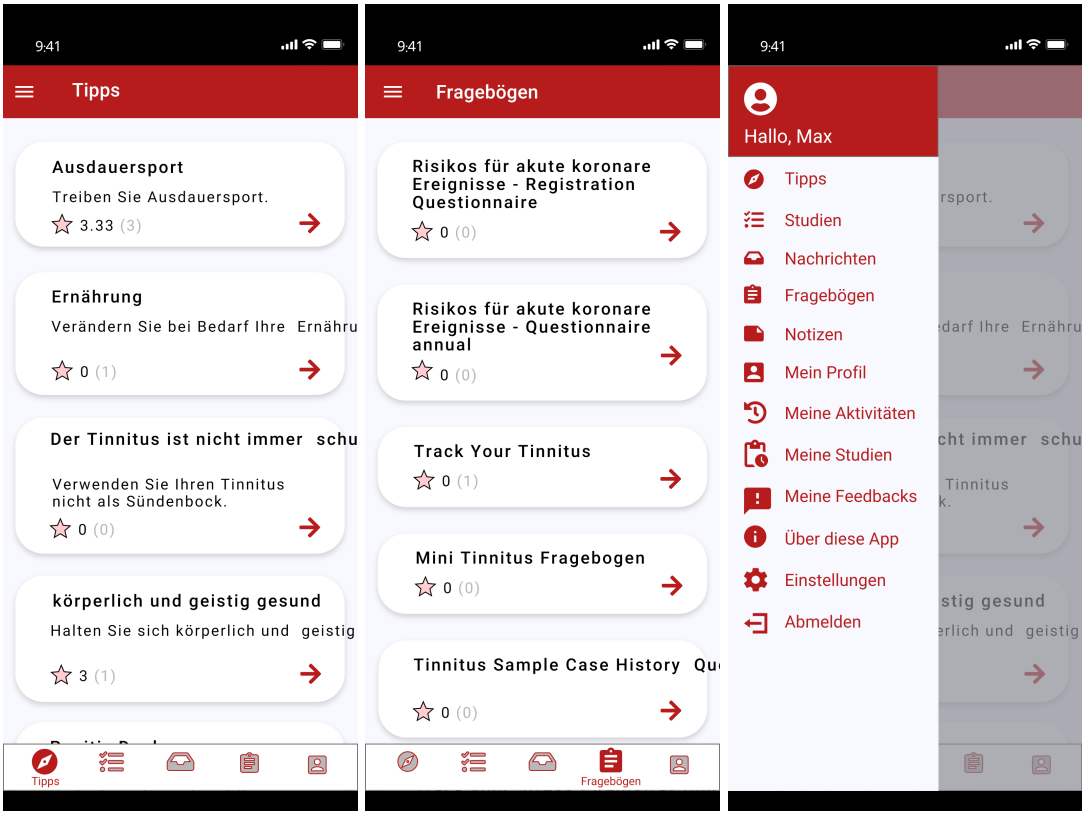
[7] `https://www.adobe.com/de/products/xd.html` (16.08.2021)
[8] 19.08.2021



(a) Startbildschirm (b) Registrierung (c) Kennwort zurücksetzen

Abbildung 5.3: Prototyp Anmelde-Ansichten

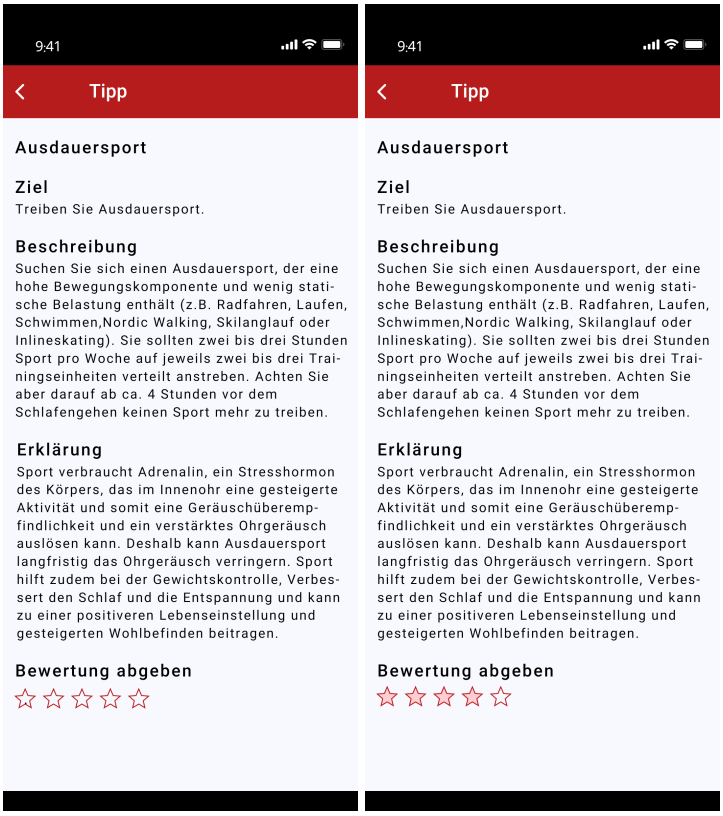
Abbildung 5.3 zeigt die Aktionen, die beim Starten der App möglich ist. Zu diesen zählt das Anmelden (Abbildung 5.3a) oder auf die Registrierung und Kennwort zurücksetzten Ansicht zu gelangen.



(a) Tipps (b) Fragebögen (c) Drawer

Abbildung 5.4: Prototyp Hauptnavigation

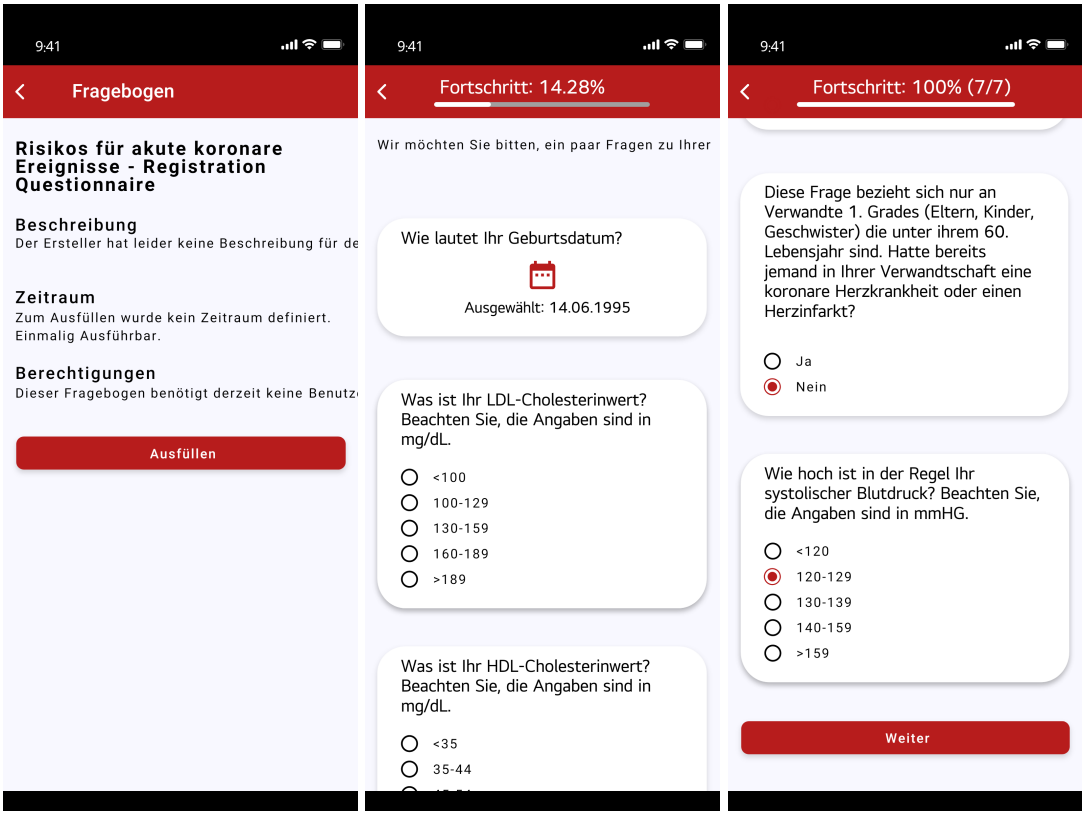
Nach der erfolgreichen Anmeldung gelangt der Benutzer auf die Hauptnavigationsfenster (Abbildung 5.4a). Diese besteht aus mehreren Tabs, dabei ist der erste Tab aktiv. Der aktive Tab wird durch das Ausfüllen der Farbe vom Icon und der jeweiligen Beschreibung hervorgebracht. Die Umsetzung ist beim Betrachten von Abbildung 5.4a und Abbildung 5.4b gut zu erkennen. Darüber hinaus kann durch das Anklicken vom *Hamburger-Icon* (links oben) der *Drawer* (Abbildung 5.4c) hervorgebracht werden. Der Drawer ermöglicht eine zusätzliche Navigation zu anderen Ansichten.



(a) Detaillierter Tipp (b) Tipp bewertet

Abbildung 5.5: Prototyp der Tipp-Ansicht

Durch das Auswählen eines Tipps aus Abbildung 5.4a gelangt der Benutzer auf eine detaillierte Ansicht des jeweiligen Tipps (Abbildung 5.5a). Der entsprechende Tipp kann durch das Anklicken der Sterne bewertet werden. Nach dem Anklicken wird die Änderung übernommen und direkt dem Benutzer angezeigt, diese findet sich in Abbildung 5.5b wieder.



(a) Detaillierter Fragebogen (b) Fragebogen-Elemente (c) Fragebogen abschicken

Abbildung 5.6: Prototyp der Fragebogen-Ansicht

Abbildung 5.6 zeigt den allgemeinen Ablauf, um einen Fragebogen auszufüllen. Nach dem Auswählen eines Fragebogens aus Abbildung 5.4b wird auf die detaillierte Ansicht (Abbildung 5.6a) vom jeweiligen Fragebogen gewechselt. Falls der Benutzer diesen Ausfüllen möchte, muss er auf den Button `Ausfüllen` klicken und gelangt dadurch auf eine neue Ansicht, worin sich die Fragebogen-Elemente befinden. Abbildung 5.6b und Abbildung 5.6c zeigt einen Ausschnitt der Fragebogen-Elemente für koronare Herzkrankheiten. Dabei wird in der Navigationsleiste der ausgefüllte Fortschritt angezeigt. Hier sind die Antwortmöglichkeiten durch Radio Buttons repräsentiert. Um den Fragebogen abschließen zu können, müssen alle Fragen ausgefüllt werden. Abschließend kann der Benutzer durch den am Ende der Seite befindlichen Button den Fragebogen mit den eingegebenen Antworten abschicken (Abbildung 5.6c).

6 Architektur

In diesem Kapitel wird die Architektur der TYH App beschrieben. Diese umfasst den Datenaustausch mit der API, die grafische Benutzeroberfläche und das eingesetzte Design Pattern in der Implementierung.

6.1 Datenaustausch

Die App kommuniziert mit der *Track Your Health* (TYH) API, welche eine REST-Architektur aufweist. An diese werden die darzustellenden Daten angefragt. Ebenso werden die Daten der ausgefüllten Fragebögen an diese übermittelt. Da es sich um eine Web-API handelt, erfolgen die Request-Nachrichten über das HTTP-Protokoll mit den entsprechenden Methoden, die in Unterabschnitt 2.3.1 erwähnt wurden. Der Body im Nachrichtenrumpf wird dabei im JSON-Format versendet. Im Folgenden werden die grundlegende Struktur der API und die erstellten Fragebögen vorgestellt. Die TYH API Dokumentation ist unter der Webseite[1] zu finden. In dieser sind die HTTP-Request Methoden mit entsprechenden Einzelheiten gelistet.

Authentifizierung: Die Authentifizierung eines Benutzers erfolgt über ein *JSON Web Token (JWT)*. Durch das JWT-Token kann die API identifizieren, um welchen Benutzer es sich handelt und welche Rechte dieser besitzt. Der Benutzer erhält den JWT-Token durch die Anmeldung über der Post-Request mit den nötigen Anmeldedaten. Anschließend verschickt die API bei einer gültigen Eingabe dem Benutzer einen JWT-Token als Response-Nachricht. Der erhaltene JWT-Token ist zeitlich begrenzt gültig und dient dem Benutzer zur Authentifizierung. Deshalb wird das JWT-Token bei vielen Request-Nachrichten benötigt und daher mitgegeben. Dadurch

[1] `https://api.dummy.trackyourhealth.net/dingodocs/v1.html` (21.08.2021)

wird die Beschränkung *Zustandslosigkeit* der REST-Architektur erfüllt, da die Sitzung nicht auf dem Server gespeichert wird, sondern die API den Benutzer anhand des JWT-Tokens authentifiziert.

Ein JWT-Token ermöglicht den sicheren Austausch von Daten zwischen zwei oder mehreren Systemen in Form von *Claims* (Ansprüchen). Dabei besteht ein JWT-Token aus einem String, der sich aus drei Komponenten zusammensetzt. Jeder dieser Komponenten ist Base64 codiert und wird durch einen Punkt getrennt. Den ersteren Teil bildet der *Header*, welcher Informationen über die Metadaten in JSON-Format enthält. Die Metadaten enthalten Informationen wie den eingesetzten Verschlüsselungsalgorithmus oder den Token-Typ [50].

Die *Payload* bildet die zweite Komponente. In dieser sind die eigentlichen Informationen bzw. Claims enthalten. Die Claims sind dabei als Key-Value-Paare vorzufinden, da die Payload ebenfalls im JSON-Format ist. Die Anzahl und Art der Claims ist nicht konkret angegeben, weshalb beliebig viele definiert werden können. Wie oben erwähnt ist ein JWT-Token begrenzt gültig mit einem entsprechenden Ablaufdatum, welches hier als Claim definiert wird. Das Ablaufdatum ist unter dem Key *exp* definiert und enthält das Datum in Form eines *Timestamps*. Mittels des Timestamp kann die API überprüfen, ob der entsprechende JWT-Token seine Gültigkeit besitzt. Deshalb müssen JWT-Tokens vor oder nach ihrem Ablauf aktualisiert werden. Ebenfalls können die Claims Informationen über die Rolle (Administrator oder Benutzer) vom Anfragenden enthalten [50].

Die letzte Komponente vom JWT-Token bildet die *Signatur*. Die angehängte Signatur ist eine Folge von Zeichen und kann durch unterschiedliche Algorithmen erstellt werden. Einer dieser Algorithmen ist der *HS256*. Mit der Signatur wird die Gültigkeit und Korrektheit des JWT-Tokens bei der Übertragung an den Empfänger verifiziert [50].

In Abbildung 6.1 ist ein JWT-Token aus der TYH API mit den entsprechenden Komponenten zu sehen.

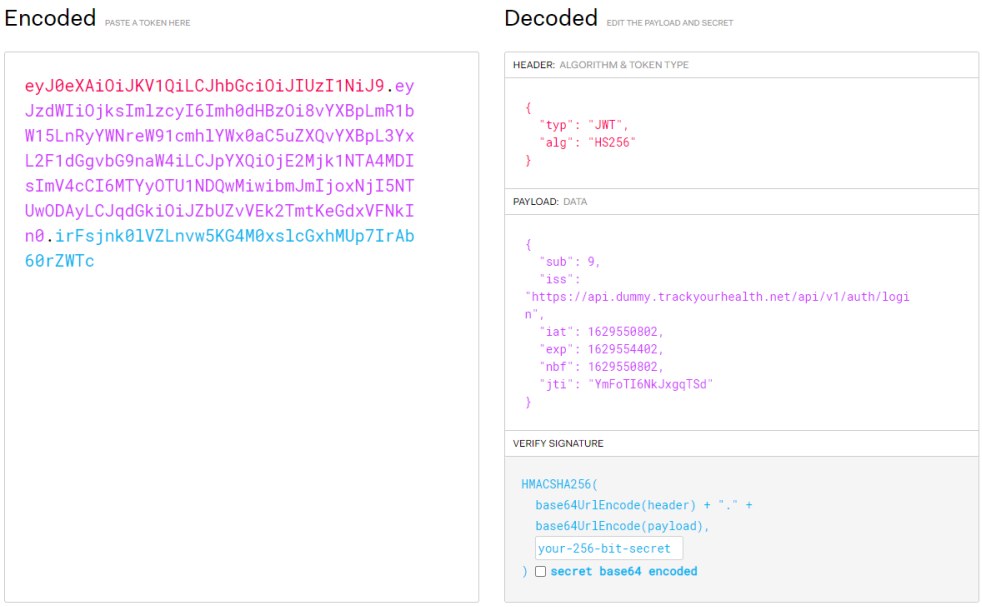


Abbildung 6.1: JWT Token mit der zugehörigen Dekodierung[2]

Struktur TYH API: Die TYH API hat Request-Nachrichten, um jeweils eine Liste aller vorhandenen Studien, Fragebögen und Tipps zu erhalten. Jeder dieser Listenelemente besitzt eine zugeordnete ID. Mit der ID kann eine Anfrage erstellt werden, um genauere Informationen über das jeweilige Element zu erhalten. Das Prinzip wird mit folgendem Beispiel veranschaulicht. Dabei sind die verwendeten Methoden aus der Dokumentation entnommen. Es wird eine GET Request-Nachricht an */api/v1/studies* gestellt, um alle Studien zu erhalten. Die Response-Nachricht beinhaltet eine Liste von Studien, in welcher jede Studie ein Attributfeld *ID* besitzt. Diese ID wird nun für die GET Request-Nachricht */api/v1/studies/<id>* benötigt, um genauere Details über die jeweilige Studie zu erhalten. Anstelle von *<id>* sollte die jeweilige ID der Studie eingesetzt werden.

Die Studien sind dabei so gegliedert, dass diese wiederum Fragebögen enthalten. Durch eine weitere Request-Nachricht welche die Studien ID enthält, können die jeweiligen Fragebögen erhalten werden. Eine andere Möglichkeit ist, dass diese in der Liste aller Fragebögen erscheinen, sobald der Benutzer sich in die Studie einschreibt. Die Fragebögen besitzen ebenfalls eine ID, die benötigt wird, um die

[2]Dekodiert mit `https://jwt.io/` (21.08.2021)

Fragebogenelemente anzufragen. Die Fragen haben dabei unterschiedliche Fragetypen. Diese Fragetypen definieren die Antwortformate, wodurch der Entwickler Einschränkungen und Darstellungsarten in der UI festlegen kann. Im Folgenden werden die im Fragebogen für koronare Herz-Risiko-Patienten eingesetzten Fragetypen erläutert:

- **SingleChoice** Fragen haben vordefinierte Antwortmöglichkeiten, von denen eine Antwort ausgewählt werden soll.
- **YesNoSwitch** Fragen können nur mit *Ja* oder *Nein* beantwortet werden.
- **TextString** Fragen akzeptieren alle Antworten, die in Textform (String) sind.

Erstellung des Fragebogens: Der Fragebogen für koronare Herz-Risiko-Patienten wurde aus den Risikofaktoren aus Tabelle 2.1 erstellt. Dazu wurden die Risikofaktoren in folgende sieben Fragen mit den Fragetypen und möglichen Antworten ausgearbeitet:

Frage	Fragetyp	Antworten
Wie lautet Ihr Gerburtsdatum?	TextString	Format DD.MM.YYYY
Was ist ihr LDL-Cholosterintwert? Beachten Sie, die Angaben sind in mg/dL.	SingleChoice	<100; 100-129; 130-159; 160-189; >189
Was ist Ihr HDL-Cholosterintwert? Beachten Sie, die Angaben sind in mg/dL.	SingleChoice	<35; 35-44; 45-54, >54
Was ist Ihr Triglyceridwert? Beachten Sie, die Angaben sind in mg/dL.	SingleChoice	<100; 100-149; 50-199; > 199
Rauchen Sie?	YesNoSwitch	Ja; Nein
Leiden Sie unter Diabetes mellitus (erhöhter Blutzucker)?	YesNoSwitch	Ja; Nein
Diese Frage bezieht sich nur an Verwandte 1. Grades (Eltern, Kinder, Geschwister) die unter ihrem 60. Lebensjahr sind. Hatte bereits jemand in Ihrer Verwandtschaft eine koronare Herzkrankheit oder einen Herzinfarkt?	YesNoSwitch	Ja; Nein
Wie hoch ist in der Regel Ihr systolischer Blutdruck? Beachten Sie, die Angaben sind in mmHG.	SingleChoice	<120; 120-129; 130-139; 140-159; >159

Tabelle 6.1: Fragen für koronare Herz-Risiko-Patienten mit Antwortmöglichkeiten

Diese Fragen wurden in zwei verschiedenen Fragebögen eingebaut und anschließend in das entsprechende JSON-Format geparst, damit diese in der API zur Verfügung gestellt werden können. Die *Baseline*-Variante enthält die Frage über das Geburtsdatum im Gegensatz zur *Follow-up*-Variante. Das hat den Hintergrund, dass der Baseline-Fragebogen einmalig ausführbar ist, da das Geburtsdatum kein änderbarer Parameter ist. Der Follow-up-Fragebogen hingegen ist wiederkehrend und kann öfters ausgefüllt werden.

6.2 Graphische Benutzeroberfläche

Abbildung 6.2 zeigt die Navigationsstruktur der App. Der *Main screen* repräsentiert dabei die Hauptansicht, welche die Tableiste beinhaltet. Über die Tab-Leiste kann der Benutzer zwischen den verschiedenen Tabs navigieren. Das Ausloggen aus der App kann nur in der Hauptansicht erfolgen. Hierbei besteht die Möglichkeit, die Aktion durch die gerätespezifische Zurück-Steuerung oder durch das Anklicken vom *Ausloggen*-Item im *Drawer* auszuführen.

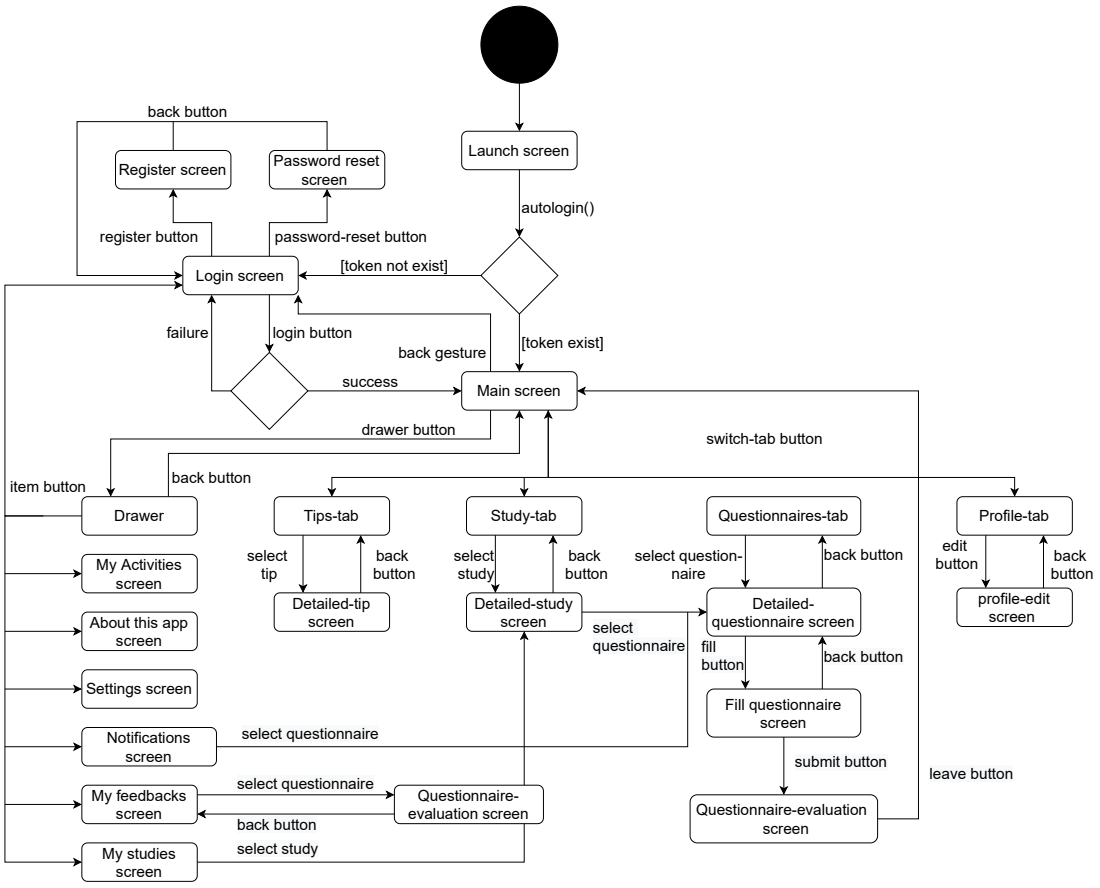


Abbildung 6.2: Zustandsdiagramm Navigation

6.3 Design-Pattern

Wie in Unterabschnitt 2.4.3 erwähnt besitzen die Widgets in Flutter States. Die Widgets sind reaktiv, da sie auf neuen Input reagieren und den State vom Widget ändern. Somit ist Flutter ein reaktives und deklaratives UI-Framework [11]. Das reaktive kommt aus der funktionalen Programmiersprache und beschreibt das Programmierparadigma *Reactive Programming* für interaktive Anwendungen. Das Reactive Programming Paradigma erleichtert die Entwicklung von Anwendungen, die auf Ereignisse basieren. Dabei werden die auszuführenden Aktionen für die entsprechenden Ereignisse definiert und die Programmiersprache verwaltet die automatische Ausführung der Aktion [4]. In diesem Zusammenhang übernimmt Flutter das Aktualisieren der Benutzeroberfläche (Widgets) bei Änderungen des States während der Laufzeit. Aus diesem Grund spielt die Wahl der Architektur bzw. das Pattern im Statemanagement eine wichtige Rolle.

Flutter hat zum Verwalten der States verschiedene Ansätze, welche auf der Webseite[3] von Flutter gelistet sind. Einer dieser Ansätze ist das *B*usiness *Lo*gic *C*omponents *(BLoC)* Pattern, das von Google vorgestellt wurde [18]. Das BLoC Pattern ermöglicht durch das Verwalten der States, die Trennung von UI und Logik. Das Pattern basiert auf der *asynchronen Programmierung* mit *Streams* [9].

Streams ermöglichen das Empfangen einer Folge von Ereignissen *(Events)*. Dabei handelt sich um Datenereignisse oder Fehlereignisse. Diese sind Benachrichtigungen vom entsprechenden Ereignis. Das Empfangen der Ereignisse wird durch das Abhören eines *Listeners* ermöglicht. Zeitgleich bringt der Listener den Stream dazu, Ereignisse zu erzeugen. Das empfangene Ereignis ist ein *StreamSubscription*-Objekt, welches das Ereignis enthält, aber auch das Beenden oder Pausieren vom Listener ermöglicht [23]. Somit lassen Streams Daten weiterleiten, die durch den Listener empfangen werden. Sobald die Daten empfangen sind, kann die UI automatisch mit dynamischen Funktionen die Daten darstellen.

Beim BLoC Pattern werden sogenannte *BLoCs* durch Klassen erstellt. Die Klassen enthalten die Programmierlogik und zugleich die States in Form von Daten, die verwaltet werden. Die Daten werden durch Events von der UI ausgelöst und müssen in States umgewandelt werden. Im Umkehrschluss müssen beim Erstellen eines

[3] `https://flutter.dev/docs/development/data-and-backend/state-mgmt/options` (23.08.2021)

BLoCs die States, Events und die *mapEventToState* implementiert werden [18]. In Abbildung 6.3 ist der Ablauf dargestellt. Die UI verschickt Events an den BLoC, dieser führt die Funktionen aus, um die Daten zu erhalten. Gegebenenfalls werden die Daten weiter verarbeitet bzw. in States umgewandelt. Anschließend wird die UI über Streams benachrichtigt, um ihren Zustand zu ändern.

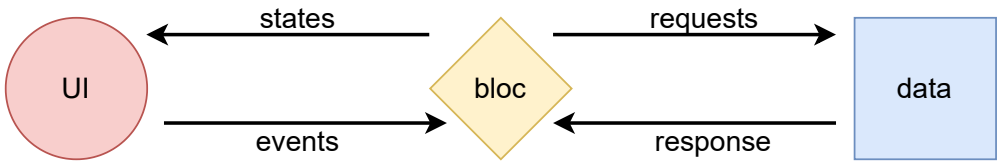


Abbildung 6.3: BLoC Pattern Funktionsweise [9]

In dieser Arbeit wurden keine BLoCs erstellt, sondern *Cubits*. Cubit ist eine Teilmenge vom BLoC Pattern mit dem Unterschied, dass in diesem der State definiert werden muss und nicht wie beim BLoC auf Events angewiesen ist. Anstelle der Events werden Methoden verwendet und die UI wird durch das Emittieren *(emit)* benachrichtigt [9]. Aus diesem Grund wurde für jedes Fenster und Tab (siehe Abbildung 6.2) ein Cubit erstellt. Jedes dieser Cubits hat somit die Logik der jeweiligen Fenster implementiert und stellt unter anderem die Requests an die TYH API, um die entsprechenden Daten zu erhalten. Das Cubit Konzept ist in Abbildung 6.4 dargestellt.

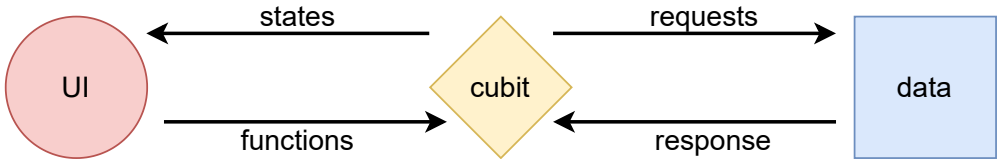


Abbildung 6.4: Cubit Funktionsweise [9]

Die Erstellung vom Cubit ist einfacher und erfordert weniger Code im Vergleich zum BLoC, da keine mapEventToState-Methode implementiert werden muss. Ein Beispiel für einen Cubit ist in Listing 6.1 aufgeführt, welches mit dem *flutter_bloc* Package erstellt worden ist. Das Package wird in Abschnitt 7.2 genauer vorgestellt. Das aufgeführte Beispiel enthält als State einen `int` Wert (Zeile 1), welcher mit `0` initialisiert wird (Zeile 2). Der Cubit hat eine `increment` Methode implementiert, welche den aktuellen Wert inkrementiert und durch das `emit` die UI benachrichtigt

(Zeile 4). Die Methode könnte beispielsweise in der UI mit einem Button-Widget aufgerufen werden, wohingegen ein Text-Widget den State repräsentiert. Das Text-Widget würde nun durch das `emit` benachrichtigt werden und anschließend seinen State ändern.

```
1 class CounterCubit extends Cubit<int> {
2   CounterCubit() : super(0);
3
4   void increment() => emit(state + 1);
5 }
```

Listing 6.1: Programm *Hello World* in Dart

7 Implementierung

In diesem Kapitel werden einige ausgewählte Technologien, Werkzeuge sowie Bibliotheken und Erweiterungen vorgestellt, die während der Implementierung eingesetzt wurden. Im Anschluss wird die entstandene App präsentiert.

7.1 Technologie und Werkzeuge

Die App wurde in Flutter *Version 2.2.3* mit der integrierten Entwicklungsumgebung (IDE) *Visual Studio (VS) Code*[1] implementiert. In der IDE ist das Installieren vom Flutter Plugin möglich. Diese unterstützt den Entwickler durch die Code-Vervollständigung, Syntax-Hervorhebung, Widget-Bearbeitung und erlaubt das Ausführen und Debuggen der Anwendung [25]. Zur Versionsverwaltung wurde die Software *GitHub*[2] genutzt. Die Kommunikation erfolgt wie in Abschnitt 6.1 vorgestellten TYH API.

7.2 Bibliotheken und Erweiterungen

Im Folgenden werden ausgewählte Bibliotheken und Erweiterungen vorgestellt, die den Arbeitsaufwand in der Implementierung verringert und zeitgleich Lösungsansätze angeboten haben.

http (Version 0.13.3)[3] ist ein Package, welches einfache und individuelle HTTP-Request-Nachrichten mit minimalem Aufwand ermöglicht. Sämtliche Request-Nachrichten

[1] `https://code.visualstudio.com/` (23.08.2021)
[2] https://github.com/ (23.08.2021)
[3] https://pub.dev/packages/http (23.08.2021)

wurden mit diesem Package erstellt. In Listing 7.1 ist eine GET-Methode (Zeile 3) aus dem verwendeten Package zu sehen. Mit dieser Request-Nachricht wird die Liste aller Studien aus der TYH API angefragt (Zeile 2). Zusätzlich sind die `Content-Type` und `Accept-Language` Header gesetzt (Zeile 7–8).

```
1  Future<dynamic> fetchStudies() async {
2    String path = 'https://api.dummy.trackyourhealth.net/api/v1/studies';
3    var response = await http.get(
4      Uri.parse(path),
5      headers: {
6        'Content-Type': 'application/json',
7        'Accept-Language': 'de',
8      },
9    );
10   return response;
11 }
```

Listing 7.1: HTTP-GET-Request mit dem http-Package

flutter_bloc (Version 7.0.0) [4] wurde verwendet, um das vorgestellte BLoC Pattern in Abschnitt 6.3 in der Anwendung umzusetzen. Das Package ermöglicht das einfache Implementieren von BLoCs und Cubits in Flutter in Form von Widgets. Der Entwickler muss dadurch keine eigenen Streams und Listener erstellen, wodurch Arbeitsaufwand erspart wird.

flutter_datetime_picker (Version 1.5.1) [5] ist ein Widget, welches von der Flutter-Community entworfen wurde. Das Widget erlaubt dem Benutzer, ein Datum durch einen Picker (Auswahl-Rad) auszuwählen. Die Funktionsweise und das entsprechende Design sind dem *iOS-Date-Picker*[6] angelehnt. Zeitgleich erlaubt das Package persönliche Anpassungen von Design und Sprache.

[4] `https://pub.dev/packages/flutter_bloc` (23.08.2021)
[5] `https://pub.dev/packages/flutter_datetime_picker`(23.08.2021)
[6] iOS Pickers: `https://developer.apple.com/design/human-interface-guidelines/ios/controls/pickers`(23.08.2021)

flutter_local_notifications (Version 6.0.0)[7] ermöglicht auf verschiedenen Betriebssystemen lokale Benachrichtigungen zu konfigurieren. Die Benachrichtigungen können dabei in Intervalle (minütlich, stündlich, täglich, wöchentlich) oder für bestimmte Tage eingerichtet werden. Zusätzlich können bei den täglichen, wöchentlichen und bestimmten Tagen die Uhrzeit der Benachrichtigung angegeben werden. Mit diesem Package wurden die Erinnerungen für tägliche und wöchentliche wiederkehrende Fragebögen in Form von Benachrichtigungen implementiert.

shared_preferences (Version 2.0.6)[8] erlaubt das Speichern von einfachen Daten im Key-Value Format auf dem plattformspezifischen Speicher des genutzten Gerätes. Dabei werden die Daten auf iOS im *NSUserDefaults* und auf Android im *SharedPreferences* gespeichert. Mit diesem wird das erhaltene JWT-Token aus der TYH API sowie das geänderte Theme auf dem jeweiligen Gerät gespeichert.

device_info_plus (Version 2.1.0)[9] ermöglicht Information über das Gerät zu bekommen. Beim Absenden vom Antworbogen (POST-Request-Nachricht) besteht die Möglichkeit Geräteinformationen mitzugeben. Die TYH API akzeptiert dabei den Namen, das Modell und das Betriebssystem vom Gerät. Durch das Package werden diese Informationen erhalten, um eine Mitgabe dieser Daten in der POST-Request-Nachricht zu ermöglichen.

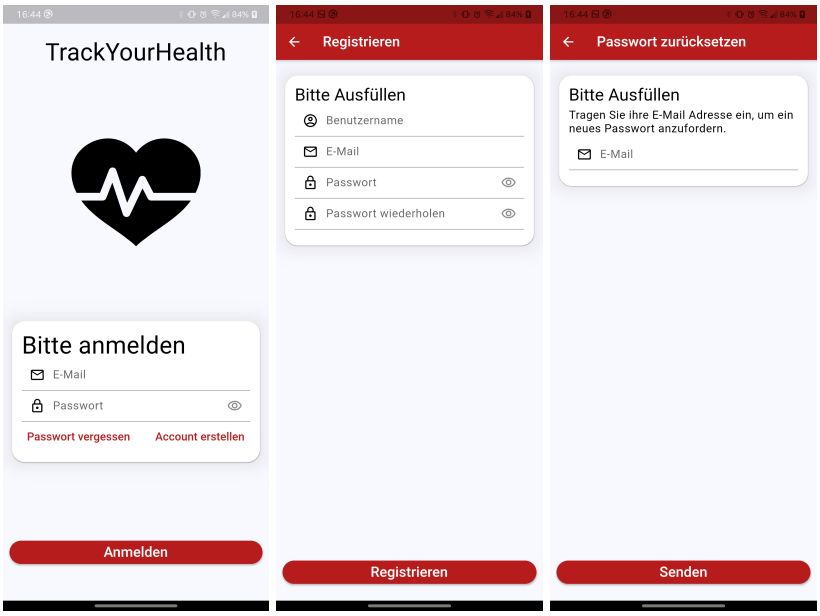
7.3 Vorstellung der App

In diesem Abschnitt wird die implementierte mobile Applikation vorgestellt. Die Applikation ist hauptsächlich im Classic Theme zu sehen. Um einen Eindruck der verschiedenen Themes zu erhalten, wurden zusätzlich einzelne Ansichten in unterschiedlichen Themes beigefügt.

[7] `https://pub.dev/packages/flutter_local_notifications` (23.08.2021)
[8] `https://pub.dev/packages/shared_preferences` (23.08.2021)
[9] `https://pub.dev/packages/device_info_plus` (23.08.2021)

7.3.1 Anmelden

Abbildung 7.1a zeigt die Ansichten, mit dem der Benutzer beim Starten der App interagieren kann. Falls sich zuvor angemeldet wurde und ein JWT-Token existiert, wird diese Ansicht übersprungen. Der Benutzer startet mit der Ansicht in Abbildung 7.1a. In diesem kann sich direkt angemeldet werden oder durch die jeweiligen Text-Buttons (farblich) auf die beiden anderen Ansichten navigiert werden. In der *Registrierung* Ansicht (Abbildung 7.1b) kann der Benutzer sich ein Konto anlegen. In der Ansicht zum Zurücksetzen des Passworts (Abbildung 7.1c) kann durch Angabe der E-Mail-Adresse das entsprechende Passwort zurückgesetzt werden.

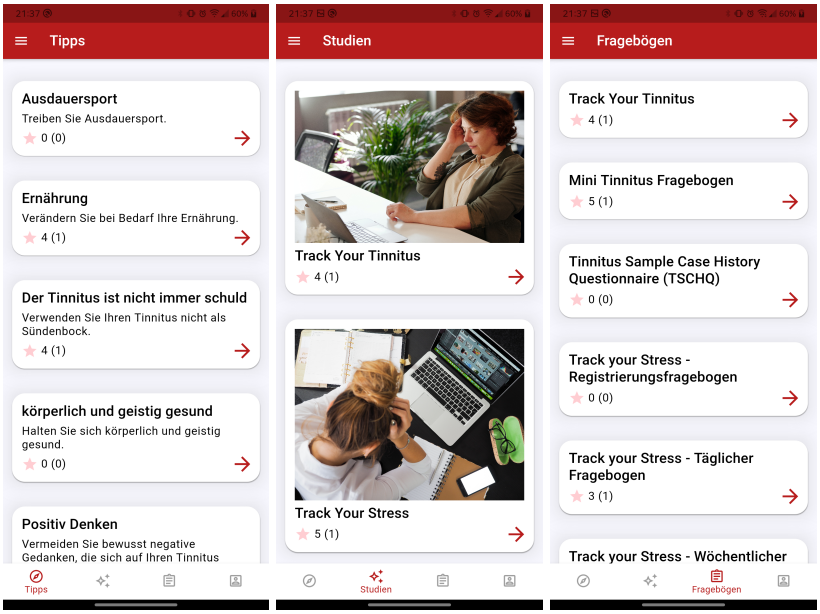


(a) Login (b) Registrierung (c) Passwort zurücksetzen

Abbildung 7.1: Anmelde-Ansichten im Classic Theme

7.3.2 Tabs

Nach einer erfolgreichen Anmeldung gelangt der Benutzer in die Ansicht in Abbildung 7.2a. Die Icons in der unteren Leiste ermöglichen das Navigieren zwischen den Tabs (Abbildung 7.2 und 7.3). Die Tabs für *Tipps*, *Studien* und *Fragebögen* sind von der Struktur identisch aufgebaut (Abbildung 7.2). Diese besitzen jeweils eine Liste von Kärtchen, welche die Listenelemente (Tipp, Studie oder Fragebogen) repräsentieren. Das Kärtchen enthält dabei Informationen wie den Titel, die Bewertung und falls vorhanden der Beschreibung. Durch das Anklicken der Kärtchen gelangt der Benutzer auf die jeweilige detaillierte Ansicht, welche in den Unterabschnitten 7.3.4-7.3.6 vorgestellt wird. Das *Profil-Tab* (Abbildung 7.3) hingegen enthält persönliche Informationen vom Benutzer. Durch das Klicken auf den `Profil bearbeiten`-Button, welcher sich unter dem *Profil-Icon* befindet, gelangt der Benutzer auf die Bearbeitungsansicht vom Profil (Unterabschnitt 7.3.7).



(a) Tipps (b) Studien (c) Fragebögen

Abbildung 7.2: Tab-Ansichten im Classic Theme

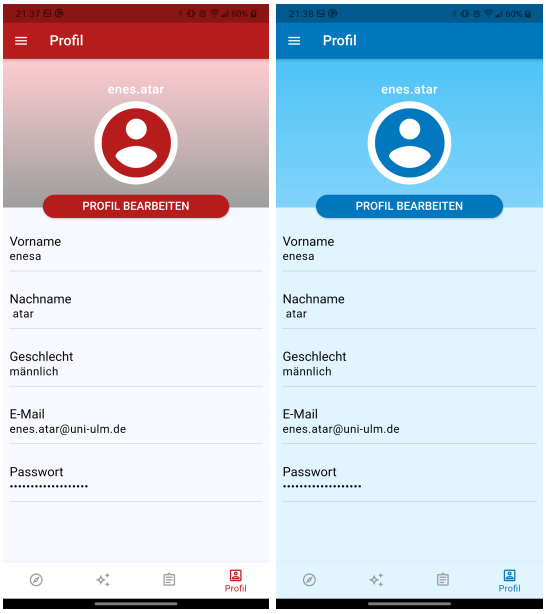


Abbildung 7.3: Profil-Tab im Classic und Ocean Blue Theme

7.3.3 Drawer

Eine zusätzliche Navigation ergänzend der Tabs ermöglicht der *Drawer*, welcher in Abbildung 7.4 zu sehen ist. Mit dieser ist es möglich, auf weitere Ansichten zu gelangen. Der Drawer wird durch das Anklicken vom *Hamburger-Icon* in den Tab-Ansichten (Abbildung 7.2) (links oben) geöffnet.

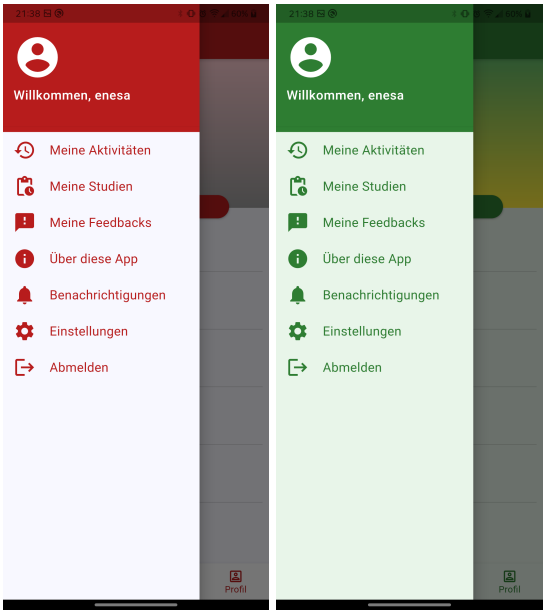
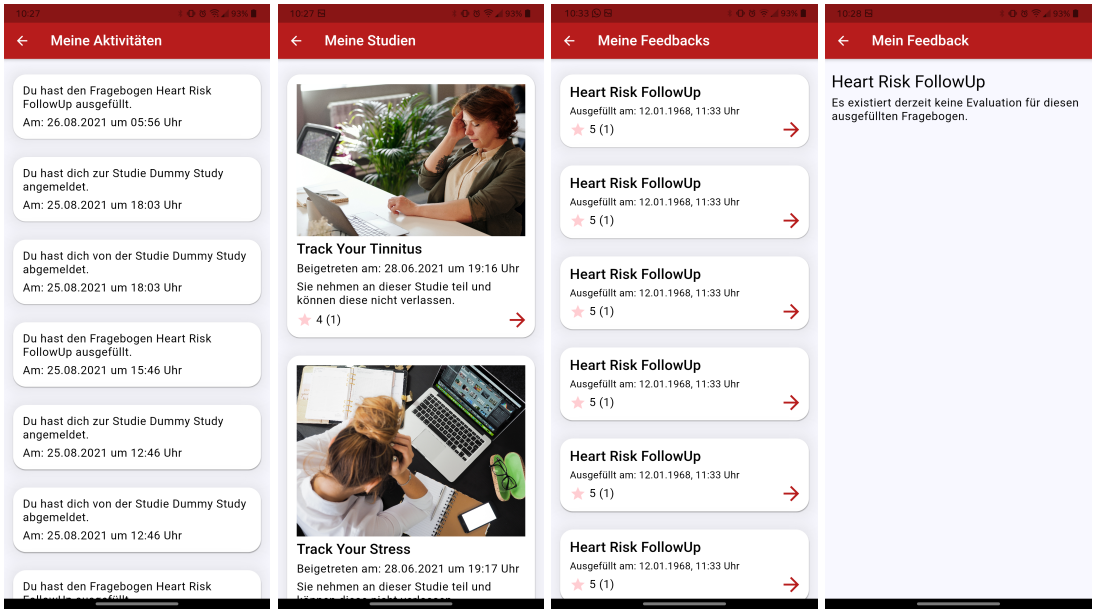


Abbildung 7.4: Drawer im Classic und Weed Green Theme

Im Folgenden werden die verschiedenen Ansichten vorgestellt, die durch den Drawer geöffnet werden können. Einer der Ansichten ist die *Meine Aktivitäten* Ansicht (Abbildung 7.5a). In dieser befinden sich alle Aktivitäten vom Profil, die auf der TYH API gespeichert sind. Eine weitere Ansicht ist die *Meine Studien* Ansicht (Abbildung 7.5b). Dort sind die eingeschriebenen Studien in Form von Kärtchen gelistet. Die Kärtchen enthalten wiederum Informationen über die jeweilige Studie und ermöglichen durch das Anklicken auf die Detailansicht davon zu gelangen (Unterabschnitt 7.3.5). Eine Übersicht über die ausgefüllten Fragebögen mit einer möglichen Evaluation sind in der *Meine Feedbacks* Ansicht (Abbildung 7.5c) zu finden. Von dort aus kann der Benutzer durch das Anklicken des jeweiligen Kärtchens auf die Evaluation vom angeklickten Fragebogen (Abbildung 7.5d) gelangen.

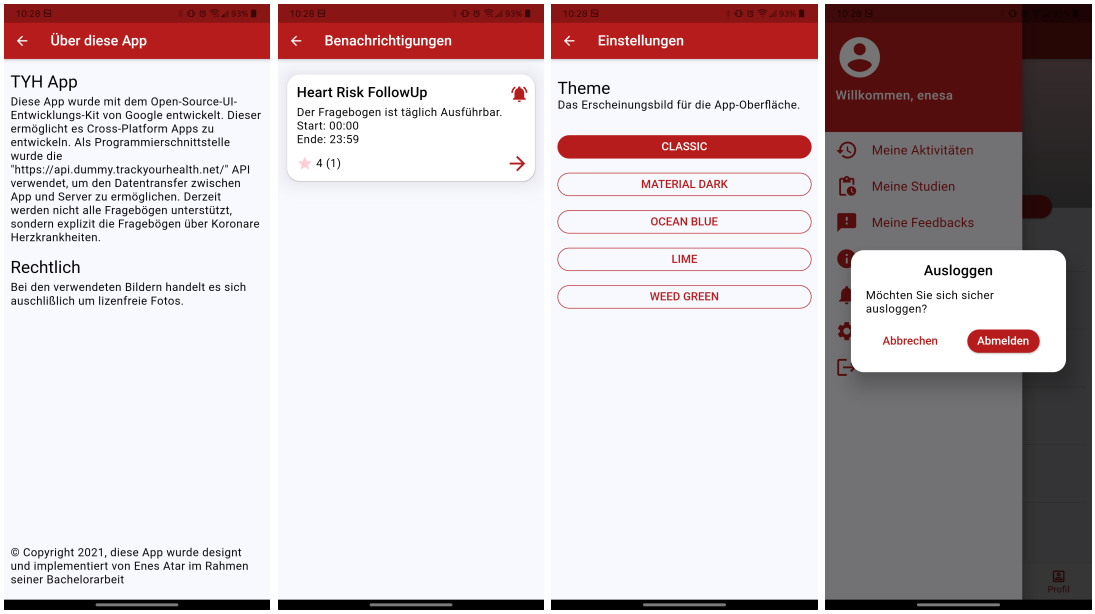


(a) Meine Aktivitäten (b) Meine Studien (c) Meine Feedbacks (d) Evaluation

Abbildung 7.5: Drawer-Fensterinhalte (1) im Classic Theme

Abbildung 7.6a zeigt die *Über diese App* Ansicht, welche Informationen über die implementierte App enthält. Die aktivierten Benachrichtigungen für wiederkehrende Fragebögen sind unter *Benachrichtigungen* zu finden (Abbildung 7.6b). Der Benutzer kann darin einzelne Benachrichtigungen durch das Alarm-Icon abschalten oder navigiert durch das Anklicken vom Kärtchen auf den jeweiligen Fragebogen. Die Themes können in der *Einstellung* Ansicht (Abbildung 7.6c) geändert werden.

Dabei wird durch das Anklicken der Buttons das Theme direkt übernommen und im Speicher gespeichert. Durch das Anklicken von *Abmelden* im Drawer hingegen wird keine Ansicht, sondern ein Dialog geöffnet (Abbildung 7.6d). Falls der Benutzer sich abmeldet, wird der gespeicherte JWT-Token aus dem Speicher gelöscht und der Benutzer gelangt wieder in die Anmelde Ansicht (Abbildung 7.1a).

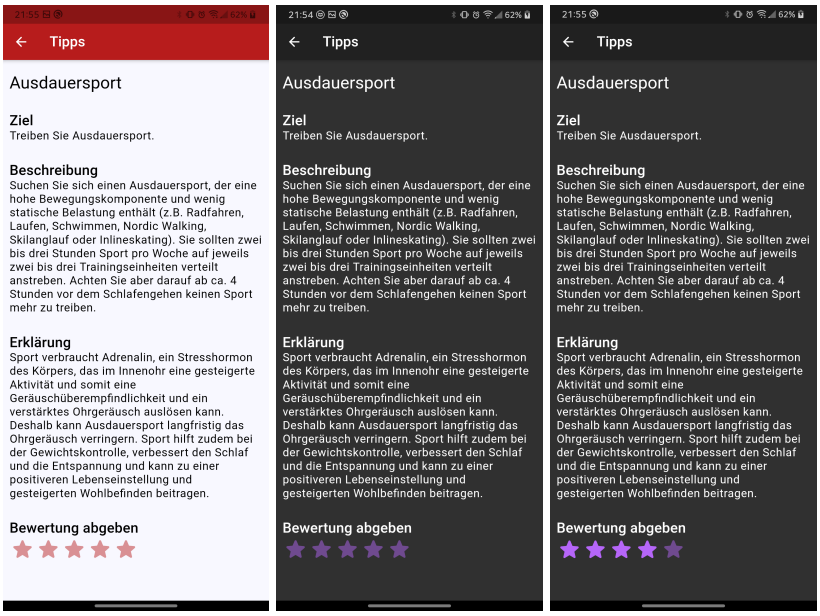


(a) Über diese App (b) Benachrichtigungen (c) Einstellungen (d) Ausloggen Dialog

Abbildung 7.6: Drawer-Fensterinhalte (2) im Classic Theme

7.3.4 Tipp

Durch das Anklicken eines Tipps im *Tipp-Tab* wird die entsprechende Detailansicht geöffnet. In Abbildung 7.7 ist ein Tipp zu sehen. Der Benutzer kann in der Detailansicht den Tipp durch die Vergabe von Sternen bewerten. Die Bewertung wird durch die Verfärbung vom Stern kenntlich gemacht. Beim Betrachten von Abbildung 7.7b und 7.7c ist die Verfärbung zu erkennen.

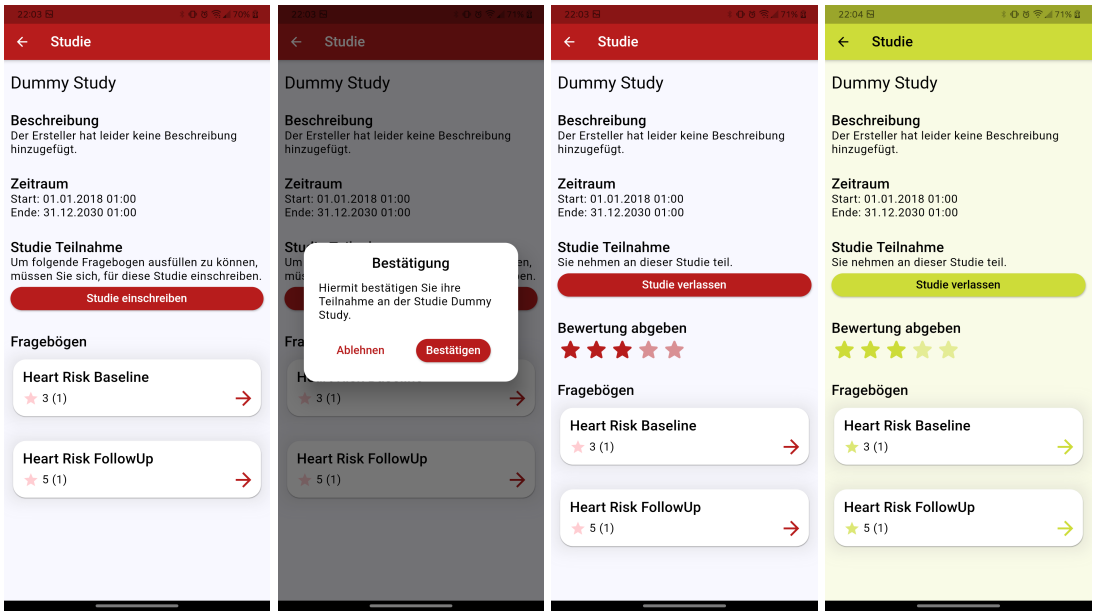


(a) Nicht bewertet (b) Nicht bewertet (c) Bewertet

Abbildung 7.7: Tipp im Classic und Material Dark Theme

7.3.5 Studie

Nach einer Auswahl einer Studie (Abbildung 7.2b) wird die entsprechende Detailansicht geöffnet. Die Abbildung 7.8a zeigt diese Ansicht. Dem Benutzer werden darin nähere Informationen über die Studie angezeigt. Diese beinhaltet, falls vorhanden, eine Beschreibung, den ausfüllbaren Zeitraum und die damit verbundenen Fragebögen. Ebenfalls kann der Benutzer durch einen Button sich in die Studie einschreiben (Abbildung 7.8a) oder diese verlassen (Abbildung 7.8c). In beiden Fällen wird ein Dialog angezeigt (Abbildung 7.8b), die bestätigt werden muss. Zusätzlich können eingeschriebene Studien durch Vergabe von Sternen bewertet werden.

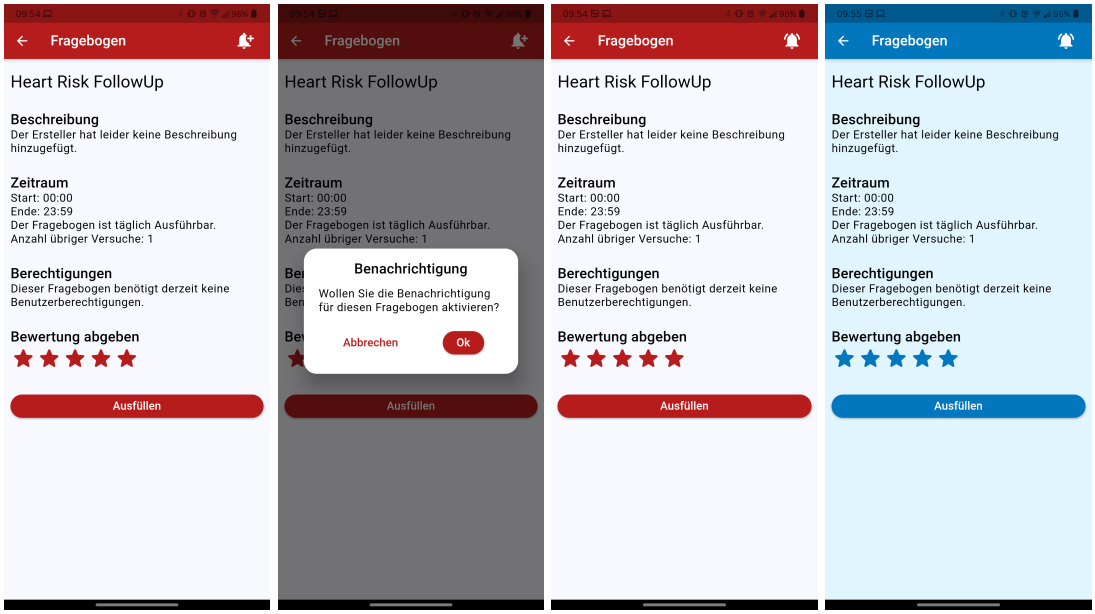


(a) Nicht Eingeschrieben (b) Dialog einschreiben (c) Eingeschrieben (d) Eingeschrieben

Abbildung 7.8: Studie-Details im Classic und Lime Theme

7.3.6 Fragebogen

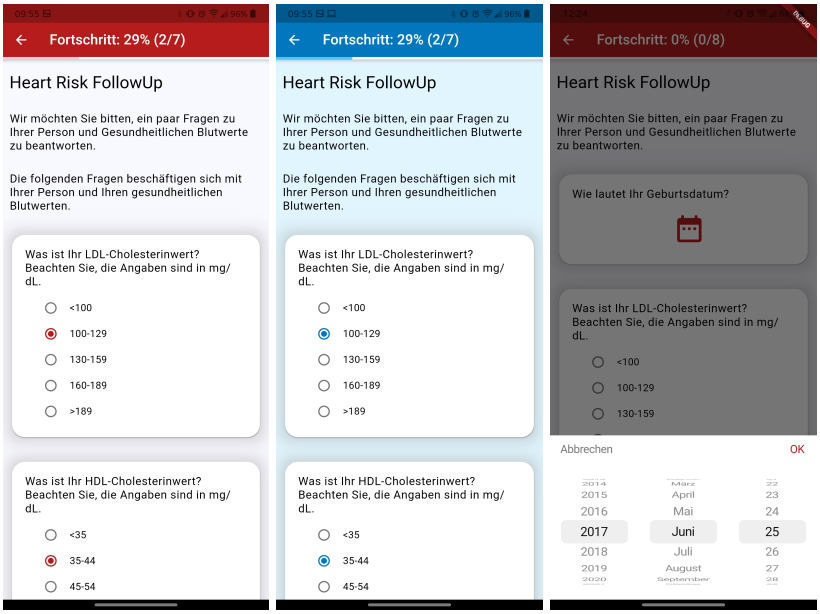
In Abbildung 7.9a ist die Detailansicht vom Fragebogen zu sehen. Der Benutzer bekommt hier, falls vorhanden, die Beschreibung des Fragebogens, den ausfüllbaren Zeitraum und die benötigte Geräteberechtigung der Sensoren. Für wiederkehrende Fragebögen kann in der Navigationsleiste durch das Alarm-Icon die Benachrichtigung für diese aktiviert werden. Diese muss ebenfalls durch einen Dialog (Abbildung 7.9b) bestätigt werden. Abbildung 7.9c zeigt den Icon bei einer aktivierten Benachrichtigung. Zugleich ermöglicht das Icon das Deaktivieren der Benachrichtigung. Der `Ausfüllen`-Button wird nur angezeigt, falls der Benutzer in der jeweiligen Studie eingeschrieben ist.



(a) Benachrichtigung deaktiviert (b) Dialog aktivieren (c) Benachrichtigung aktiviert (d) Benachrichtigung aktiviert

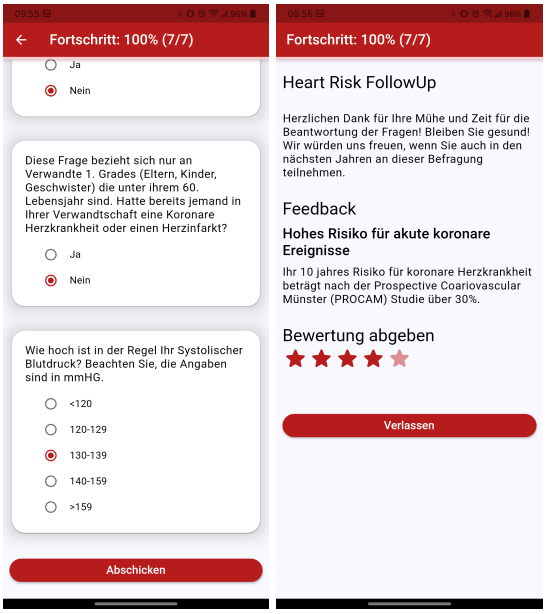
Abbildung 7.9: Fragebogen-Details im Classic und Ocean Blue Theme

Abbildung 7.10 und 7.11 zeigt den Fragebogen-Ablauf für koronare Herz-Risiko-Patienten. In Abbildung 7.10a sind die *SingleChoice*-Fragetypen abgebildet. Der Fortschritt wird dabei durch den Fortschrittsbalken unterhalb sowie in der Navigationsleiste angezeigt. Der in Abschnitt 7.2 vorgestellte Datum-Picker ist in Abbildung 7.10c zu sehen. Sobald der Benutzer die Fragen beantwortet hat, können die Antworten durch den *Abschicken*-Button (Abbildung 7.11a) verschickt werden. Im Anschluss wird dem Benutzer eine abschließende Ansicht angezeigt, die in Abbildung 7.11b zu sehen ist. Diese enthält die Evaluation der Antworten, die Bewertungsmöglichkeit vom Fragebogen und anschließend den Button, um die Ansicht zu verlassen.



(a) Fragen (b) Fragen (c) Datum-Picker

Abbildung 7.10: Fragebogen (1) im Classic und Ocean Blue Theme

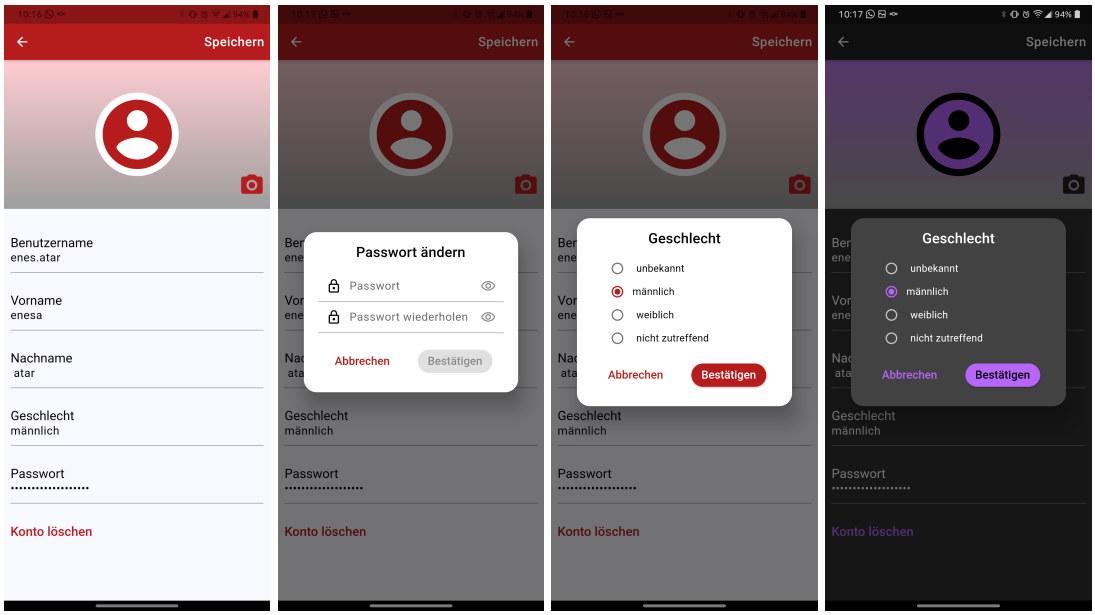


(a) Abschicken (b) Evaluation

Abbildung 7.11: Fragebogen (2) im Classic Theme

7.3.7 Profil bearbeiten

Beim Bearbeiten vom Profil (Abbildung 7.12a) kann der Benutzer die Felder `Benutzername`, `Vorname` und `Nachname` direkt durch das Anklicken der jeweiligen Textfelder bearbeiten. Beim `Passwort` und `Geschlecht` hingegen wird ein Dialog angezeigt (Abbildung 7.12b, 7.12c), worin der Benutzer die Eingabe bestätigen muss. Anschließend kann der Benutzer in der Navigationsleiste durch den `Speichern`-Text-Button die Änderungen übernehmen. Falls eine Änderung stattgefunden hat und der Benutzer die Ansicht versucht, zu verlassen ohne zu speichern, wird eine Meldung angezeigt, ob die Änderungen übernommen oder verworfen werden sollen.



(a) Konfigurierbare Inhalte (b) Dialog Passwort ändern (c) Dialog Geschlecht ändern (d) Dialog Geschlecht ändern

Abbildung 7.12: Profil bearbeiten im Classic und Material Dark Theme

8 Anforderungsabgleich

Im Folgenden werden die spezifizierten Anforderungen aus Kapitel 4 mit der entwickelten mobilen Anwendung abgeglichen. Das Kapitel untergliedert sich in funktionale und nicht-funktionale Anforderungen wie in der Anforderungsanalyse. Dabei wurde die Reihenfolge der Anforderungen übernommen.

8.1 Funktionale Anforderungen

FR01 **Registrierung** Erfüllt

Der Benutzer kann sich in der App mit einem Benutzernamen, gültiger E-Mail-Adresse und Passwort ein Konto erstellen. Bei fehlerhaften Eingaben wird eine entsprechende Meldung angezeigt.

FR02 **Passwort vergessen** Erfüllt

Durch Eingabe der E-Mail-Adresse vom entsprechenden Konto kann der Benutzer das Passwort zurücksetzen.

FR03 **Anmeldung** Erfüllt

Die Anmeldung mit einem gültigen Konto ist möglich. Bei einem auftretenden Fehler wird dem Benutzer eine entsprechende Meldung angezeigt. Zusätzlich wird der empfangene JWT-Token auf dem Gerät gespeichert.

FR04 **Abmeldung** Erfüllt

Der Benutzer kann sich aus dem Benutzerkonto erfolgreich abmelden. Dabei wird das gespeicherte JWT-Token auf dem Gerät gelöscht.

FR05 **Aktualisierung vom Token** Erfüllt

Der aktuelle JWT-Token wird alle 8 Minuten automatisch erneuert. Ebenfalls wird dieser für die weiteren Anfragen eingesetzt.

FR06 **Vergabe von Bewertungen (Likes)** Erfüllt

In der App ist es möglich, einzelne Tipps, Studien und Fragebögen durch Vergabe von 1 Stern bis zu 5 Sternen zu bewerten.

FR07 **Tipps** Erfüllt

Dem Benutzer werden die auf der TYH API liegende Tipps in Form von Kärtchen angezeigt.

FR08 **Tipp-Detailansicht** Erfüllt

Das detaillierte Anzeigen eines Tipps ist erfüllt. Ebenfalls kann der Benutzer darin den Tipp bewerten.

FR09 **Studien** Erfüllt

Dem Benutzer werden die auf der TYH API liegende Studien in Form von Kärtchen angezeigt.

FR10 **Studie einschreiben und verlassen** Erfüllt

Das Einschreiben und Verlassen einer Studie ist in der Studien-Detailansicht möglich.

FR11 **Studie-Detailansicht** Erfüllt

Das detaillierte Anzeigen einer Studie ist erfüllt. Ebenfalls kann der Benutzer darin sich in die Studie einschreiben, verlassen oder diesen bewerten.

FR12 **Fragebögen** Erfüllt

Dem Benutzer werden die auf der TYH API liegende Fragebögen in Form von Kärtchen angezeigt. Diese sind in der Studien-Detailansicht und im Fragebogen-Tab zu finden.

FR13 **Benachrichtigungen aktivieren (Notifications)** Teilweise erfüllt

Für wiederkehrende Fragebögen können Benachrichtigungen aktiviert und deaktiviert werden. Aufgrund der Einschränkung der genutzten Erweiterung (*flutter_local_notifications*) können nur tägliche und wöchentliche Erinnerungen aktiviert werden.

FR14 **Fragebogen-Detailansicht** Erfüllt

Das detaillierte Anzeigen eines Fragebogens ist erfüllt. Ebenfalls kann der Benutzer für wiederkehrende Fragebögen Benachrichtigungen aktivieren/-deaktivieren. Falls sich in die jeweilige Studie eingeschrieben worden ist, kann der Fragebogen ausgefüllt und bewertet werden.

FR15 **Fragebogen ausfüllen** Teilweise erfüllt

Der Benutzer kann ausschließlich den Fragebogen für koronare Herz-Risiko-Patienten ausfüllen, da nur die dort verwendeten Fragetypen vom System unterstützt werden.

FR16 **Benachrichtigung Übersicht** Erfüllt

Es wurde eine Übersicht für die aktivierten Benachrichtigungen implementiert. Der Benutzer kann in dieser auf den jeweiligen Fragebogen gelangen oder die Benachrichtigung deaktivieren.

FR17 **Profilverwaltung** Erfüllt

Das Ändern vom Benutzernamen, Vornamen, Nachnamen und Geschlecht sind in der Profilansicht möglich. Ebenfalls kann das erstellte Konto gelöscht werden.

FR18 **Aktivitäten anzeigen** Erfüllt

Dem Benutzer werden die persönlichen Aktivitäten, die auf der TYH API liegen, in Form von Kärtchen angezeigt.

FR19 **Feedbacks** Erfüllt

In *Meine Feedbacks* wird dem Benutzer eine Übersicht aller ausgefüllten Fragebögen, zu denen es ein Feedback existieren könnte, in Form von Kärtchen angezeigt.

FR20 **Evaluationen** Erfüllt

Der Benutzer kann die Evaluation eines Feedbacks sich anzeigen lassen. Falls die Evaluation leer sein sollte, erscheint eine entsprechende Meldung.

FR21 **Einstellungen** Erfüllt

Derzeit werden fünf verschiedene Themes angeboten, die der Benutzer nutzen kann. Dabei wird das geänderte Theme auf dem Gerät gespeichert.

FR22 **Über die App** Erfüllt

Die Anforderung wird in der *Über diese App* Ansicht erfüllt. Dort werden die Hintergrundinformationen über die App angezeigt.

8.2 Nicht-funktionale Anforderungen

NFR01 **Benutzerfreundlichkeit** Teilweise erfüllt

Bei einer Interaktion bekommt der Benutzer visuellen Feedback. Jedoch werden nicht alle Gestensteuerungen unterstützt, welche die meisten Benutzer von anderen Applikationen kennen. Ein Beispiel dafür ist das Wechseln der Tabs durch Wischen.

NFR02 **Effizienz** Erfüllt

Die Entlastung der TYH API und der Anfragedauer auf dem Gerät wird durch das Speichern der Daten in den entsprechenden Cubits erreicht. Neue Anfragen werden nur erstellt, sobald eine Interaktion vom Benutzer getätigt wird, welche die Daten auf der TYH API ändern.

NFR03 **Verlässlichkeit** Erfüllt

Es wird auf jede Eingabe vom Benutzer reagiert und die Inhalte der JSON korrekt dargestellt. Bei einem System- oder der TYH API Fehler werden dem Benutzer entsprechende Fehlermeldungen angezeigt.

NFR04 **Erweiterbarkeit** Erfüllt

Die Anwendung kann erweitert werden, ohne die grundlegende Struktur zu ändern. Dabei können neue Fensteransichten im Drawer hinzugefügt werden. Die Fragen im Fragebogen werden dynamisch generiert, wodurch das Implementieren der restlichen Fragetypen die Unterstützung der App für andere Fragebögen ermöglichen kann. Eingesetzte Texte in der UI sind nicht statisch, sondern durch Variablen eingefügt. Die Variablen wurden in einer externen Datei deklariert, wodurch das Erweitern von anderen Sprachen ohne viel Aufwand möglich ist. Dieses Prinzip wird in Flutter *Internationalizing*[1] genannt.

NFR05 **Portierbarkeit** Erfüllt

Das Flutter Framework ermöglicht die Generierung von APK-Dateien für Android und IPA-Dateien für iOS. Mit diesen Dateien können Benutzer einfach und mit minimalem Aufwand die App auf das entsprechende Gerät installieren.

NFR06 **Wartbarkeit** Teilweise erfüllt

Während der Entwicklung ist die Modularisierung teilweise nicht eingehalten worden. Aus diesem Grund könnte es zu Komplikationen in der Wartbarkeit der Anwendung kommen.

NFR07 **Systemunabhängige Benutzung** Erfüllt

Das Flutter Framework baut die UI durch die enthaltene Engine. Dabei werden die Positionen der Widgets mit relativen Angaben angegeben. Dadurch passen sich je nach Bildschirmgröße die Widgets dynamisch an die UI an. Folglich sieht die App unabhängig vom Betriebssystem auf allen Gräten identisch aus und bietet dieselben Funktionen.

[1] `https://flutter.dev/docs/development/accessibility-and-localization/internationalization` (25.08.2021)

9 Fazit

In diesem Kapitel wird zunächst der Inhalt der Arbeit zusammengefasst. Anschließend folgt ein Ausblick über die möglichen Erweiterungen und Verbesserungen der entstandenen mobilen Anwendung.

9.1 Zusammenfassung

In Kapitel 1 wurden die Motivation und das Ziel der Arbeit beschrieben. Dabei wurde auf die Wichtigkeit der Smartphone-Branche im Bereich des Gesundheitswesens eingegangen und der Grund vom Einsatz eines Cross-Platform-Framework in der Entwicklung der mobilen Anwendung erläutert. Nachfolgend wurden in Kapitel 2 die Grundlagen eingeführt, die im späteren Verlauf der Arbeit benötigt wurden. Dazu zählen die Grundlagen der koronaren Herzkrankheit, REST APIs, das HTTP-Protokoll und die unterschiedlichen Arten von mobilen Anwendungen. Zusätzlich wurde im Unterabschnitt 2.4.3 das Flutter Framework vorgestellt, mit welchem die Applikation implementiert worden ist. Anschließend wurde ein kurzer Überblick über verwandte Arbeiten gegeben, die ebenfalls ähnliche Anwendungen entwickelt haben, um Fragebögen mit der TYH API auszufüllen.

In Kapitel 4 wurden die Anforderungen an das zu entwickelnde System definiert, die sich in funktionale und nicht-funktionale Anforderungen untergliedern. Aufbauend auf die Anforderungen wurde der User-Interface-Entwurf in Form von Prototypen in Kapitel 5 vorgestellt. In Kapitel 6 wurde die Architektur beschrieben. Diese beinhaltet den Datenaustausch zwischen der TYH API und der Anwendung sowie das verwendete BLoC Pattern während der Implementierung. Darauf aufbauend wurden in der Implementierung eingesetzten Technologien, Werkzeuge, Bibliotheken und Erweiterungen in Kapitel 7 aufgeführt, als auch die endgültig realisierte Appli-

kation vorgestellt. Abschließend wurden die Anforderungen aus Kapitel 4 mit der umgesetzten Applikation in Kapitel 8 verglichen.

9.2 Ausblick

Die entstandene Applikation könnte in einigen Aspekten erweitert und verbessert werden, welche aus Zeitgründen nicht realisiert wurden. Eine Erweiterungsmöglichkeit ist die Unterstützung von mehreren Sprachen. Dabei muss die bereits existierende Datei mit den verwendeten Texten dupliziert und in die gewünschte Sprache übersetzt werden. Ebenfalls muss das Ändern der Sprache innerhalb der Anwendung implementiert werden. Des Weiteren bietet die TYH API die Nachrichten und Notizen Funktion an, die in dieser Arbeit nicht realisiert worden sind.

Zusätzlich könnte durch eine Benutzerumfrage die Benutzerfreundlichkeit evaluiert und entsprechend verbessert werden. Zur Messung könnte dabei das *System Usability Scale* von *Jon Broke* verwendet werden. Dabei bekommen die Benutzer Aufgaben, die sie mit dem System lösen müssen. Anschließend werden 10 Fragen über das das System bezüglich der Benutzung gestellt [5].

Eine weitere Verbesserung der Anwendung wäre das Unterstützen der Gestensteuerung, die in Abschnitt 8.2 erwähnt wurde. Derzeit unterstützt die Applikation für wiederkehrende Fragebögen nur tägliche und wöchentliche Benachrichtigungen, da die verwendete Erweiterung keine monatlichen Benachrichtigungen unterstützt. Somit könnte beobachtet werden, ob die verwendete Erweiterung in diesem Kontext eine Aktualisierung bekommt oder ob es sich andere Lösungsansätze finden lassen, die das Problem lösen würde.

Literatur

[1] Gerd Assmann, Paul Cullen und Helmut Schulte. „Simple scoring scheme for calculating the risk of acute coronary events based on the 10-year follow-up of the prospective cardiovascular Munster (PROCAM) study“. In: *Circulation* 105.3 (2002), S. 310–315.

[2] Atul R, Rahul Gaba, Kakul Gapta. *React Native Internals*. 2018. URL: https://github.com/react-made-native-easy/book/blob/master/3-react-native-internals/3.1-react-native-internals.md (besucht am 06. 08. 2021).

[3] Hanns-Wolf Baenkler. *Innere Medizin: 299 Synopsen, 611 Tabellen*. Georg Thieme Verlag, 2001, S. 119–120.

[4] Engineer Bainomugisha u. a. „A survey on reactive programming“. In: *ACM Computing Surveys (CSUR)* 45.4 (2013), S. 1–34.

[5] Aaron Bangor, Philip T Kortum und James T Miller. „An empirical evaluation of the system usability scale“. In: *Intl. Journal of Human–Computer Interaction* 24.6 (2008), S. 574–594.

[6] Robert Battle und Edward Benson. „Bridging the semantic Web and Web 2.0 with representational state transfer (REST)“. In: *Journal of Web Semantics* 6.1 (2008), S. 61–69.

[7] Mike Belshe, Roberto Peon und Martin Thomson. *Hypertext transfer protocol version 2 (HTTP/2)*. 2015.

[8] Salma Charkaoui, Zakaria Adraoui u. a. „Cross-platform mobile development approaches“. In: *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*. IEEE. 2014, S. 188–191.

[9] Bloc Community. *BLoC library documentation*. URL: https://bloclibrary.dev/#/ (besucht am 23. 08. 2021).

[10] Flutter Community. *Desktop shells*. URL: https://github.com/flutter/flutter/wiki/Desktop-shells (besucht am 06. 08. 2021).

[11] Flutter Community. *The Engine architecture*. URL: https://github.com/flutter/flutter/wiki/The-Engine-architecture (besucht am 06. 08. 2021).

[12] Ionic Community. *Ionic Documentation*. URL: https://ionicframework.com/ (besucht am 06. 08. 2021).

[13] Pascal Damasch. „Konzeption einer modernen Web-Application zur Verwaltung von Dynamischen Mobile Crowdsensing Plattformen im Healthcare Bereich“. Masterthesis. Ulm University, 2019.

[14] Ericsson. *Number of smartphone subscriptions worldwide from 2016 to 2026 (in millions)*. 2021. URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (besucht am 28. 07. 2021).

[15] Facebook. *React Native Documentation*. URL: https://reactnative.dev/ (besucht am 06. 08. 2021).

[16] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (besucht am 29. 07. 2021).

[17] Roy Fielding u. a. *Hypertext transfer protocol–HTTP/1.1*. 1999.

[18] FlutterDevs. *BLoC pattern in Flutter*. 2020. URL: https://flutterdevs.com/blog/bloc-pattern-in-flutter-part-1/ (besucht am 23. 08. 2021).

[19] The Apache Software Foundation. *Overview*. URL: https://cordova.apache.org/docs/en//latest/guide/overview/index.html (besucht am 06. 08. 2021).

[20] Bundesministerium für Gesundheit. *E-Health*. 2020. URL: https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-z/e/e-health.html (besucht am 28. 07. 2021).

[21] Martin Glinz. „On non-functional requirements“. In: *15th IEEE international requirements engineering conference (RE 2007)*. IEEE. 2007, S. 21–26.

[22] Google. *Create an Android project*. 2004. URL: https://developer.android.com/training/basics/firstapp/creating-project (besucht am 05. 08. 2021).

[23] Google. *Dart overview*. URL: https://dart.dev/ (besucht am 06. 08. 2021).

[24] Google. *Dart*. URL: https://dart.dev/overview (besucht am 06. 08. 2021).

[25] Google. *Flutter Dev and Flutter Documentation*. URL: https://flutter.dev/ (besucht am 06. 08. 2021).

[26] Google. *Material Design Documentation*. 2004. URL: https://material.io/design (besucht am 16. 08. 2021).

[27] Hugo Haas und Allen Brown. *W3C Web Services Glossary*. 2004. URL: https://www.w3.org/TR/ws-gloss/ (besucht am 29. 07. 2021).

[28] Julian Haug. „Track Your Stress: Konzeption und Realisierung einer mobilen (Android) Anwendung zur Messung des Stresslevels“. Bachelorthesis. Ulm University, 2017.

[29] Jochen Herrmann. „Konzeption und technische Realisierung eines mobilen Frameworks zur Unterstützung tinnitusgeschädigter Patienten“. Diplomarbeit. University of Ulm, 2014.

[30] Ulrike Häßler. *Der HTTP-Request: Daten vom Server holen*. 2019. URL: https://www.mediaevent.de/tutorial/http-request.html (besucht am 29. 07. 2021).

[31] Global Market Insights. *Umsatz des weltweiten Digital Health-Marktes im Jahr 2019 und Prognose für das Jahr 2026 (in Milliarden US-Dollar)*. 2020. URL: https://de.statista.com/statistik/daten/studie/1184488/umfrage/umsatz-des-globalen-digital-health-marktes/ (besucht am 28. 07. 2021).

[32] JetBrains. *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021*. 2021. URL: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (besucht am 06. 08. 2021).

[33] William Jobe. „Native Apps vs. Mobile Web Apps.“ In: *International Journal of Interactive Mobile Technologies* 7.4 (2013).

[34] Shaun Kelly und Tyler Gregory. „Typography in human-computer interaction“. In: *Group* (2011).

[35] Youn-Kyung Lim, Erik Stolterman und Josh Tenenberg. „The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas“. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 15.2 (2008), S. 1–27.

[36] Ivano Malavolta. „Beyond native apps: web technologies to the rescue! (keynote)“. In: *Proceedings of the 1st International Workshop on Mobile Development*. 2016, S. 1–2.

[37] Mark Masse und API REST. *Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. 2011.

[38] World Health Organization u. a. *mHealth New horizons for health through mobile technologies*. 2011. URL: https://www.who.int/goe/publications/goe_mhealth_web.pdf (besucht am 28. 07. 2021).

[39] Felipe Pezoa u. a. „Foundations of JSON schema“. In: *Proceedings of the 25th International Conference on World Wide Web*. 2016, S. 263–273.

[40] Kevin Ports u. a. *Stencil: A Compiler for Web Components and High Performance Web Apps*. URL: https://stenciljs.com/docs/introduction (besucht am 06. 08. 2021).

[41] Thomas Probst u. a. „Does tinnitus depend on time-of-day? An ecological momentary assessment study with the “TrackYourTinnitus” application“. In: *Frontiers in aging neuroscience* 9 (2017), S. 253.

[42] Thomas Probst u. a. „Emotional states as mediators between tinnitus loudness and tinnitus distress in daily life: Results from the “TrackYourTinnitus” application“. In: *Scientific reports* 6.1 (2016), S. 1–8.

[43] Rüdiger Pryss u. a. „Mobile crowd sensing in clinical and psychological trials–a case study“. In: *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*. IEEE. 2015, S. 23–24.

[44] Rüdiger Pryss u. a. „Mobile crowd sensing services for tinnitus assessment, therapy, and research“. In: *2015 IEEE International Conference on Mobile Services*. IEEE. 2015, S. 352–359.

[45] Rüdiger Pryss u. a. „Mobile crowdsensing for the juxtaposition of realtime assessments and retrospective reporting for neuropsychiatric symptoms“. In: *2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE. 2017, S. 642–647.

[46] Rüdiger Pryss u. a. „Mobile crowdsensing services for tinnitus assessment and patient feedback“. In: *2017 IEEE International Conference on AI & Mobile Services (AIMS)*. IEEE. 2017, S. 22–29.

[47] Reed u. a. „User interface guidelines and standards: progress, issues, and prospects“. In: *Interacting with Computers* 12.2 (1999), S. 119–142.

[48] Winfried Schlee u. a. „Measuring the moment-to-moment variability of tinnitus: the TrackYourTinnitus smart phone app“. In: *Frontiers in aging neuroscience* 8 (2016), S. 294.

[49] Michael Schrempp. „Konzeption und Realisierung einer mobilen Anwendung zur Erfassung des Stresslevels am Beispiel von iOS“. Bachelorthesis. Ulm University, 2017.

[50] Krishna Shingala. „Json web token (jwt) based client authentication in message queuing telemetry transport (mqtt)“. In: *arXiv preprint arXiv:1903.02895* (2019).

[51] StatCounter. *Mobile Operating System Market Share Worldwide 2020*. 2020. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2020-2020-bar (besucht am 28. 07. 2021).

[52] TLV Ulbricht und DAT Southgate. „Coronary heart disease: seven dietary factors“. In: *The lancet* 338.8773 (1991), S. 985–992.

[53] Kathy Walrath und Seth Ladd. *Dart: Up and Running: A New, Tool-Friendly Language for Structured Web Apps*. Ö'Reilly Media, Inc.", 2012.